Björn Regnell
Inge van de Weerd
Olga De Troyer (Eds.)

LNBIP 80

Software Business

Second International Conference, ICSOB 2011
Brussels, Belgium, June 2011
Proceedings

Springer

Björn Regnell
Inge van de Weerd
Olga De Troyer (Eds.)

Software Business

Second International Conference, ICSOB 2011
Brussels, Belgium June 8-10, 2011
Proceedings

Springer

Volume Editors

Björn Regnell
Lund University
Department of Computer Science
221 00 Lund, Sweden
E-mail: bjorn.regnell@cs.lth.se

Inge van de Weerd
Utrecht University
Department of Information and Computing Sciences
3508 TB Utrecht, The Netherlands
E-mail: i.vandeweerd@cs.uu.nl

Olga De Troyer
Free University of Brussels
Department of Computer Science
1050 Brussels, Belgium
E-mail: Olga.DeTroyer@vub.ac.be

ISSN 1865-1348 e-ISSN 1865-1356
ISBN 978-3-642-21543-8 e-ISBN 978-3-642-21544-5
DOI 10.1007/978-3-642-21544-5
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2011928548

ACM Computing Classification (1998): K.6, D.2

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

The Second International Conference on Software Business (ICSOB 2011) was organized in Brussels, Belgium: the heart of Europe and administrative center of the European Union. The conference theme “Managing Software Innovation for Tomorrow’s Business” reflects the specific challenges in the research domain of software business. Although the software business shares common features with other international knowledge-intensive businesses, it carries many inherent features making it a challenging domain for research. The goal of the ICSOB conference series is to bring together researchers with a specific focus on this domain.

We received 27 scientific paper submissions in various categories. Each submission was reviewed by at least three members from the Program Committee. During the Program Committee meeting, held in Utrecht, The Netherlands, 12 participants from various countries discussed the papers and their reviews. The committee decided to accept 14 papers, yielding an overall acceptance rate of 48%.

We scheduled three days with scientific paper sessions. These sessions were complemented with an Industry Track, containing best-practice presentations and discussions with practitioners from the software industry. Furthermore, each day started with an excellent keynote speaker to challenge us with new perspectives. The conference was preceded by two workshops: one focusing on software Ecosystems (IWSECO 2011) and the other one on leveraging empirical research results for software business success (EPIC 2011).

ICSOB 2011 was created by the hard work of the different committees and chairs. As the Program Chairs for ICSOB 2011, we thank all members of these committees and chairs for their dedication and effort and our corporate sponsors for their support. A special thanks goes to the members of the Program Committee for their careful and timely reviews, in particular those who attended the Program Committee meeting in Utrecht, physically or virtually, and those who volunteered as shepherds to help improve promising papers.

April 2011

Björn Regnell
Inge van de Weerd

Organization

Chairs and Committees

Program Chairs

Björn Regnell	Lund University, Sweden
Inge van de Weerd	Utrecht University, The Netherlands

Local Organizing Chair

Olga De Troyer	Vrije Universiteit Brussel, Belgium

Industry Track Chair

Sjaak Brinkkemper	Utrecht University, The Netherlands

Review and Publication Chair

Kevin Vlaanderen	Utrecht University, The Netherlands

Publicity Chair

Eetu Luoma	University of Jyväskylä, Finland
Oleksiy Mazhelis	University of Jyväskylä, Finland

Steering Committee

Kalle Lyytinen	Case Western Reserve University, USA
Sjaak Brinkkemper	Utrecht University, The Netherlands
Pekka Abrahamsson	University of Helsinki, Finland
Pasi Tyrväinen	University of Jyväskylä, Finland
Slinger Jansen	Utrecht University, The Netherlands

Program Committee

Aybuke Aurum	University of New South Wales, Australia
Jan Bosch	Intuit, USA
Peter Buxmann	Darmstadt University of Technology, Germany
David Callele	University of Saskatchewan, Canada
Erran Carmel	American University, USA
Michel Chaudron	Leiden University, The Netherlands
Michael Cusumano	MIT Sloan School of Management, USA
Daniela Damian	University of Victoria, Canada

Additional Reviewers

B

Barrios Albornoz, Judith
Boucher, Quentin

F

Ferrari, Remo

K

Khadka, Ravi
Khurum, Mahvish

M

Metzger, Andreas

S

Santos, Rodrigo

T

Tamburri, Damian Andrew

Table of Contents

Part 1

Keynote

Keynote: Engineering Challenges of New Business Models in Software 1
Anthony Finkelstein

Part 2

Research Papers

How to Sell SaaS: A Model for Main Factors of Marketing and Selling Software-as-a-Service 2
Pasi Tyrväinen and Joona Selin

Business Continuity Solutions for SaaS Customers 17
Tommy van de Zande and Slinger Jansen

Software Ecosystems: A Set of Management Practices for Platform Integrators in the Telecom Industry 32
Martti Viljainen and Marjo Kauppinen

Steering Insight: An Exploration of the Ruby Software Ecosystem 44
Jaap Kabbedijk and Slinger Jansen

Study of the Competition between Proprietary Software Firms and Free/Libre Open Source Software Firms Using a Simulation Model 56
Luisanna Cocco, Katiuscia Mannaro, Giulio Concas, and Michele Marchesi

Adoption of Open Source Software and Software-as-a-Service Models in the Telecommunication Industry 70
Eetu Luoma, Nina Helander, and Lauri Frank

Examining the Effects of Agile Methods and Process Maturity on Software Product Development Performance 85
Mikko Rönkkö, Juhana Peltonen, and Christian Frühwirth

Online Distribution of Packaged Software 98
Shuangzeng Hu and Rod B. McNaughton

Scenarios on Adoption of Open Source Software in the Communications Software Industry 110
Eetu Luoma, Mikko Riepula, and Lauri Frank

Improving Quality and Cost-Effectiveness in Enterprise Software Application Development: An Open, Holistic Approach for Project Monitoring and Control 125
Luigi Buglione, Ernesto Damiani, Fulvio Frati, Sergio Oltolina, and Gabriele Ruffatti

Transformations of a Solution Strategy: A Case Study 140
Marko Komssi, Marjo Kauppinen, Matti Ropponen, and Pirkka Palomäki

The Sun Also Sets: Ending the Life of a Software Product 154
Slinger Jansen, Karl Michael Popp, and Peter Buxmann

Requirements Scoping Visualization for Project Management 168
Krzysztof Wnuk and David Callele

Variability-Based Release Planning 181
Samuel Fricker and Susanne Schumacher

Part 3

Workshops

EPIC 2011: Third Workshop on Leveraging Empirical Research Results for Software Business Success 187
Maya Daneva and Andrea Herrmann

IWSECO 2011: Third International Workshop on Software Ecosystems 188
Slinger Jansen, Jan Bosch, Faheem Ahmed, and Piers Campbell

Author Index 189

Keynote: Engineering Challenges of New Business Models in Software

Anthony Finkelstein

Computer Science, University College London
London, UK

The software engineering research agenda has largely emerged from experience of software development in the context of established software business models. Thus, principally bespoke software development or more rarely 'product' software development - unitary software delivered under a product licensing agreement to a market. The reshaping of the software business through the introduction of new ways of 'delivering' software with associated business models must necessarily lead us to reconsider the software engineering agenda. Thus applications offered 'as a service', 'apps' with associated 'channels', 'appliances', fine-grain software services and so on each have associated with them unique engineering challenges which need to be addressed.

In this talk I will examine the prospects of each of these approaches, review the associated challenges and set them in the context of broader research directions for software engineering. I will suggest that not only will the content of software engineering research need to change but that the way in which software engineering research is conducted will need to change also. I will introduce the idea, drawn from biomedicine of the 'translational pipeline' and explain how such a pipeline can be implemented. I will describe particular experience with the development of an 'application as a service' offering and highlight some contrasts between that experience and that of developing software within a more conventional product business model.

How to Sell SaaS: A Model for Main Factors of Marketing and Selling Software-as-a-Service

Pasi Tyrväinen and Joona Selin

Department of Computer Science and Information Systems
Agora, P.O. Box 35
FIN-40014 University of Jyväskylä, Finland
{pasi.tyrvainen,joona.v.selin}@jyu.fi

Abstract. Software-as-a-Service providers have been growing fast while the contemporary research literature has neglected analysis of their business-critical marketing and sales processes. In this paper we collect the key factors characterizing how to market and sell SaaS to business customers into an eight dimensional model. We also use an explorative multi-case study to observe six SaaS providers and validate the model. The interviewed providers emphasized use of the Internet for marketing communication while personal direct sale was the dominating sales approach. Customer acquisition cost was the key performance indicator for marketing and sales while customer lifetime value and churn were the KPIs in customer relationship management.

Keywords: Sales and marketing models, business models, software business, Software-as-a-Service, SaaS, key performance indicators, KPI.

1 Introduction

Youseff, Butrico and Da Silva [1] defined Cloud computing as services used on demand through networks. They divide cloud services into five layers: cloud applications, cloud software environment, cloud infrastructure, software kernel and firmware/hardware. The most commonly known form of cloud applications is Software-as-a-Service (SaaS), where an application is used cross the network without installing it into the user device [2]. According to Gartner [3], the SaaS market will grow annually 17.7% till year 2013. For software firms this will mean new business opportunities as well challenges in adapting to the new business environment.

A transition from a packaged software business provider to a SaaS provider following the best practices of the SaaS model will set new challenges to strategic management, but also to marketing and sales. So far the research literature has not addressed this problem while practitioners have published a volume of related material in the Web. Kaplan [4] and Mallya [5] have focused on the differences between marketing and selling traditional software and SaaS. An alternative approach emphasizes providing a solution to the customer rather than selling SaaS as such [6]. Domergue [7] focuses on providing value to the customer when selling SaaS. In general, some of the previous studies emphasize ease of selling SaaS [8] while others speak about difficulty [9].

Our purpose is to initiate academic research on marketing and selling SaaS by presenting a model of the key factors for marketing and selling SaaS and related key performance indicators (KPI). In this paper we describe, how software products can be marketed and sold as services in business-to-business (B2B) markets according to the SaaS model. Later research can focus on the individual factors as well as to study the relationships between the factors. In section 2 of this paper we summarize shortly a literature review of the related background theories on marketing, software business and SaaS. In section 3 we construct the model for the main factors determining how a firm markets and sells a SaaS offering based on the literature. Empirical research in section 4 is used for exploring the field and for validating the model in a multi-case study of six companies selling SaaS. In the end, we present a revised mode adopting the changes implied by the empirical observations.

2 Marketing and Sales in Software Business

Kotler and Keller [10] define marketing as a function and process of the organization, which creates, communicates and delivers value to customers and maintains customer relationships by means profitable to the firm and interest groups. One of the functions included is selling aiming at completing the sales case [11]. At the same time, marketing tries to make sales unnecessary by providing self-selling products, which the customers are willing to buy [10].

The two main approaches to marketing are transactional marketing and relationship marketing. From the transactional marketing viewpoint the firm can compete with four Ps, product, price, place, and promotion [12]. According to the relationship marketing approach, marketing is an interactive process, which builds, maintains and develops relationships, which comply with the goals of the participants [13]. Kotler and Keller [10] present a holistic view, which integrates the transactional view under the tile integrated marketing with the relationship view, and in addition, internal marketing and performance marketing. The integrated marketing addresses questions related to products and services, delivery channels as well as communication. Relationship marketing includes customers, partners and delivery channel related questions. Internal marketing focuses on firm internal marketing between the marketing department, other department and top management. Performance marketing focuses on revenue, brand value and ethical and legal operating environment. In this study we focus on questions related to integrated marketing which will match the expected low transaction costs of SaaS offerings as well as relationship marketing.

Customers have traditionally been divided into consumers and businesses, out of which we focus only on businesses in this study. Each firm can create applicable means to divide target customers into market segments sharing similar needs [11]. Criteria used can include demographic factors, such as vertical industry [14], firm size and location. Also technologies and practices used, purchasing practices and firm-specific characteristics can be used for segmenting [10] as well as customer lifetime value [15]. Following Berger and Nasr [16] we define customer lifetime value as the surplus of long-term income from customer reduced by customer relationship maintenance costs. This can be used for identifying profitable relationships but also for evaluating potential future customer base [17].

Software business includes segments with rather different characteristics with respect to marketing and sales. Embedded software is usually developed for a single company either as professional services or in-house. IT firms providing professional services implement bespoken systems as well as deploy and tailor enterprise systems for the customer. In professional service business trust is important as the software to be delivered does not yet exist [18]. The number of customers is small while transaction costs and the revenue per customer are high [19], which require investing on customer relationship management [18]. Instead, standardized and packaged software products with relatively low prices are examples of offerings of Internet-generation firms with strong brand marketing and marketing alliances [18]. The customer-base is large and the users are distant to the vendors [19]. For these software product firms the marketing costs are a major share of the budget [20].

The relevant marketing means for software service business include relationship management, seminars, fairs and other form related to personal communication, software product business relies more on advertising and direct sales while both use Internet as a marketing channel [19]. Personal selling, representatives and value-added-resellers (VAR) are typical sales channels for software service business. A network of software service firms can also co-produce a service offering for customers in process of value co-creation following the service-dominant logic. Instead, software product businesses use more wholesale and resale organizations as well as Internet as a sales channel [19]. In the international markets software service firms cooperating closely with customers tend to use representatives in the market, whereas firms developing semi-standardized enterprise solutions prefer sales subsidiaries of their own and firms offering mass-market products to consumers tend to choose cooperative entry modes where local organizations possess deep knowledge about the target market [21].

Software-as-a-Service (SaaS) can be characterized as a standard software product operated by the SaaS provider, delivered using standard Internet protocols and consumed as on-demand services by the customers, typically using Web browsers as the user interface. For the purposes of the empirical study we combined the following criteria for compliance with the SaaS model from multiple sources (including [1] [2] [22] [23]):

1. Software is used with a Web browser or other thin client making use of standard internet protocol.
2. A standardized software product is provided with no customization.
3. There is no need to install software to the customer site.
4. Deployment requires no major integration or installation.
5. Customers pay for use of the software rather than licenses.
6. The same multitenant installation is provided for several customers.

From the user viewpoint low entry cost and pay-as-you-go pricing make adoption and use of SaaS attractive. From customer's perspective SaaS can be seen as outsourcing IT back-end management activities to the provider. From SaaS vendor viewpoint SaaS can be viewed as a software delivery and deployment model. It can also be seen as a business model sharing characteristics with software product business with high number of customers and low transaction costs. From this perspective the multitenant software can be seen as a cost-efficient model to provide services to new, underserved market segments, such as small enterprises [22]. SaaS shares also some characteristics of software services business with a need to avoid churn and invest on customer relationship management to retain customers.

3 Marketing and Sales Model for SaaS

Based on the literature it seems that providing SaaS services is technically cost efficient, but controlling the marketing and sales costs will be a major challenge for profitability of SaaS providers. The main factors of marketing and selling SaaS collected from the literature have been collected to the model presented in Figure 1. The model consists of eight dimensions representing eight variables; service provider size, service and implementation model, customer size, market communication channel, sales channel, role of buyer, entry transaction size, and customer life-cycle value. Based on the literature we expect the categories in the middle of the diagram are likely to co-occur and the categories in the outer rings are likely to co-occur. For example, a small SaaS service provider would be more likely to provide self-services through the internet to end users with small entry cost while a large service provider is likely to provide added value services to large customers.

Provider size and customer size dimensions in this model are categorize based on firm headcount to micro (less than 10 persons), small (10-49), mid-size (50-249) and large enterprises. According to Moore [24] firm size has major impact to the markets it will operate. The target market of a provider is defined by the target customer group, which is strongly related to the offering of the provider [19]. Zoltners, Sinha and Lorimer [25] state further, that a firm has to be able to organize the sales according to the status of the market including appropriate resourcing and size of the sales organization.

Service and implementation model describes the product strategy of the firm and the role of services and implementation in the business model. It also reflects the share of sources of income in the business. The categories in this dimension are self service and professional services including deployment, integration, tailoring, as well as training and consulting (adopted from [26]).

Customer size dimension in this model reflects the group of target customers. The categories follow the categories of the providers. According to Choudhary [27] and Sääksjärvi et al. [28] the target customer groups of SaaS providers span the full range from small enterprises to large ones. The chosen target customer group will impact the product strategy [19], the channels of market communication [11] as well as the sales channels [22].

Market communication channel represent the means of a SaaS provider to deliver information about the service to the customer [19] providing the means to increase the sales [11]. The marketing and sales channel solutions are thus tightly interconnected. Software product companies have typically high costs for creating brand awareness and recognition due to aggressive advertising, promotion and relationship building activities [18]. Use of Internet has been emphasized as a means to replace these traditional channels for SaaS firms [29] potentially providing some relief to the costs.

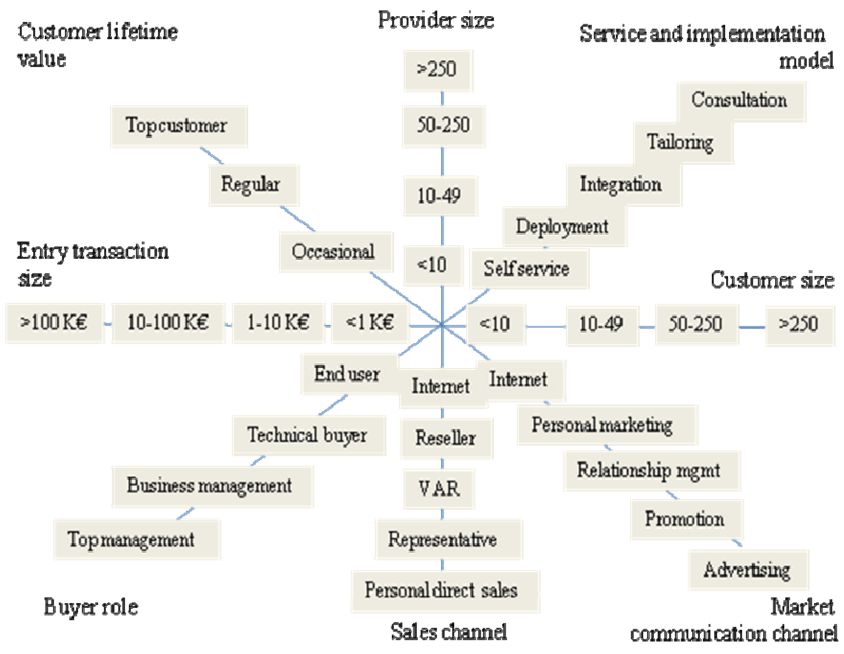


Fig. 1. The main factors of marketing and selling SaaS collected from the literature

Sales channel dimension describes the sales solutions of a SaaS firm. That is, how the firm aims to complete the sales transactions. Chong and Carraro [22] mention use of Internet and direct personal sales as a sales channel for SaaS while Weobong [30] adds resellers, VAR and other channel partners, which are here referred to as "representatives".

Role of buyer of SaaS is shifting from technical buyers to business directors [31]. Moore [32] divides roles of buyers into top management, business management, technical buyers, and end users. He further states that the role of buyer shifts according to the life cycle of the product.

Entry transaction size will have major impact to adoption of SaaS in many cases. On one hand, the low cost of first transaction compared to traditional software licenses has commonly been referred as one of the customer benefits of SaaS [33]. On the other, the mismatch of cash flows is one of the challenges of SaaS providers [34]. By this Gardner refers to a situation, where the sales and marketing costs materialize at the sales event while the income realizes monthly during the contract period.

Customer lifetime value dimension represents two roles in this model. It can be used for value of existing and potential customer relationships. The costs of marketing communication and sales allocated to the new customers will have major impact to the lifetime value of a customer. Thus the transaction costs will set constraints to marketing communications and sales. [16] The categories for this dimension are based on the categories of Tähtinen and Parvinen [19].

4 Multi-case Study

The primary goals of this empirical work are to explore the target domain, which has not been researched earlier as well as to evaluate the marketing and sales model for SaaS constructed based on the literature. We first describe the methodology chosen, next, present the results, and finally, analyze the model. Based on the analysis we present an updated model adopting the changes implied by this empirical part.

4.1 Research Process and Methods

Due to explorative nature of the research we chose multi-case study as the research method. Case study match well with research of processes and provides detailed and intensive data from small set of relate cases [35]. We chose thematic interviews as the main data collection method due to the new and unexplored nature of the topic. This enables also refining the questions and answers if needed. Validity of interview data can be verified from complementary data sources and interviews can be used for exploring relationships of phenomena and for creation of new hypothesis [36]. Complementary data was collected from the Web pages and annual statements of the interviewed companies to verify and complement the interview data.

The target group of interviewees was sales and marketing managers representing SaaS providers in Finland. Within this target group we aimed at finding as heterogeneous set of firms as possible to include firms with a variety of SaaS business models. We used the National Software Industry Survey [23] and Web pages of the Cloud Software Program [37] as sources of potential SaaS providers. With the resources available, we chose seven firms providing SaaS in B2B markets. We determined the managers representing the firms after approaching them by email followed by a phone call. One of the firms did not answer to our contacts, thus the final set of interviewees included persons from six firms. The roles of the six interviewees include CEO, service manager, sales director, product manager and executive advisor. All the interviewees had the deep understanding of SaaS business needed for carrying out the interviews.

We sent an introductory letter to the interviewees with a 1.5 pages questioner annex prior to the interviews to give the persons a chance to get familiar with the questions. The questioner annex contained a listing of the topic areas and 3-5 interview questions under each topic area. The topic areas were formed from the dimensions of the model and the required background data: background information of the interviewed firm and person (summarized in Table 1), SaaS offering (compliance with the SaaS criteria in section 2, implementation and delivery model, networks), customers, sales process, marketing communication, sales financials, customer lifetime value, and summary questions.

We tested the use of the interview questions in a pre-interview and on this basis implemented minor modifications to improve the fluency of the later interviews. These changes have no major impact to the results and thus the results of the pre-interview are included in the results. All the interviews took place during August-September 2010, four in firm premises, two were conducted by phone. The average duration of the interviews was 1 hour and 11 minutes. The interviews were digitally recorded. Chosen segments of the recordings were transcript to written documents on the first pass and verified on the second pass soon after the interviews. The written documents were sent for interviewees for comments, corrections and additions. The collected data was classified thematically based on the model presented in Figure 1.

4.2 Characteristics of the Case Firms

Table 1 presents characteristics of the firms whose representatives we interviewed. The headcount figures represent the number of employees in Finland. Number of years the firm has conducted SaaS business is reflected in their business model. Firms A, B, and C have conducted SaaS business since they were established while firms D, E, and F have started providing SaaS later on. The annual revenue 2009 refers to the total business revenue including both SaaS and other business. SaaS revenue share includes revenue from SaaS service fees and related added value services for five firms. This share was not recorded and could not be estimated by firm D. Similarly, firm B did not disclose profitability data, but is still in early phase, like firm A. Compliance with SaaS model is estimated based on the criteria presented earlier.

Table 1. Firm characteristics and background information of the firms

Firms	Firm A	Firm B	Firm C	Firm D	Firm E	Firm F
Business model	ASP / SaaS	SaaS	SaaS	Professional SW services	Professional SW services	Software product
Interviewee role	Managing Director	Sales Director	Managing Director	Executive Advisor	Service Director	Product Manager
Headcount / in sales	<10 20%	<10 25%	10-49 52%	>250 3,5%	50-250 7,5%	>250 42%
Years in SaaS business	2	1	11	5	1	3
Revenue 2009 (change from prev)	<1M€ (+40%)	<1M€ (N/A)	1-10M€ (+7%)	>1000M€ (-9%)	11-100M€ (N/A)	101-1000M€ (+11%)
SaaS share from revenue	93%	100%	100%	na.	15%	50%
Profit 2009 (change from prev)	-90% (na.)	na.	21% (+19%)	12% (-2%)	13% (na.)	23% (-1%)
Compliance with SaaS definition	3/6	6/6	6/6	4/6	6/6	5/6
Marketing & sales cost per revenue	50%	40%	60%	na.	na.	41%

4.3 Detailed Results

This sub-section presents the results organized according to the factors impacting marketing and selling SaaS presented in the model in section 3. Table 2 summarizes the results representing the factors and their values in rows. Each column from A to F represents the results of one firm.

The first set of rows represents the alternative *components of service and implementation models*. Half of the firms – A, B and F – deliver SaaS as self-service while others provide some level of deployment, training and other services. Firm D has positioned itself as a provider of high added value services. Firm E embeds the SaaS offering in a bundle provided for a single fee, but not as a self-service package.

> *"We do not want to consult ourselves, we want our partners to do it."* – Sales Director, Firm B.

The second set of rows describes *service and implementation model* of the interviewed SaaS providers by presenting the split of revenue between SaaS fees and professional services. The use of self-service is visible as a high share of revenue from SaaS in Firms B, C and F. Firm D did not provide an estimate share of the share of SaaS fees from the total revenue. The content of professional services provided varies according to the firm. Firm A provides specification, deployment and consulting services and these form 75% of their revenue. Firm B gets only 5% from services containing mainly training. Firm C divides services into training, deployment projects and after sales support. There are also differences between the firms in using external resources. Firms D and F do not use partners for service creation while A, B and C use technology partners for providing hosting services and platforms for providing SaaS services. Firms B and E use also external R&D resources.

Customer and provider sizes are represented in the same set of rows with "C" denoting size of the Customer organizations in the main customer groups and "P" denoting Provider size. The sizes of customer follow nicely the provider sizes. The small SaaS Firms A and B targeted small customers with less than 50 employees while the larger providers have targeted mid size and large customers. For Firm F the size of customer is less relevant while for others it pays an important role.

> *"In this kind of services the main common factor in firm purchase behavior is the number of employees."* – CEO, Firm C.

The buyer in small firms is usually a top management. In medium and large companies the business managers buy SaaS services matching their needs and departmental budgets.

> *"Business is more interested in SaaS... customer's IT organization feels threatened by new solutions."* – Executive advisor, Firm D.

The next set of rows is the *market communication channels* of the interviewed SaaS providers. In addition to traditional marketing means all the interviewees used Internet for marketing their SaaS offering, e.g. using Web pages, targeted e-mail campaigns, newsletters, search marketing, viral campaigns, banners etc.

> *"We have tried to live in the world, that when purchases are made in the Web then also visibility will be in the Web."* – Sales director, Firm B.

The rows representing *sales channels* of the interviewed SaaS providers follow the rows representing the marketing channels. All the interviewed firms use personal selling and for Firms D and E this is the only sales channel. Other four firm use also value added resellers. Firm F used early Internet sales but gave up and uses now strongly resellers. Firm B is the only one using Internet as a sales channel, although self-service was the only service and implementation model for Firms B, C and F.

Table 2. Summary of the interview results. Each column with title from A to F represents a firm. A set of rows represents alternative categories in one dimension of the model. "X" denotes that this category applies to the firm. Customer and provider sizes are represented in the same rows with "P" denoting Provider. There are also two sets of rows for service and implementation modes, and customer lifetime value is excluded from this table.

Firms	A	B	C	D	E	F
Service components						
Self service	X	X	X			X
Deployment service	X			X	X	
Integration				X	X	
Tailoring	X			X		
Training and consulting	X			X	X	
Revenue split						
SaaS fees (%)	25%	95%	87%	na.	80%	98%
Professional services (%)	75%	5%	13%	na.	20%	2%
Customer and Provider sizes						
< 10	C P	P				C
10-49	C	C	P			C
50-250			C	C	C P	C
> 250				C P	C	C P
Buyer roles						
End user						
Technical buyer				X		
Business management			X	X	X	
Top management	X	X				
Market communication						
Internet	X	X	X	X	X	X
Personal marketing	X		X	X	X	X
Relationships	X	X	X			X
Promotion			X	X	X	X
Advertising						
Sales channels						
Internet		X				
Reseller						X
VAR	X	X	X			X
Representative						
Personal selling	X	X	X	X	X	X
Entry transaction size						
< 1 K€						X
1-10 K€	X	X	X	X		
10-100 K€					X	
> 100 K€						

> *"The market is not mature enough for self-service [through the Internet] to be an effective, rational and scalable alternative."* – CEO, Firm C.

For the companies using personal selling personal direct marketing was also important means for marketing. In addition to the sales channels listed in the literature, the interviewees mentioned new marketing and sales models, such as use of trusted recommenders or sales agents forwarding leads to the SaaS provider sales personnel. The high level of sales provisions and channel management costs were seen to prohibit use of external resellers. Typically the external resellers receive 10-50% of the first year revenue from the new customer, from the total value of the agreement or from the sales transaction.

Next set of rows presents the *entry transaction sizes* of the interviewed SaaS providers categorized based on the order of magnitude. Most of the revenue is realized soon after the sales while fees form a continuous cash-flow, which can increase based on additional users or training fees. Contract periods used include annual and three-year contracts charged typically in the beginning while continuous contracts are charged monthly.

The interviews included multiple themes related to *customer lifetime value*; customer potential evaluation, customer categorization based on the relationship and estimation of customer lifetime value. The interviews indicated that the approach created based on literature is insufficient for estimating customer lifetime value. Most of the firms used the number of potential customers as the main metrics in evaluating the potential. However,

> *"Unfortunately we cannot classify the customers to high-potential or non-potential beforehand rather than after starting the discussion"* – Sales Director, Firm B.

Customers were not classified in Firm A, while Firm B assigned points to the customers based on the market segment and target group and firm D focused on strategic customers included in a ranking list. Various metrics were used for estimating business and sales performance related to customer lifetime value. Firm A evaluated customers based on time spent and revenue. Firm C used multiple metrics: new sales per salesman, outbound calls, conversations, scheduled sales meetings, inbound customer contacts, and number of new licenses sold. Firm D did not bring up tools for estimating customer lifetime value or customer potential. Firm B defined a clear goal for the first year customer lifetime value:

> *"Customer acquisition cost should be less than customer relationship value of the first year. It would be a great ratio, if the customer would pay [us] during the first year half of what it costs us."* – Sales Director, Firm B.

4.4 Updated Model

Based on the empirical results, we found it useful to revise the model somewhat. The eight dimensions were clustered into four internally interconnected areas represented as the leaves of a four-leaf clover in Figure 2. We added the key performance indicators (KPI) to the four areas taking into account both short and long term success of the firm. These KPIs are presented next to the area outside the clover.

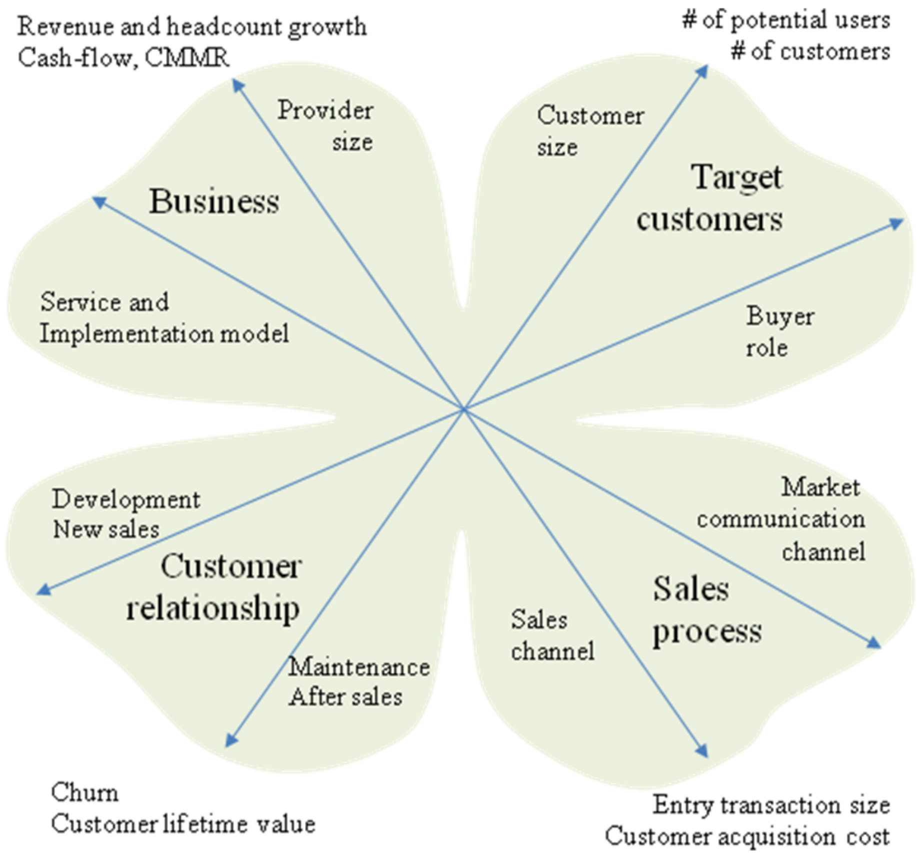


Fig. 2. The updated clover model for marketing and selling SaaS

Out of the eight dimensions two were converted into KPIs, namely entry transaction size as a KPI of Sales Process and customer life-time value as a KPI of new area Customer Relationship. This new area contains two new dimensions related to development and maintenance of customer relationship. The scales of the old dimensions are mainly the same as in Figure 1 and due to simplicity they are not presented in Figure 2. The four areas containing the updated dimensions are as follows.

- *Business* connects the dimensions Service and implementation model and Provider size. The scales of these dimensions are the same as in Figure 1. For example, Service and implementation model has values Self service, Development, Integration, Tailoring, as well as Training and Consulting. A typical SaaS firm in the empirical study was a small growth venture established especially for SaaS business. In this case the typical service and implementation model is self-service. Based on the empirical study the key performance indicators measuring best the success in business area include revenue and headcount growth, cash-flow and committed monthly recurring revenue (CMRR) from current customer base.

- *Target customers* area connects together the two interconnected dimensions of Customer size and Buyer role. The scales of these dimensions are the same as in Figure 1. As observed in the interviews, in large customer organizations SaaS is purchased by business managers while in small organizations the top management does the decisions related to purchases. The number of potential users was found to be the critical indicator when defining target markets while the number of customers was key indicator of long-term success.
- *Sales process* includes the interconnected dimensions of Sales channel and Market communication channel. Unlike expected in the original model the common practice to use of Internet marketing did not imply use of Internet as a sales channel for business customers. Instead, use of personal marketing and personal selling were connected in the empirical study. For this reason the ordering of the values in Market and communication channel dimension needs to be updated accordingly positioning personal marketing along with personal direct sales in the outer ring representing high unit costs per customer. Based on the empirical study customer acquisition cost is the most important performance indicator of sales process, as also suggested in [38]. Interviews indicate also, that entry transaction size is directly related with cash-flow and should be treated as a performance indicator of the sales process rather than as a separate dimension of the model.
- *Customer relationship* area includes two new dimensions, Development referring to new sales and Maintenance referring to after sales. The maintenance of customer relationship is important to avoid churn and guarantee continuous cash-flow from the customer base. After sales support enables also development of the relationship and generation of new sales from the same customer organization. The two key performance indicators in SaaS customer relationship management are churn and customer lifetime value.

This revised model can be utilised in design of marketing and sales strategy for SaaS services as it brings together the key dimensions and their interconnections. The form of a four-leaf clover emphasizes the interconnectedness of the factors within and in between the areas. Especially, customer lifetime value cannot be discussed outside a wider context of a customer relationship. The model also provides key performance indicators for following progress in the main areas.

From research perspective the customer relationship area of the updated model rising up from the empirical study has most development potential. This study has been focusing on the activities needed to make a customer to buy SaaS while the after sales activity gained less attention. Thus work is needed to deepen the understanding of the KPI – churn and customer lifetime value – in forthcoming research.

5 Summary and Conclusions

This paper presented a model for the main factors of marketing and selling SaaS constructed based on the literature. To our knowledge, such has not been presented in research literature prior to this paper. An explorative multi-case study was used for collecting empirical data and for validating the model. In general, the literature and the empirical observations were rather well in line although some changes were implemented into the updated model.

Based on the empirical results it seems that SaaS providers include small, medium size and large enterprises. The pure-player SaaS providers in our target group were small growth ventures, whose business model is based on providing SaaS services mainly for small enterprises based on customer self-service.

In general, the interviewed firms provided SaaS services for a wide range of firms from large corporations to small micro firms in the long tail. The dominant factor to determine the target group of customers was the number of potential users, which is directly related to the headcount of the customer organization.

The main sales channel was direct personal sales supported with Internet-based marketing communication. Internet as such was not much used as a sales channel, and advertising was not used in marketing communication.

The most important performance indicator for marketing and sales was customer acquisition cost. Customer lifetime value and churn were the key performance indicators for customer relationship management. There seems to be a need for better tools for estimating the customer lifetime value, which would be a fruitful target for further research. In addition, studying the impact of service platforms to marketing and selling SaaS would be a useful direction for further research.

In general, the studied SaaS providers resemble software product firms in having a high number of customers, small revenue per customer and high marketing and sales costs. They also share the problem of delay in cash flow between software development and revenue. The SaaS providers also share challenges of professional service providers in maintaining customer relationships and avoiding customer churn. Finding a marketing and sales approach matching these combined challenges originating from of both software product and professional services business will be critical to success of any SaaS firm in the near future.

Acknowledgments. This work was supported by TEKES (the Finnish Funding Agency for Technology and Innovation) as part of the Cloud Software Program of TIVIT (Finnish Strategic Centre for Science, Technology and Innovation in the field of ICT).

References

1. Youseff, L., Butrico, M., Da Silva, D.: Toward a Unified Ontology of Cloud Computing. In: Grid Computing Environments Workshop, pp. 1–10 (2008)
2. Vaquero, L.M., Rodero-Merino, L., Caceres, J., et al.: A Break in the Clouds: Towards a Cloud Definition. ACM SIGCOMM Computer Communication Review 39, 50–55 (2008)
3. Pettey, C., Stevens, H.: Gartner Says Worldwide SaaS Revenue to Grow 18 Percent in 2009 (2009), `http://www.gartner.com/it/page.jsp?id=%201223818`
4. Kaplan, J.M.: Software-as-a-Service Myths, `http://www.businessweek.com/technology/content/apr2006/tc20060417_996365.htm`
5. Mallya, S.: SaaS Sales Strategy, `http://www.prudentcloud.com/saas/saas-sales-strategy-25062009/`
6. Jamcracker: When Selling SaaS, Don't Sell SaaS, `http://www.jamcracker.com/When-Selling-SaaS-Dont-Sell-SaaS-0`

7. Domergue, D.: PrudentCloudSaaS: Value Based Selling, http://www.prudentcloud.com/saas/saas-value-based-selling-03082009/
8. Cone, L.: The SaaS Model - Easy to Sell, http://it.toolbox.com/blogs/coneblog/the-saas-model-easy-to-sell-18333
9. Radizeski, P.: Top 3 Reasons its Hard to Sell SAAS, http://blog.tmcnet.com/on-rads-radar/2009/03/top-3-reasons-its-hard-to-sell-saas.html
10. Kotler, P., Keller, K.L.: Marketing management. Pearson/Prentice Hall, Upper Saddle River, NJ, USA (2009)
11. Jobber, D., Lancaster, G.: Selling and sales management. Prentice Hall/Financial Times, Harlow (2003)
12. Kotler, P.: Megamarketing. Harvard Business Review 64, 117–124 (1986)
13. Grönroos, C.: Keynote Paper from Marketing Mix to Relationship Marketing-Towards a Paradigm Shift in Marketing. Management Decision 35, 322–339 (1997)
14. Tyrväinen, P.: Model for evolution of a vertical software industry. In: Tyrväinen, P., Mazhelis, O. (eds.) Vertical Software Industry Evolution Analysis of Telecom Operator Software, pp. 25–33. Springer, Heidelberg (2009)
15. Gupta, S., Hanssens, D., Hardie, B., et al.: Modeling Customer Lifetime Value. Journal of Service Research 9, 139 (2006)
16. Berger, P.D., Nasr, N.I.: Customer Lifetime Value: Marketing Models and Applications. Journal of Interactive Marketing 12, 17–30 (1998)
17. Venkatesan, R., Kumar, V.: A Customer Lifetime Value Framework for Customer Selection and Resource Allocation Strategy. J. Market. 68, 106–125 (2004)
18. Hoch, D.J., Roeding, C., Lindner, S.K., et al.: Secrets of software success. Harvard Business School Press, Boston (2000)
19. Tähtinen, J., Parvinen, P.: Ohjelmistojen markkinointi. In: Hyvönen, E. (ed.) WSOY, Vantaa, Finland, pp. 41–76 (2003)
20. Cusumano, M. A.: The business of software: What every manager, programmer, and entrepreneur must know to thrive and survive in good times and bad. Free Press, New York (2004)
21. Ojala, A., Tyrväinen, P.: Business Models and Market Entry Mode Choice of Small Software Firms. Journal of International Entrepreneurship 4, 69–81 (2006)
22. Chong, F., Carraro, G.: Architecture Strategies for Catching the Long Tail. MSDN Library, Microsoft Corporation (2006)
23. Rönkkö, M., Ylitalo, J., Peltonen, J., Koivisto, N., Mutanen, O., Autere, J., Valtakoski, A., Pentikäinen, P.: National Software Industry Survey 2009. Helsinki University of Technology, Espoo (2009)
24. Moore, G.A.: Inside the tornado: Marketing strategies from silicon valley's cutting edge. HarperBusiness (1999)
25. Zoltners, A.A., Sinha, P., Lorimer, S.E.: Match Your Sales Force Structure to Your Business Life Cycle. Harv. Bus. Rev. 84, 80 (2006)
26. Rajala, R., Rossi, M., Tuunainen, V. K.: A Framework for Analyzing Software Business Models (2003)
27. Choudhary, V.: Comparison of Software Quality Under Perpetual Licensing and Software as a Service. J. Manage. Inf. Syst. 24, 141–165 (2007)
28. Sääksjärvi, M., Lassila, A., Nordström, H.: Evaluating the Software as a Service Business Model: From CPU Time-Sharing to Online Innovation Sharing, pp. 177–186 (2005)

29. Anonymous SoftwareMarketingAdvisor.comTips for SaaS Marketing, http://www.software-marketing-advisor.com/saas-marketing.html
30. Weobong, D.: How to Sell SaaS, http://howtosellsaas.com/
31. Burrell, C.: SaaS New Engagement Approach in Europe by Fujitsu Services. Fujitsu Sci. Tech. J. 45, 275–282 (2009)
32. Moore, G.A.: Living on the fault line, revised edition: Managing for shareholder value in any economy. HarperBusiness, New York (2002)
33. Anonymous THINKstrategiesCIO's Guide to Software-as-a-Service, http://thinkstrategies.icentera.com/portals/default.asp
34. Gardner, T.: Financial Implications of the SaaS Business Model, http://www.sterlinghoffman.com/newsletter/articles/article339.html
35. Hirsijärvi, S., Remes, P., Sajavaara, P.: Tutki ja kirjoita, 13th edn., Tammi (2007)
36. Hirsjärvi, S., Hurme, H.: Tutkimushaastattelu: Teemahaastattelun teoria ja käytäntö. Yliopistopaino (2001)
37. Anonymous Cloud Software Program, http://www.cloudsoftwareprogram.org/
38. Deeter, B., Cowan, D., Goodman, B., et al.: Bessemer Venture PartnersBessemer's Top 10 Laws for Being "SaaS-y", http://www.bvp.com/downloads/saas/BVPs_10_Laws_of_Cloud_SaaS_Winter_2010_Release.pdf

Business Continuity Solutions for SaaS Customers

Tommy van de Zande and Slinger Jansen

Dept. of Information and Computing Sciences,
Utrecht University
{T.JacobusmeergenaamdvandeZande,R.L.Jansen}@uu.nl

Abstract. Organizations are increasingly adopting SaaS-solutions in favor of traditional on-premise solutions, because of the advantages in terms of cost reduction, implementation time and scalability. Business continuity of these SaaS-solutions is often neglected, even when business processes that depend on the SaaS-solution are critical. This paper addresses business continuity for SaaS-solutions by identifying and evaluating different business continuity solutions to protect customers from the risk of their SaaS-provider going bankrupt. Two solutions; 'SaaS-escrow' and the 'SaaS-guarantee-fund', are evaluated in expert interviews and a survey. The conclusions of this research are that there is a need for SaaS business continuity solutions, SaaS continuity solutions are not frequently employed, and that the two solutions presented here are favored equally by a panel of business managers.

Keywords: Business Continuity, SaaS, Escrow.

1 Introduction

Software as a service (SaaS) is a form of software deployment, that delivers software on a subscription basis through the internet. With SaaS, companies no longer have to buy or develop a complete software solution up front, instead they *rent* it. Pricing models can vary, but generally customers pay on a subscription-basis or on a usage volume-basis [1]. SaaS is rapidly growing in popularity. Recent developments in Internet technology and broadband adoption enabled online software to serve as a true desktop software replacement. Also the recent economic and financial crises have played their part in showing the advantages of SaaS, because of its low upfront investment and high scalability. Because of these advantages in comparison to traditional software licensing models, a large number of small businesses and start-ups are already using SaaS solutions for some time now, and even large institutions and corporations are migrating their data from on-premises software to externally developed and hosted software.

Service Level Agreements are legal documents where customer and provider agree on the quality of service, by quantifying minimum quality of service [2]. For SaaS, these mostly discuss availability and response time. There are some problems that are generally neglected, like what happens when a SaaS-provider goes out of business. If a company is using mission-critical software that is hosted off-premises, it could get into big problems when the provider decides to pull the plug. The business continuity risks include (in order of importance):

R1: Lose access to data
R2: Lose access to the application itself
R3: Lose support and maintenance

These problems are considered deal breakers by decision makers and counter measures need to be taken to assure continuity, before SaaS can truly serve as a replacement for offline software.

For shrink-wrapped software, business continuity can be arranged using a source-code escrow [3]. With source-code escrow, the developer stores its source code with a trusted third party, the escrow agent. If this developer goes bankrupt or otherwise defaults on his support and maintenance obligations, the customer requests the escrow agent to release the source-code. If the escrow-agent confirms that a valid *release condition* has been met, he will transfer the source-code to the customer.

In the late 90s of the previous century, application service providers started hosting applications off-site, offering some of the same advantages that current SaaS-providers offer. In the end those traditional ASPs were not able to deliver reliability and quality standards demanded by business customers [4]. The business models of most traditional Application Service Providers (ASPs) were fundamentally different from current SaaS business models; traditional ASPs repackaged existing legacy software and offered it off-site, so customers still had to buy a license to use the software [5,1]. After the burst of the *dot-com bubble* in the year 2000, many of those traditional ASPs faced financial problems, and some even bankruptcy, leaving their clients with no access to their software [6,7]. The new generation of ASPs, who offer true SaaS-solutions, differ in a way that they do not resell usage rights for existing enterprise applications. They develop their own web-based applications on a new multi-tenancy design paradigm, which makes serving multiple clients more scalable and cost-effective [5].

If SaaS is going to serve as a more cost effective and flexible replacement for shrink-wrapped software, it is imperative that there is a clear way for both customers and providers to arrange a viable continuity solution. Several scholars identify the need for research attention to the business aspects of SaaS, and mention the risk of SaaS-providers going bankrupt as a serious issue [8,9].

The goal of this paper is to give a *state-of-the-art* report on available continuity solutions for SaaS. We discuss the necessity of such continuity solutions and compare different initiatives. Since there is no clear solution to the aforementioned problem, the research is explorative. First the requirements needed for a successful SaaS business continuity solution are identified. The identified requirements are based on the results of five semi-structured interviews: four interviews with CEOs from several Dutch SaaS vendors and one interview with the CEO of a Dutch Escrow agent. During these interviews, we distinguished two different SaaS continuity solution. These solutions are described and compared with each other. We also compare these existing solutions with the requirements that have been identified. Then we discuss the necessity of SaaS continuity arrangements. And finally, a small survey is carried out among SaaS providers and customers.

The paper is built up as follows; Section 2 describes the interviews and presents some initial findings. Section 3 discusses what risks should be covered to provide SaaS continuity, and presents the requirements for a SaaS continuity solution. In section 4 we discuss what solutions are currently being offered. Section 5 discusses the necessity of business continuity solutions for SaaS. Section 6 presents the results of the survey. In section 7 we conclude the paper.

2 Interviews

Because SaaS Business Continuity is a new and unknown topic, we started our research by conducting five semi-structured interviews with several people in the SaaS and continuity business. These interviews provided data for both the requirements for a continuity solution as well as details of the two existing continuity solutions.

2.1 Interview Design

The five interviews were conducted with four CEOs of several (small) Dutch SaaS-providers and the CEO of one Dutch escrow-agent. The interview participants were selected pragmatically. Two were approached within the network of the authors. Two others were found at a conference on SaaS and cloud computing. They were all approached because their SaaS-services targeted other businesses. The interviewed escrow-agent was referred by one of the SaaS-providers. The interviewee selection process threatens the validity of the research, in that the participants were selected based on availability and willingness to participate. However, due to the method used to find participants we dare say with some certainty that the interviewees' responses were relevant, on-topic, and addressed the topic with sufficient experience (avg. 5 years) and insight. The SaaS-provider interviews discussed themes such as customer demand for business continuity solutions, use of business continuity solutions, what business continuity solution they would prefer and what would happen to their customers if the provider suddenly disappeared, what problems they would face and what data they would lose. We also asked the interviewee's if there were any other solutions to provide business continuity. The interview with the escrow-agent was conducted to gather detailed information about how their SaaS-escrow solution works. All the interviews took approximately one hour and data was recorded by taking notes. The notes were later transcribed and sent to the interviewees for verification. Three of the SaaS-provider interviews and the escrow-agent interview were conducted on locations at the companies. One SaaS-provider interview was conducted by phone. All interviews took place during March-June 2010. Table 1 shows some characteristics of the different companies where we conducted the interviews.

Table 1. Overview of the interviewed companies along with the type of product they offer and the approx. number of customers

Company	Product	Customers	Business Continuity Solution
Provider 1	SaaS CRM	100	None
Provider 2	SaaS Planning suite	39	Escrow for specifc customer
Provider 3	Several solutions	60	None
Provider 4	SaaS ERP software	80	Escrow (optional)
Escrow agent	Saas-Escrow	20	N.A.

2.2 Initial Results

Two of the interviewed SaaS-providers (Provider 1 and 3) explained that they have thought about business continuity solutions, but in the end did not go through with it, because almost none of their customers demanded such a solution. They stated that a lot of customers do not worry about the financial stability of their SaaS-provider, but that it was also possible that they didn't understand the risks. Provider 2 did use an escrow-solution, but only with one of their customers, because they were the only one demanding such a solution. Provider 4 offered escrow as an extra service, but none of their current customers actually applied for it.

According to the SaaS-providers, there is not much demand for SaaS business continuity from the customers. Some of the SaaS-providers believed that customers do not fear the continuity risks because of the following fact; a SaaS-provider, like any other subscription-based service, has clear vision and control over its finances. A SaaS-provider can predict with a high amount of certainty what his revenue will be over a certain period of time. When the SaaS-provider uses one-year contracts, then he knows how much revenue he will make for the next 12 months at any given moment. So the only way that he could get into trouble is if his costs will go up unexpectedly, which is rare because most of his costs are also on a subscription-basis. Added to those clear costs is the trust that, when bankruptcy does occur, a Bankruptcy Trustee will keep the software running for as long as possible because it is a revenue producing part of the company. Section 5 elaborates on this assumption.

The clear and stable financial situation of SaaS-providers might explain the absence of business continuity worries among SaaS-customers. However, demand for business continuity solutions does exist. The escrow-agent stated that he currently sees an increase of demand for their SaaS coninuity solution. The explanation of SaaS-providers that their customers do not have to fear bankruptcy might be true for most customers, but some simply can not bear the risk.

3 Guaranteeing Business Continuity for SaaS

In this section, we explain the business continuity risks and requirements for a successful business continuity solution. These risks and requirements are based on the interview data. The risks and list of requirements were created during the first interview. These risks and requirements did not change throughout the further interviews, although they were reformulated and reshaped during the second and third interview. The fourth interview corroborated the results from the previous interviews.

The SaaS-model is fundamentally different from the traditional software-licensing model. The difference is that the customer does not possess the object-code on-premises but, instead, accesses the application on a remote server, using the internet. This remote server is managed by the SaaS-provider, either on-site or by using a hosting-provider. The actual hardware where the software and data reside is out of the customer's reach and control. Some SaaS-solutions even include content from third-party content providers in their SaaS-software. A customer does not have anything to do with all those external parties, and commonly they do not even know that external parties are being employed. The customer pays its SaaS-provider for access to the software, the SaaS-provider in his turn pays the different parties involved to deliver its service. Together with the interviewed experts, we identified several requirements for a business continuity solution.

The main goal with a business continuity arrangement is the assurance that the customer continues to have access to his SaaS application and data, even if the SaaS-provider disappears. To make the continuation of access and data work several key elements should be covered. The most important element is the customer's data. Even if access to the application has been suspended, with a recent backup a customer at least does not have to worry about losing his data, and he can start migrating towards an alternative solution, and only lose access to his application for a couple of days or weeks, depending on the size of the data and type of application. Losing access to the application would still be a major problem for any organization, but without access to (a recent backup of) the data, problems would be much worse; imagine a company losing all its data that resided in their CRM system. The company would not be able to service their current customers or process new leads. Losing access to CRM data could be disastrous for a lot of companies. So the first step towards business continuity would be the ability to acquire regular backups of the data.

The next step towards a more complete business continuity agreement would be an agreement with the hosting-provider, in such a way that they ensure they will continue hosting the application even when the SaaS-provider gets into financial difficulties. Such an agreement could be arranged by a SaaS-customer itself, but that would not work if there are more customers hosted on the same server, which is generally the case with SaaS. Therefore, a logical step would be to arrange this hosting continuity agreement with a separate legal entity. This legal entity can either be a commercial escrow-agent, or a foundation/fund founded by the customer(s) or SaaS-provider themselves. Such a separate entity can also provide some additional services next to simply continuing hosting (and providing the funds to do so). They could offer support for the application when the SaaS-provider fails to do so. This hosting continuity is a kind of insurance, and will be cheaper if arranged with multiple SaaS-providers at the same time, because the chance that all of the SaaS-providers fail at the same time is lower than the chance that one of them fails. Some SaaS-providers also use third party content or services in their applications. Sometimes this content is free, but frequently the SaaS-provider pays the content provider for the content. For better continuity these third party providers also have to be included in the business continuity arrangement. The last step for complete business continuity is continuing support and maintenance for the application, to help customers with possible problems and keep the application running.

To summarize, the requirements for a complete SaaS business continuity solution are (in order of importance):

1. **Own Backup:** Every SaaS customers should be able to download all of its data.
2. **Hosting Insurance:** A third party should create an arrangement with the hosting provider to continue hosting even if the SaaS provider fails.
3. **Arrangement with content providers:** If the SaaS application contains (paid) content from third parties, they should also continue providing the content.
4. **Support and maintenance for the application:** If the SaaS provider disappears, the customer also loses support. A third party could try to continue support for the application.

A solution that meets these requirements, should be able to effectively protect customers of a SaaS-provider when it goes bankrupt or otherwise out of business. Two solutions that were identified during the interviews are presented in the next section.

4 Available Solutions

Even though SaaS business continuity guarantees are not common, several companies, like the escrow-agent we interviewed, are already offering solutions. In this section these solutions are discussed, and compared with the requirements we identified for a successful business continuity guarantee. Data about the SaaS-escrow solution came from the interviewed escrow-agent, completed with data from websites of several escrow-agents. Data concerning the SaaS Guarantee fund came from the interviewed CEOs of Provider 1 and 4. The SaaS-providers also pointed out the downsides for both solutions.

4.1 SaaS-escrow

A common solution for SaaS business continuity, SaaS-escrow, is offered by existing escrow-agents, who already offered source-code escrow for traditional software. As the interviewed escrow-agent pointed out, most escrow-agents have added a ‘SaaS-escrow’ service to their product portfolio. SaaS-escrow usually is a modified version of the regular source-code escrow of the escrow-agent. The modification generally consists of the addition of a data back-up with the deposit of source-code. More complete escrow solutions also provide 'continuation of hosting', where they arrange an agreement with the hosting provider, that whenever the SaaS-provider gets into problems, the escrow-agent takes over the financial obligation towards the hosting-provider. The hosting provider in return promises that they will continue hosting the SaaS-application and data under any circumstance. Escrow-agents differentiate their solution by offering different extra services for SaaS-escrow, like delivering support and maintenance of the escrowed application when the escrow is released. The escrow-agent takes over support and maintenance for a predetermined amount of time. During this time, customers have the ability to migrate their data to another more permanent solution. SaaS-escrow solutions can be arranged on two different levels. The first one is a three-party arrangement with the SaaS-provider, the SaaS-customer and the escrow agent. In this arrangement the individual customer is the only customer who is able to access the application when the escrow is released. But when multiple customers demand an escrow-arrangement, the second arrangement makes more sense; a two party 'master-contract' arrangement between the SaaS-provider and the escrow-agent. In this arrangement there is no limit on how many customers become a beneficiary of the escrow-arrangement, to benefit from the arrangement only depends on each individual customer if they want to sign up for it (and pay the price of course). Such a master-contract arrangement is initially more expensive than a single three-party arrangement, but as an advantage it is much easier to add new customers to the arrangement, and spread the costs over all the participating customers.

With SaaS-escrow, as opposed to traditional escrow services, the initial purpose of storing the source-code and releasing it to the customer on certain release-events is less important than the continuation of application access. Most SaaS-customers would not have any use for the source-code, because they probably do not have the hardware and infrastructure to deploy the software application on-premises. As an added value, the escrow company can offer support and maintenance for the SaaS application, by storing documentation and remaining in contact with key-persons involved with the software-maintenance at the SaaS-provider. So with SaaS-escrow, the escrow-agency acts more like an insurance company for hosting costs than a storage facility for sensitive information. This also creates a possible risk for the business continuity of the escrow-company itself. If the SaaS-provider grows in size, the cost for hosting the application grows accordingly. That way, it could become too expensive for the escrow-company to take over hosting-costs if a big SaaS-provider goes bankrupt.

Another possible problem with SaaS-escrow arrangements is that the solution is general and standardized, so for some specific SaaS-solutions the escrow-solution simply would not work or only cover a part of the business continuity problems. For example, typical escrow solutions do not offer support for SaaS-applications, which use a lot of third-party content in their application, because they only cover continuation of payment towards the hosting provider, but not the payment towards third-party content providers. Another example is a SaaS-provider who uses a lot of different hosting-providers to host their application, for example to provide better reliability and speed for customers around the world. The escrow-company then should sign a contract with every single one of those hosting-providers to be able to continue hosting the application for every customer.

4.2 SaaS Guarantee Fund

The CEOs of Provider 1 and Provider 4 both explained another possible solution to guarantee business continuity: a SaaS Guarantee fund. The SaaS Guarantee Fund is based on the idea of the so-called *Travel Guarantee Fund*, which exists in several countries. Such a Travel Guarantee Fund covers the risk for travelers who book a trip with a travel-agency or tour-operator which goes bankrupt before or during the actual trip. The Fund provides customers of a participating travel-agency with (financial) protection against such risks, so that customers are guaranteed that their trip is paid-for even though the travel-agency defaults. An adaption of such a fund could function as a business-continuity guarantee solution for SaaS-providers. SaaS-providers have a clear image of their financial situation over the coming months. They know what their costs will be to pay every third-party involved in running the SaaS application for upcoming months. With this financial forecast in mind, they could set up a fund with a budget large enough to cover those costs for several months. The fund then arranges an agreement with all those third-parties, to continue their services towards the SaaS-provider under any circumstance. Since the fund is a different legal entity than the provider itself, it is not affected by financial problems or bankruptcy of the provider. In case of bankruptcy or severe financial problems, the fund can take over the payment towards all third-parties for a few months, during which customers have time to migrate their data towards another solution, or during which the SaaS-provider can make a new start again *"Storing the code and data at a third party could be dangerous regarding theft of IP and setting up a guarantee-fund is not that hard to achieve and has the advantage of (expected) lower costs."* is what one of the survey respondents answered when asked which arrangement he thinks works best.

The guarantee fund could initially be small and only support one single SaaS-provider, so it can be perfectly tailored to support that single solution (and its customers of course) on all (business continuity) aspects. To lower costs and efforts, multiple SaaS-providers could set-up a fund together, lowering the required cash-deposit per provider because it is unlikely that the fund has to cover for all the participating providers at the same time. But a fund for multiple providers also has its disadvantages, for example: the fund will be less customizable towards every specific SaaS-solution. Another disadvantage is that when one of the participating SaaS-providers fail, then the others feel that they pay for its failure.

Currently, there are no known SaaS-guarantee funds with multiple participators. The problem probably lies in the initial start-up of such a fund. Who takes the initiative and invests the initial time and money in it? SaaS-providers are commercial companies, and they justify their investments and projects with a business case which predicts a profitable outcome. A SaaS-guarantee-fund for multiple providers does not have any apparent extra benefits for the SaaS-provider who initiates the fund. Some companies have started a SaaS-guarantee-fund for their own solution though. An apparent party to set up and manage a multiple SaaS fund would be a (software) trade association. They can provide the fund along with possible certification for participating SaaS-providers.

4.3 Comparison

An advantage of the SaaS-escrow solution is that it is easy to set up, because the solution is an existing package for which providers only have to sign up once. Most escrow-companies already exist for several years, and have the required legal and technical knowledge to provide a reliable business-continuity solution. Another advantage are its transparent costs. Because escrow solutions are relatively standard, the costs are known in advance. A downside is that SaaS-escrow is a standard solution with less customizability than a custom SaaS-guarantee-fund, a SaaS-provider who uses a lot of different hosting parties and content providers would have a hard time finding a suitable escrow-solution. Also, escrow-solutions are more expensive than SaaS-guarantee-funds because of overhead costs and the for-profit nature of escrow-companies. Survey respondents who preferred the escrow-arrangement used arguments as: *"The SaaS Escrow guarantees that the source code and data are stored and will be given to the customer when things go wrong, whereas a Guarantee Fund will only give help to customers (missing the actual 'guarantee' as given by the other arrangement)"*, *"They [The escrow company] have the legal expertise available"* and *"Because it is easier to set up and can be arranged on forehand"*.

An advantage of a SaaS-fund is that it can be set-up for a single SaaS-solution, so the provider remains completely in control of who has access to its source-code and other intellectual property, while still providing a workable business-continuity guarantee. Also, because of its non-profit nature and low overhead costs, all of its income will be used to serve its core activity; provide continuity, instead of overhead costs like marketing and management. Arguments favoring the SaaS-fund were: *"Without the capital knowledge of the technical staff, the code/data is of very little use."*, *"Storing the code and data at a third party could be dangerous regarding theft of IP and setting up a guarantee-fund is not that hard to achieve and has the advantage of (expected) lower costs."* and *"It is the only option that can cover the complete infrastructure, including all third parties and resellers."* Table 2 summarizes the possible advantages and disadvantages.

When we compare both solutions to the previously mentioned requirements on how a complete business continuity solution should work, we conclude that both solutions can cover the basics: they both offer data backups and continuation of hosting. SaaS-escrow does not offer extra continuity agreements for other third-parties like content providers or extra hosting-partners. The SaaS-fund could contain all of those options, but in the end the features of a SaaS-fund depend on how the fund is implemented.

Table 2. Overview the advantages and disadvantages of the different solutions

	SaaS-escrow	**Guarantee Fund**
Advantages	Easy to arrange Legal knowledge Clear costs Experience	Complete control Customizable for specific solution (Expected) lower costs
Disadvantages	Expensive External party	Requires more effort from provider Responsibility stays with the provider No prior experience

5 Necessity of SaaS Continuity Arrangements

There are several ways to ensure business continuity with SaaS. The only question that remains is: is it necessary? This question has to be answered by each (potential) SaaS-customer individually. Several authors doubt the necessity of the traditional source-code escrow [10,11,12] for shrink-wrapped software, because few escrows are actually released. So it is only logical to doubt the necessity of SaaS business continuity solutions as well. Of course, the model of SaaS versus that of shrink-wrapped software is completely different, and so are the risks at stake. With SaaS, if things go wrong, they could have disastrous consequences for some customers, because they could lose access to their software as well as their data. But the chances of a SaaS-provider going bankrupt and leaving its customers without any access to their application or data from one day to another is in fact small, when for example considering the interests of a bankruptcy trustee. *"A great deal depends on the application itself – how critical is the data in the application? What are the workarounds I can use to back up my information without incurring a lot more work?"* one respondent answered on how he thinks about business continuity arrangements.

5.1 The Bankruptcy Trustee

The interview results show that many SaaS-customers have not given much thought to business continuity risks, and frequently rely on a SaaS solutions without proper business continuity guarantees. Others, as pointed out by the CEO of Provider 4, simply did not worry that they lose access to their application because they believe in a simple yet effective assumption, that is based on the SaaS business model itself; the SaaS model consists of a constant revenue stream. When a SaaS-provider files for bankruptcy, a Bankruptcy Trustee will always keep that constant stream of revenue flowing because it can be used to pay off creditors. To keep the revenue flowing, he will have to keep the SaaS application running. The Bankruptcy Trustee would cut off departments like marketing and R&D, but will keep the core services running.

The danger is that this trust in the bankruptcy trustee is based on an assumption, albeit a logical one. A Bankruptcy Trustee is not obliged to keep the service running, and could decide to liquidate all assets instead, leaving its customers without their software. Also, a SaaS-provider can stop its services even though it did not go bankrupt. For example, the Californian-based Platform-as-a-Service provider Coghead, which provided an online hosted platform to create enterprise database applications, announced in February of 2009 that their intellectual property assets were acquired by SAP, and that they would stop supporting the platform within the next month [13]. Customers had one month to develop a new application on another hosted platform and migrate their data towards it. Some companies see these risks as an ultimate downside for outsourcing their IT to an external service provider [14], and use it as an argument to stick to the traditional on-premises software.

6 Survey

To gain some insights in customer preference, and in an attempt to verify the identified requirements, we carried out a small survey. The survey focussed on gaining insight in the number of companies who utilize business continuity solutions and tries to identify a preference for one of the two different continuity solutions.

6.1 Survey Design

In an online survey, we asked several IT decision-makers and SaaS providers about their thoughts on business continuity with SaaS. We asked them if their SaaS-provider provided a solution for business continuity, how important they think such a continuity solution is, and which of the previously presented continuity alternatives they think is best. The survey results are based on 20 respondents. We selected participants by posting a message in several SaaS-user and provider groups on LinkedIn. We also asked the members of the Dutch CIO platform to participate. The survey first asked respondents wether they were a customer or provider of a SaaS provider. Then the respondents were asked if their SaaS-solution already offered a business continuity solution. Respondents who indicated that they are a customers of a SaaS-solution were asked wether they would consider a SaaS-solution if it did not provide a business continuity solution. Then the survey asked respondents to rate several aspects of SaaS-products/providers on a scale of importance ranging from *"not important"* (depicted as 0 in the graph) to *"extremely important"* (depicted as 3.0 in the graph). The different aspects were: size of the company, location of the company, price of the SaaS-solution, financial situation of the SaaS provider, technical continuity solutions, business continuity solutions and data export abilities. Finally, we presented a short description of the two business continuity solutions discussed in this paper; the SaaS escrow solution and the SaaS guarantee fund, and we asked respondents to indicate which one they preferred and explain why.

6.2 Survey Results

From the 20 respondents, 50% claimed to provide a SaaS-solution, the other 50% claimed to be (professional) SaaS-users. Only three of the in total 20 respondents (or their provider) currently offered a complete business continuity solution like SaaS-escrow or a guarantee fund. The other 17 respondents either simply answered *"no"* or stated that they were still looking into it. Five respondents stated that their provider offered data-export abilities as a continuity solution. These results are quite surprising when we look at the results of the second survey question, which we asked to the group that stated they were (potential) SaaS-users: *"Would you consider a SaaS-solution if it does not provide a clear solution for business continuity?"* Seven out of ten respondents answered that they *"could not take such a risk"*, while the remaining three respondents answered that it depended on the specific company and SaaS-solution.

The combined results of the ranking of importance of several SaaS aspects are visualized in Fig. 1. As expected, the location of the company is perceived less important than aspects like price, financial situation and technical continuity, since the access to SaaS-applications is location independent. Respondents rated data export abilities as the most important aspect when selecting a SaaS-solution. This is in line with our findings in the previous chapter, but it is surprising to see that even though it is perceived as extremely important, not every SaaS-solution offers this possibility. The next most important aspect is technical continuity. That is not surprising, because the likelihood of technical problems is much higher than that of business continuity problems. The respondents rated business continuity solutions as the third most important aspect of a SaaS-solution, with almost the same score as price and financial stability. This is quite surprising because business continuity solutions are not common in SaaS-solutions, based on the survey results which showed that only 3 out of 20 SaaS-solutions offer some kind of business continuity solution. So there is a big difference between how important people think business continuity solutions are and the number of SaaS-providers actually using a business continuity solution. There was no significant difference between the ratings of SaaS-providers and customers.

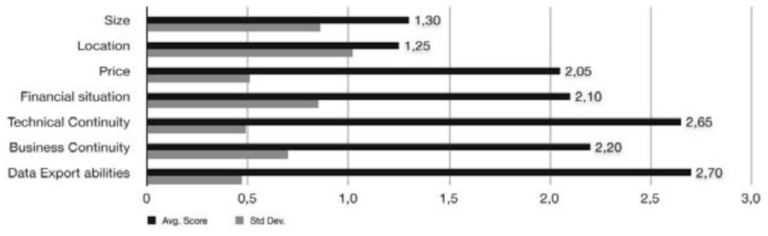


Fig. 1. A graph showing the average score of several aspects related to SaaS-solutions on a scale of importance (from 0 to 3.0), along with the standard deviation

Exactly 50% of the respondents preferred the escrow-solution, while the other 50% preferred the guarantee-fund alternative. Some arguments in favor of the escrow-solution were: *"Because it's easier to set-up and the arrangement is active immediately"* and *"they have the legal expertise available"*. Arguments in favor of a SaaS guarantee fund were: *"Without the capital knowledge of the technical staff, the code/data is of very little use"* and *"Storing the code and data at a third party could be dangerous regarding theft of IP and setting up a guarantee-fund isn't that hard to achieve and has the advantage of (expected) lower costs"*. There was nog significant difference between the solution-preference among SaaS-providers and SaaS-users. Several respondents explained that they prefer an escrow arrangement for standard SaaS-solutions, but that they would prefer a custom guarantee fund for more complicated solutions with a lot of third parties, because a guarantee fund has the ability to *"cover the whole chain"*.

To summarize, the survey showed that people think business continuity solutions are important for SaaS-providers, but currently not many SaaS-providers actually use a business continuity solution. The survey showed that both of the discussed solutions are seen as viable options for business continuity guarantees, with equal votes, but each with different advantages and disadvantages.

7 Conclusions

A company going bankrupt will always be a risk for a customer, no matter what kind of business it is in. A business continuity solution could work to make the consequences for a customer less disastrous, but it is hard to provide complete protection in every scenario. In most cases, customers of SaaS-providers can find comfort in the fact that because of the SaaS business model, keeping the service running has the highest priority even if the company goes bankrupt, because it provides a continuous revenue stream. In any case, the customer at least has to make sure that he has access to his data and be able to acquire a backup. This is the most important step towards business continuity. A customer should ask himself how problems with the SaaS provider would affect his own business. If a customer would get into serious problems with its own business continuity if the SaaS provider fails, then a business continuity arrangement makes sense.

As our survey showed, most providers and customers think business continuity arrangements are important, but not many providers are currently offering a business continuity solution. This probably has to do with the fact that SaaS is a relatively new phenomenon and that there is no 'best practice' yet. With this paper we hope to give some clarification on this topic, and help clarify the different options available to arrange business continuity. Several issues that could influence the validity of the survey results is that the number of respondents is quite small and it did not distinguish between different SaaS-solution. It is possible that several respondents were using the same solution. However, the goal of this paper is not to identify how many companies currently use business continuity solutions, or point out which solution is preferred. Instead, this paper calls for attention on the topic and provides insights in the risks at stake and the two possible solutions.

The types of business continuity solutions we discussed both have their pros and cons, and there is no one *best* solution. Because the SaaS model, in its current form, is relatively new, there are no practical examples to assess the effectiveness of both solutions in real life. There are no known cases where one of the two continuity arrangements were ever actually put into effect yet.

Theoretically both the escrow-solution and the fund-solution should function as a reliable solution. The big difference between the two is that the escrow-solution is a commercial solution, which could be more expensive because the escrow-company needs to make a profit, but offers a complete and ready to use solution with professional (legal) support. The fund-solution can be cheaper to set-up, but is more time consuming and requires a lot of effort from the SaaS-provider itself. What is the best solution depends on the type of SaaS solution and personal preferences of both the provider and its customers. We think that when business continuity solutions are needed, for most standard SaaS solutions the escrow version is preferred because of its simplicity and low effort requirements. When the SaaS solution is more exotic or needs more specific arrangements with many third parties or for a difficult infrastructure, the fund-solution appears to be a better alternative.

Acknowledgements. The authors would like to thank the interviewed CEOs and survey respondents, for their valuable insights. We would also like to thank Pasi Tyrväinen, as well as the anonymous reviewers, for their valuable comments and suggested improvements.

References

1. Abdat, N., Spruit, M., Bos, M.: Software as a service and the pricing strategy for vendors. In: Strader, T. (ed.) Digital Product Management, Technology and Practice: Interdisciplinary Perspectives. Advances in E-Business Research (AEBR) Book Series, pp. 154–192. IGI Global (2010)
2. Hiles, A.: Service Level Agreements: Panacea or Pain? The TQM Magazine 6(2), 14–16 (1994)
3. Freeman, E.: Source Code Escrow. Information Systems Security 13(1), 8–11 (2004)
4. Dubey, A., Wagle, D.: Delivering software as a service. The McKinsey Quarterly (2007)
5. Kaplan, J.: SaaS: friend or foe? Business Communications Review 37(6) (2007)
6. Currie, W., Seltsikas, P.: Exploring the supply-side of IT outsourcing: evaluating the emerging role of application service providers. European Journal of Information Systems 10(3), 123–134 (2001)
7. Chen, M., Chen, A., Shao, B.: The implications and impacts of web services to electronic commerce research and practices. J. Electron. Commerce Res. 4(4), 128–139 (2003)
8. Marston, S., Li, Z., Bandyopadhyay, S., Zhang, J., Ghalsasi, A.: Cloud computing - The business perspective. Decision Support Systems 51(1), 176–189 (2011)
9. Armbrust, M., Stoica, I., Zaharia, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A.: A view of cloud computing. Communications of the ACM 53(4), 50 (2010)

10. Mezrich, J.: Source Code Escrow: An Exercise in Futility. Marquette Intellectual Property Law Review 5 (2001)
11. Denson, W.: Source Code Escrow: A Worthwhile or Worthless Investment. Rutgers Bankruptcy Law Journal 1 (2002)
12. Helms, S., Cheng, A.: Source code escrow: Are you just following the herd? (2008), http://www.cio.com/article/187450/Source_Code_Escrow_Are_You_Just_Following_the_Herd_
13. Savvas, A.: Coghead customers left high and dry despite sap acquisition (2009), http://www.computerweekly.com/Articles/2009/02/20/234935/Coghead-customers-left-high-and-dry-despite-SAP-acquisition.htm
14. Spiotto, A., Spiotto, J.: Ultimate Downside of Outsourcing: Bankruptcy of the Service Provider. The American Bankruptcy Institute Law Review 11 (2003)

Software Ecosystems: A Set of Management Practices for Platform Integrators in the Telecom Industry

Martti Viljainen and Marjo Kauppinen

Aalto University School of Science and Technology
Department of Computer Science and Engineering
Espoo, Finland
mtviljai@cc.hut.fi, marjo.kauppinen@tkk.fi

Abstract. There is an emerging trend for software companies to adopt ecosystem strategies. A software ecosystem consists of an open technology platform with complementary components produced by several software companies or communities. By open innovation, ecosystems add value to the platform integrator's core offerings, while causing management challenges. This paper investigates what management practices support platform integrators operating in software ecosystems. The set of management practices that is synthesised from the literature includes technology scouting, orchestration, software supply network management, and technology asset management. This paper gives a structured overview of the management practices and links them to the technology and innovation management processes. A case example shows how a platform integrator utilises these practices in the telecom industry.

Keywords: software ecosystem; technology management; industry platform; telecommunications; open innovation.

1 Introduction

The product line approach, which originally emerged around the car industry, has been adopted in the software industry too. It has made possible the efficient mass production of standardised products, while, however, providing limited diversification [1]. One major approach to software product lines is the use of a common platform as a basis for product derivation. In the 1990s, the popular discussion was concentrated on the benefits of software reuse in intra-organisational "product platforms" [2]. Later in the 1990s, the strategic potential of "industry-wide platforms" became a topic demonstrated by the dominant market positions among the platform leadership (Microsoft Windows, Intel, Cisco) [3, 4].

Compared to the product platform, the industry platform has four significant characteristics. First, the industry platform consists of components that are likely to come from different companies called "complementors" [2]. The complementors, together with the industry platform, create an ecosystem in which the participants are dependent on each other to some extent. Second, the industry platform has relatively little value to users without complementary products or services [2]. Third, the industry platform tends to create "network effects", which are positive feedback loops that can grow at increasing rates as the adoption of the platform and the complements rises [2]. The network effect can be "direct", such as technical compatibility (Microsoft-Intel). Alternatively, an "indirect" network effect is caused when a number of application developers, content producers, buyers and sellers, or advertisers adopt a particular platform (Google, Facebook). The more external adopters create or use complementary innovations in the ecosystem, the more valuable the platform (and the complements) becomes [2]. This encourages more users to adopt the platform and more complementors to enter the ecosystem. In market-driven software development, many possibly anonymous stakeholders influence requirements value chains in an ecosystem [5]. Fourth, markets for industry platforms tend to be "multi-sited" [6]. Companies compete not only for customers but also for developers and other complementors.

Jansen et al. [7] define a software ecosystem (SECO) as a set of actors functioning as a unit and interacting with a shared market for software and services. A SECO consists of actors such as independent software vendors (ISV), outsourcers, and customers [7]. A SECO typically is interconnected with institutions such standardisation organisations, open source software communities, research communities, and the related ecosystems. There are two main types of roles actors can take in a healthy SECO [8]. The keystone players (or shapers) are the drivers of platform technologies and standards [9]. The niche players (participants/followers) require the standard or platform technology provided by the keystone player for creating business value [9]. In this paper, we focus on the role of a keystone player i.e. a software vendor that acts as a platform integrator.

There are several reasons why a company may decide to open up its platform and transition from a software product line to a software ecosystem. A company may realise that the amount of functionality that needs to be developed to satisfy customer needs is far more than what represents a reasonable investment for the company, which calls for external development. The ecosystem approach may increase the value of core offerings to existing users, enhance the attractiveness of the ecosystem for new users, increase the "stickiness" of the platform (lock-in makes it harder to change platforms), accelerate innovation through open innovation in the ecosystem, and reduce innovation and maintenance costs [10].

Besides the benefits of the ecosystem approach, companies face challenges in, for instance, gaining an insight into ecosystems, identifying survival strategies inside ecosystems, and disclosing IPR (Intellectual Property Rights) selectively [7]. The competition is often about who has the best platform strategy and the best ecosystem to back it up [2]. If the challenges are made more understandable and manageable, the realisation of the benefits becomes more likely. In the literature there is still a fragmented view of supporting management practices. This paper aims to create an overview of available technology management practices. The telecommunication industry is selected as an example since it demonstrates strong network effects leading to clear interdependence between platforms and complements [4]. The focus is on software ecosystems, since telecom platforms are becoming more and more software-intensive. The research question is formulated as follows: What management practices support platform integrators operating in software ecosystems?

The management practices are synthesised from the recent literature related to the software ecosystems, technology management, and industry platforms. As software ecosystems are a relatively new research topic, the most of available publications could be found and acknowledged. Especially, publications of Jansen et al. [7, 9] are used as a basis for terminologies and subsequent searches. A case example from the telecom industry is provided on the basis of experiences of the first author working in Nokia Siemens Networks.

This paper is structured as follows. Section 2 synthesises the management practices that were found in the literature. Section 3 presents a case example in the telecommunication industry where the management practices are utilised. Finally, the paper summarises the key results and suggests directions for further research

2 Management Practices

The management of a software ecosystem can be viewed as a part of the general technology and innovation management processes. Rohrbeck presents these processes in [11]. Jansen et al. investigated challenges related to the software ecosystems at three scope levels - the ecosystem, software supply network, and organisational levels [9]. These challenges include how to gain insights into ecosystems, how to identify survival strategies at the ecosystem and supply network levels, and how to open up to a software ecosystem without losing critical intellectual property [9].

Our paper synthesises a set of four management practices that are linked to the technology management process introduced by Rohrbeck [11] and aim to address the challenges identified by Jansen et al [9]. Following Rohrbeck [11], the first practice in the technology management process is selected as technology scouting. Furthermore, the three subsequent management practices address the development and storage of the technology that is exploited in products or sold. We divided these management practices according to the activities needed at the three scope levels. The first one is orchestration as defined by Jansen et al [9]. We labeled the last two management practices as software supply network management and technology asset management. The management practices are summarised as follows.

- *Technology scouting* is a systematic approach to gather information in the field of science and technology and to facilitate technology sourcing [11]. It is a way to gain insights into a SECO when deciding on which level of intensity to participate in it and finding out about new technologies and relationships. This involves estimating opportunities and threats on the basis of identified SECO characteristics, such as health.
- *Orchestration* is a method to leverage the advantages of the SECOs and it consists of the arrangement, coordination, and management of actors and networks at the ecosystem level [9].
- *Software supply network management* is needed in organising the collaborative production of products and services. It involves strategies about who are the suppliers and how to govern relationships and quality assurance with them [7].
- *Technology asset management* includes organisational-level strategic and operational decisions regarding how technology is organised to support the selective revelation of IPR.

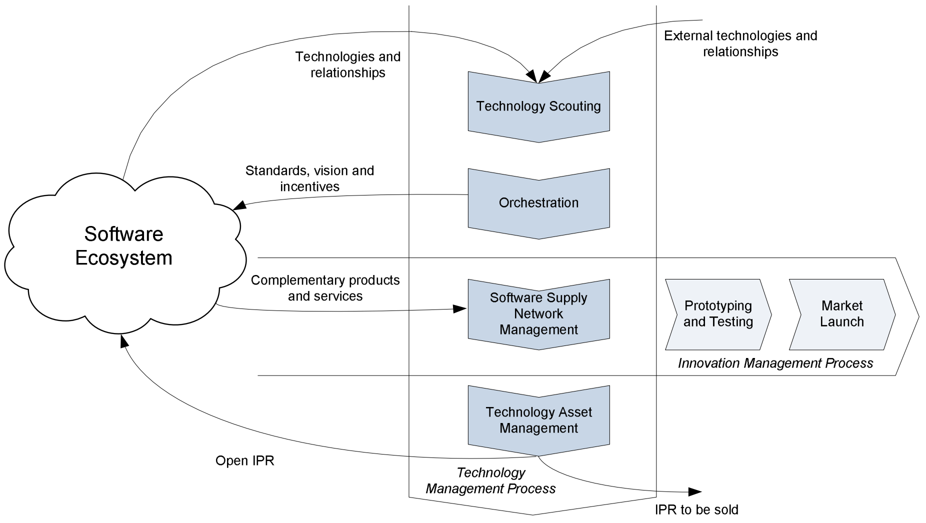

Technologies and relationships
External technologies and relationships
Technology Scouting
Standards, vision and incentives
Orchestration
Software Ecosystem
Complementary products and services
Software Supply Network Management
Prototyping and Testing
Market Launch
Innovation Management Process
Technology Asset Management
Open IPR
Technology Management Process
IPR to be sold

Fig. 1. The set of four management practices in the context of the technology and innovation management processes (adapted from [11])

Figure 1 illustrates the four management practices in the context of the technology and innovation management processes. The technology management process describes the development and life cycle of technology assets in a company. We see that the development of relationships is an important component in this process when operating in a SECO. In technology scouting, technologies and relationships are searched for and sourced from the ecosystem, related ecosystems and communities. Technologies and relationships are developed in orchestration resulting in commonly agreed standards, vision and incentives for SECO participants. Technologies and relationships are further developed in the collaborative development of products and services in software supply networks. In technology asset management, company's technology assets are stored for their potential use in future products and services. Thus feedback is possible to the previous management practices. A part of the technology assets is opened and returned to the ecosystem. Alternatively, the technology is kept internal or sold. In the innovation management process, software supply network management is used to assess and integrate complementary products and services. The resulting integrated platform is further prototyped and tested as a part of the products and services that are later launched to the markets.

2.1 Technology Scouting

Technology scouting is used to detect advances in technology at an early development stage including the identification and assessment of new technologies [12]. In addition, technology scouting plays a key role in technology sourcing [13]. Personal relationships established by the scouts for information-gathering purposes can be used when negotiating about joint research, licensing, buying IPRs, creating joint ventures, or the acquisition of start-ups. Organising technology scouting involves the building and use of networks of experts [11]. This requires to address the following three factors: 1) the goal of the technology scouting should be defined (directed or undirected scouting): 2) the incentive systems should be aligned carefully with all the actors in the process, and 3) the company should offer something in exchange for the information that is collected [11]. For example, Deutsche Telekom AG utilises internal and external scouts that are typically rewarded by money, while academic sources can be rewarded by recognition and joint research projects and the industry sources are offered by collaboration opportunities, for instance [11].

Utilising open source software ecosystems is an opportunity for system integrators. As well as being available free of charge, open source software can be of high quality. Building an effective open source software ecosystem should involve collaboration at the international level and the realisation of effective projects and communities in the field of the most strategic technologies combined with commercial success [14]. The company may choose only source solutions developed by others but active contribution enables a collaborative network to be created, which enriches the company's offerings to the market [14].

2.2 Orchestration

Orchestration is mainly keystone players' task [9]. The introduction of interoperable software standards is an important task in orchestration [15]. Other possible methods involve introducing quality standards and legislation about them, introducing certification programmes, the sharing of a SECO vision, and explicitly defining the boundaries of a SECO [9]. Samuel Fricker proposes requirements value chain analysis as a means of support for stakeholder management in understanding the relative power of stakeholders or the maturity of an ecosystem [5]. The requirements value chain analysis covers the inception of requirements, communicating them in a SECO, and solving conflicts regarding goals related to functionality and quality in order to formulate a set of agreed requirements with stakeholders. Supporting research fields that are identified are social network theory, group theory, and negotiation theory [5].

The applicability of orchestration methods depends on the characteristics of a SECO. In the case of a stable SECO (e.g, Visual Studio) where niche players are locked-in, the introduction strict rules may be appropriate, but in a more open SECO (e.g. Eclipse) that could scare developers [9]. In a young SECO there are usually bootstrap problems such as a lack of adopters of the keystone technology in order to get a return on investments [9]. To overcome this, the keystone technology should be made reusable through APIs or plug-in infrastructures [9]. After this niche players should be encouraged to be active in a SECO. There are several methods, such as revenue sharing or establishing partner networks [9].

The tipping strategy (with the product strategy) is applicable for system integrators if a dominant platform exists [16]. Tipping is a set of acts and strategic moves to shape market dynamics in favour of a particular technology and to win a platform war [16]. The tipping strategy consists of technology and business actions. The technological actions are [4]:

- try to develop unique features that are hard to imitate and attractive for users:
- tip markets by absorbing features from adjacent markets.

The business actions are [4]:

- offer more incentives for complementors than competitors do:
- build coalitions with competitors to defend against the entry of new platform wannabes:
- utilise pricing subsidy mechanisms to attract users.

Linux is a successful example of a tipping strategy. It consists of a large coalition of service provider companies and users. Technology scouting seems to support the tipping strategy in absorbing features from adjacent markets.

2.3 Software Supply Network Management

At the software supply network level there are challenges in identifying and selecting partners, the governance of relationships, and sorting out the dependencies between component developers [7]. Software supply networks cause particular risks in the innovation process. Ron Adner suggests that the risk management should be divided into three parts [17].

- Assessing the initiative risks of the project: evaluate the feasibility, customer value, competition, and capabilities of your own offering. Decide how risks are handled internally and which it is better to outsource.
- Assessing the interdependence risks of coordinating with complementary innovations: list whose projects need to be ready before yours can be. Estimate the probabilities of success for each key partner (by consulting with managers, double-checking with suppliers, and examining historical precedents) and multiply them to identify the overall probability of the success or delay.
- Assessing integration risks: identify who has to adopt the solution before the customer can. Adding up the estimated adoption cycles of intermediaries can be used in the integration risk assessment.

There are additional challenges in coordinating cross-organisational quality assurance and synchronising release timing [7]. External development teams cannot be subjected to standardised process models, tools, and ways of working [10]. This means that traditional process maturity approaches, such as CMMI (Capability Maturity Model Integration), are not well suited. Traditional centralised coordination, based on requirements, architecture, and software configuration management, which has worked for internal product lines is not appropriate for the ecosystem approach [10]. Centralised coordination of complex relationships tends to cause overheads that reduce competitiveness. Instead a decentralised composition-oriented approach is recommended [10]. In this approach coordination is done through software architecture principles rather than processes [18]. Moreover, component teams announce roadmaps and requirement specifications that are released at the end of the next iteration cycle, in contrast to the centralised top-down roadmap and requirement management [10]. A component team needs to reach out to other teams for local discussions on potential changes in component interfaces. Quality assurance should also be decentralised, since, as complexity and dependencies increase, centralised integration and quality assurance becomes a major effort [18]. To achieve a sufficient integration rate a software stack should support different release frequencies for different layers [10]. Backward compatibility is important in order to simplify the software configuration management and it requires loosely coupled component interfaces [10]. In other words, once a new component version is working, older versions or branches may not need to be maintained. In order to guarantee sufficient user experience and seamless integration for the overall system, a platform integrator has to provide a basic user experience framework and guidelines to achieve the composition-orientation [10].

2.4 Technology Asset Management

Technology asset management covers how technology is organised, stored and revealed by a company. A company needs to make internal decisions regarding how to design product technology strategically [3] and disclose IPR selectively [4]. A company may choose to open interfaces by offering APIs, open the source code, or open other IPRs, such as a requirement engineering process, road-maps, release times, customer and supplier information, bug repositories, or market research [9]. There is a natural tension associated with allowing external developers to become involved in the development process and, to some extent, even taking over part of the customer relationship [10]. A platform integrator may be concerned about gaining a negative reputation or the risk of the platform becoming irrelevant over time. A useful approach to avoid an undesired reputation while still sourcing new functionalities developed by external developers is to publicly release a long-term platform roadmap that indicates the intentions of the platform integrator company [10]. In addition, a company may develop innovative licensing methods, such as, for instance, open source commercial business models, and establish internal and external component markets [9].

The modularity of a functionality enables chosen components to be selectively disclosed [9]. Potential industry standards support software architects using uniform interfaces. A company may open the architecture gradually in order to remain able to develop the software without it becoming a maintenance nightmare [7]. Other practices involve designing reuse policies, creating a reuse enabling architecture, and supporting interchangeable data formats [9]. Opening up account system interfaces enables external developers to gain faster-paced access, as well as enabling content updates, feedback, licensing, and billing, for instance, to be automated [7].

3 A Case Example from the Telecom Industry

Telecom equipment manufacturers have traditionally organised themselves along product lines. Platforms used to be proprietary and mostly built internally from ASIC circuits to operating systems and middleware components. However, increasing competition among telecom equipment manufacturers has pushed down margins, which leads to requirements for a lower channel cost and shorter time-to-market [19]. In order to cut down costs and speed up development, a platform must be created from standardised components. Then components can be sourced from COTS (Commercial Off The Shelf) vendors for competitive prices and they may already be available in the markets when needed in products. The standardisation of components enables components to be replaced by more competitive ones, if necessary, without a lock-in to one vendor, and, additionally, this supports low integration and verification costs. For instance, Ericsson and Nokia Siemens Networks leverage basic platform technologies and add value in the application domain. They have adopted ecosystem strategies, as well as an integrator role for building platforms.

3.1 Technology Scouting

Nokia Siemens Networks, since found (2007), has researched ways to facilitate systematic technology scouting in order to find emergent software technologies. A reason for this is several common trends occurring in the telecom and IT platforms, which call for technology scouting to identify inter-industry opportunities and threats. Convergent technology and market needs are exemplified by the generalising IP-based (Internet Protocol) services such as IMS (IP Multimedia Subsystem), and Unified Communications. Technology scouting is focused on the activities from the definition of the search areas to data interpretation. The outcome is communicated, for instance, to product lines for decision making. Technology sourcing is not explicitly linked to technology scouting. A scouting network consisting of internal and external experts including university researchers collects data in the three modes: 1) non-focused scouting done continuously company-wide and communicated through web 2.0 tools (blogs), emails, and informal expert networks: 2) focused scouting on strategic areas done by responsible experts and communicated by strategy materials, and 3) focused intelligence studies typically triggered by product lines.

3.2 Orchestration

Nokia Siemens Networks orchestrates selected ecosystems in order to contribute to their development directions. It is shown that the tipping strategy is used for shaping market dynamics in favour of particular ecosystems and technologies. For instance, this involves contribution in standardisation groups and industry consortia in order to agree specifications together with potential competitors. Active marketing efforts are used to create credibility around emergent ecosystems. A vision of an ecosystem is communicated, for instance, in worldwide industry conferences. This helps smaller participants to align their strategies and identify new business opportunities. In addition, different events provide valuable connections to partner networks.

Telecom equipment manufacturers convey requirements from telecom operator customers to other ecosystem participants, who may have insufficient domain knowledge to realise them alone. It appears to be challenging to specify and communicate requirements so that supplied components are easily integrated. Especially, non-functional requirements may have a great impact throughout the platform architecture and reworking them may affect on many components. The most important non-functional requirements in a telecom platform include availability, scalability, and serviceability.

A modular architecture with well-defined interfaces is needed in order to integrate a platform from externally developed components and achieve the interoperability of components. Figure 2 presents a common telecom platform architecture that contains standardised and proprietary components. Currently, the following prominent and relatively stable industry standard ecosystems can be identified, which have emerged around certain components in the telecom platform architecture [19].

- Hardware platform (ATCA): the Advanced Telecommunications Computing Architecture is a set of specifications defined by the PCI Industrial Computer Manufacturers Group [20]. The ATCA is a modular hardware platform designed for scalability and carrier-grade availability (that is services are running 99.999% of the time). The evolution of the ATCA can be compared to the evolution of the standardised IT rack mountable hardware platform.

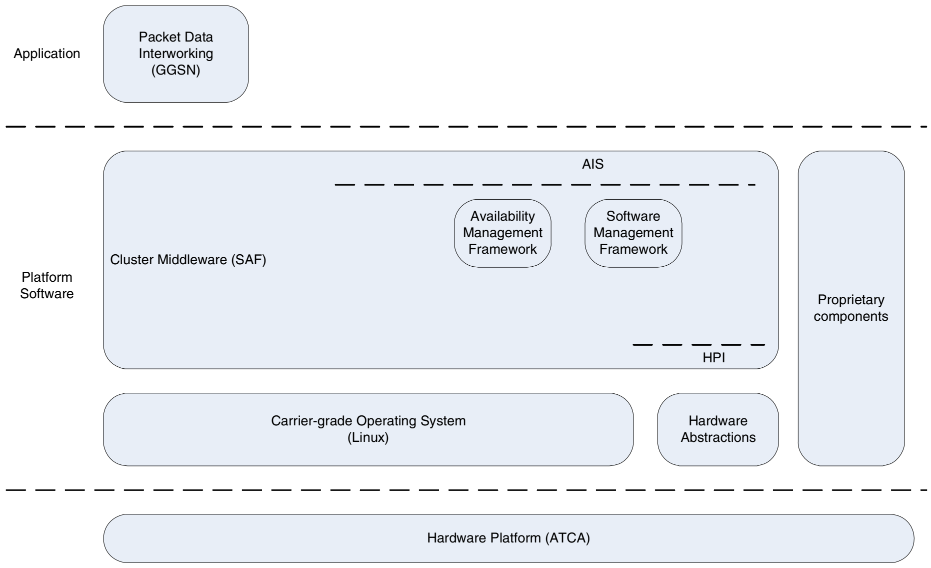

Application
Packet Data Interworking (GGSN)
Platform Software
AIS
Availability Management Framework
Software Management Framework
Cluster Middleware (SAF)
HPI
Proprietary components
Carrier-grade Operating System (Linux)
Hardware Abstractions
Hardware Platform (ATCA)

Fig. 2. A telecom platform architecture

- Carrier-grade operating system (Linux): by the acceptance in the server market, the telecom equipment vendors also started seriously considering Open Source Linux for telecom systems. The Open Source Development Lab's carrier-grade Linux working group was established to collect requirements from network and telecom equipment providers and independent software vendors, specifically for Linux [21].
- Cluster middleware (SAF): the Service Availability Forum™ is a consortium of industry-leading communications and computing companies working together to develop and publish open software interface specifications for services requiring high availability. The two main specifications are the Hardware Platform Interface (HPI) and Application Interface Specification (AIS) [22]. The hardware abstractions encapsulate hardware (such as temperature and voltage sensors) coming from different vendors so that they can be integrated into the HPI in a common way. The AIS is designed to support applications ranging from web servers to telecom applications like GGSN (Gateway GPRS Support Node) [19]. The Availability Management Framework (AMF) and Software Management Framework (SMF) are examples of AIS Services.

3.3 Software Supply Network Management

Nokia Siemens Networks has developed an assessment method for third-party software component selection during 2001-2006 [23]. The simplified software development process contains the following phases [23]:

- Design & implementation – the supplier designs and implements components based on the standard specifications
- Assessment – candidate implementations are assessed on the basis of functional and non-functional requirements as specified in the standards
- Integration – the chosen component is integrated into the product
- Operation – the product is deployed in a global volume and is under maintenance

In terms of risk management the assessment phase presents the estimation of the probability of success for each key partner candidate. Picking a sub-optimal implementation may cause considerable costs in the later phases of the life cycle. Conformance testing (IEEE 2003-1997) has been applied to test robustness of the AIS Services such as AMF and the process was feasible, repeatable, and reusable [23].

Feedback and reintegration mechanisms are seen to be critical and developed at least on project or supplier basis in order to handle fault situations. Typically, there are some shortcomings in standard specifications co-created in orchestration and new requirements appear during the integration phase. The interoperability of components is verified sometimes by hands-on plug-fests. Sometimes, a platform integrator can fix a shortcoming in the supplied software but occasionally a correction is needed quickly from the supplier. Usually, direct engineer-to-engineer communication between component teams, as suggested in the decentralised composition-oriented approach, is the fastest way to solve problem. Greater shortcomings may require renegotiation and road-mapping with a supplier.

3.4 Technology Asset Management

Nokia Siemens Networks has recently started to improve software reuse systematically by asset management services that provide better asset visibility through the company and with potential external collaborators. By the software reuse, the expansion of the software base is tried to keep manageable when new products and product variants are created. The number of maintained platforms is restricted and consolidated across product lines by dividing products into groups on the basis of their cost structure and technical requirements. The number of technology standards and partners that are used is also restricted for the sake of simplicity. In order to support the easy integration of third-party software components, software architectures are developed to be flexible and modular enough. Proprietary software development tools are replaced by commonly used ones, which are preferably available as open software. This allows the gradual opening of software repositories, version control systems, and continuous integration builds (such as SVN and CruiseControl). Code validation tools are also used for legal checks on migrated open source software.

4 Conclusions

In this paper, we presented a structured overview of the four management practices that support platform integrators operating in software ecosystems. The four management practices identified from the literature are technology scouting, orchestration, software supply network management, and technology asset management. These management practices are linked to the technology and innovation management processes, which may help to systematically respond to challenges in software ecosystems.

The assumption is that a company has a strategic intent to operate in software ecosystems. A discussion about whether or not the ecosystem approach is appropriate for certain businesses is another relevant topic. We focused on the technology management perspective and all business management practices were not covered. The presented management practices are seen to be equally important for a company's success and they should be aligned according to technology and business strategies. We made the first attempt to validate the set of the four management practices by the case example in the telecom industry. It should be noted that these management practices are not specific for the telecom industry but we offer them for platform integrators that are keystone players in other industries as well. Our case example indicates that a platform integrator in the telecom industry utilises the four management practices. Technology scouting is organised for finding new technologies. By orchestration, the development of ecosystems is directed, especially, towards the common standards-based platform architecture. Software supply network management is used for managing supplier risks and integrating platform components. Technology asset management prepares for opening repositories by architectures supporting software reuse and by using standard development tools. In the case example, the management practices are not utilised in the extent presented in the literature synthesis of this paper. This does not mean that it has been the objective but both viewpoints can enlighten each other. The synthesis helps to optimise the management practices together to support holistic technology management when operating in software ecosystems. One of our future research objectives is to validate the management practices by means of case studies. Our aim is to identify the success factors and challenges related to the application of the management practices.

References

1. Pohl, K., Böckle, G., Linden, F.J.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer, Heidelberg (2005)
2. Cusumano, M.: Technology Strategy and Management: The Evolution of Platform Thinking. Communications of the ACM 53(1), 32–34 (2010)
3. Gawer, A., Cusumano, M.: Platform Leadership: How Intel, Microsoft, and Cisco Drive Industry Innovation. Harvard Business School Press, Boston (2002)
4. Gawer, A., Cusumano, M.: How Companies Become Platform Leaders. MIT Sloan Management Review 49(2), 29–30 (2008)
5. Fricker, S.: Requirements Value Chains: Stakeholder Management and Requirements Engineering in Software Ecosystems. In: Wieringa, R., Persson, A. (eds.) REFSQ 2010. LNCS, vol. 6182, pp. 60–66. Springer, Heidelberg (2010)
6. Rochet, J., Tirole, J.: Two-Sided Markets: A Progress Report. RAND Journal of Economics 37(3), 645–667 (2006)
7. Jansen, S., Finkelstein, A., Brinkkemper, S.: A Sense of Community: A Research Agenda for Software Ecosystems. In: 31st International Conference on Software Engineering, New and Emerging Research Track, pp. 187–190 (2009)
8. Iansity, M., Levien, R.: Keystones and Dominators: Framing Operating and Technology Strategy in a Business Ecosystem. Harvard Business School, Working Paper #03-061 (2004)

9. Jansen, S., Brinkkemper, S., Finkelstein, A.: Business Network Management as a Survival Strategy: A Tale of Two Software Ecosystems. In: Proceedings of the First Workshop on Software Ecosystems. CEUR-WS, pp. 34–48 (2009)
10. Bosch, J.: From Software Product Lines to Software Ecosystems. In: Proceedings of the 13th International Conference on Software Product Lines, SPLC (2009)
11. Rohrbeck, R.: Technology scouting – a Case Study of the Deutsche Telekom Laboratories. In: ISPIM-Asia 2007 Conference, pp. 1–14 (2007)
12. Brenner, M.S.: Technology Intelligence and Technology Scouting. Competitive Intelligence Review 7(3), 20–27 (1996)
13. Wolff, M.F.: Scouting for Technology. Research Technology Management 35(2), 10–12 (1992)
14. Ruffatti, G.: SpagoWorld, the Open Source Initiative by Engineering. The European Journal for the Informatics Professional X(3), 44–50 (2009)
15. Bannerman, P.L., Zhu, L.: Standardization as a Business Ecosystem Enabler. In: Feuerlicht, G., Lamersdorf, W. (eds.) ICSOC 2008. LNCS, vol. 5472, pp. 298–303. Springer, Heidelberg (2009)
16. Gawer, A., Cusumano, M.: Strategy Toolkit for Platform Leader Wannabes. In: DRUID Summer Conference 2007 on Appropriability, Proximity, Routines and Innovation, pp. 1–33 (2007)
17. Adner, R.: Match Your Innovation Strategy to Your Innovation Ecosystem. Harvard Business Review 84(4), 98–107 (2006)
18. Bosch, J.: The Challenges of Broadening the Scope of Software Product Families. Communications of the ACM 49(12), 41–44 (2006)
19. Kamalvanshi, A., Jokiaho, T.: Build the Next Generation of Telecom Systems with Open Interfaces, Part 2. Commsdesign (2005), http://www.commsdesign.com/design_corner/showArticle.jhtml?articleID=163700304 (accessed on May 27, 2010)
20. PCI Industrial Computer Manufacturers Group, PICMG 3.0 AdvancedTCA^{TM} Base Specification, http://www.picmg.org/v2internal/specifications.htm
21. The Linux Foundation, Carrier Grade Linux Workgroup, http://www.linux-foundation.org/en/Carrier_Grade_Linux
22. Service Availability Forum, Application Interface Specification, http://www.saforum.org
23. Francis, T.: Service Availability Standards for Carrier-grade Platforms: Creation and Deployment in Mobile Networks. PhD thesis, Tampere University of Technology (2009)

Steering Insight: An Exploration of the Ruby Software Ecosystem

Jaap Kabbedijk and Slinger Jansen

Utrecht University
Department of Information and Computing Sciences
Princetonplein 5, 3584CC, Utrecht, Netherlands
{j.kabbedijk,s.jansen}@cs.uu.nl

Abstract. Software products are part of a larger network of products, suppliers and partners, called a *software ecosystem*, working together in order to provide functionality for the users and generate profit for the vendors. Not much is known about the characteristics and relationships within such a software ecosystem. This paper presents an overview of the open source Ruby ecosystem and lists its elements, characteristics, descriptives, roles, cliques and relationships. Data is gathered using the Git decentralized source code management system and is analyzed using social network and statistical analysis techniques. Our analysis shows that the Ruby ecosystem exists out of a couple very distinctive roles developers fulfil. It also shows that within the Ruby ecosystem only a small 'core' of approximately 10% of all developers and gems (Ruby packages) are dominant within the ecosystem. At this point in time it appears that the rails community would benefit from motivating current developers to work together more, instead of supporting new developers or gems in order to get a healthy ecosystem.

Keywords: software ecosystem, Ruby, ecosystem governance, exploratory case study.

1 Introduction

As a software vendor, your profit is not determined by one independent product, but by all parties and products related to your product. The development of software has a lot of differences compared to how physical products (e.g. bread or furniture) are created. First of all software is not one physical product, but a product that can be multiplied an infinite amount of time without substantial extra costs [1]. Second and most importantly, the total value of a software product is determined by sum of all additional products using the software product. The Android mobile phone operating system for example, has a limited value as a product on its own. The overall value of Android is determined by all the applications built on top of Android that extend the possibilities of the main software product (software platform). This network of all products, companies and services working together in one big network is called a Software Ecosystem. The term Software Ecosystem (SECO) was first coined by Messerschmitt and Szyperski in 2005 [2], but it took until 2009 before a clear definition was formed by Jansen, Finkelstein and Brinkkemper [3], who define a SECO as "a set of businesses functioning as a unit and interacting with a shared market for software and services, together with the relationships among them".

"Making profit from your software product" became "making profit from your software ecosystem" [4]. Profit is not generated only by one product or service, but by all companies together making use of one core service to deliver their own products. The popularity of a product created by one company can benefit all companies and products in the same SECO. Software companies are increasingly becoming aware of this development and start to recognize their position within the SECO. Software vendors face new problems, as they need insight in their software ecosystems. They need to know the dynamics within their SECO, the participants and how to steer their SECO in order to generate the most profit. Jansen et al. [3] state that to the extent of getting a good insight into a SECO, characteristics must be quantified and measured. Possible characteristics are the number of sub communities, the reciprocity of the ecosystem or the outdegree of keystone actors. Iansiti and Levien [5] state that: "the performance of a firm is a function not only of its own capabilities or of its static position with respect to its competitors, customer, partners and suppliers, but of its dynamic interactions with the ecosystem as a whole". This statement emphasizes the importance of getting an overview of the entire SECO in order to judge the position of one participant in it.

Open source projects are typical environments in which SECOs develop around the community. By open source we mean that the source code of the project is available for download and everyone can participate in extending or adapting the code [6]. Among Free Open Source Software (FOSS) projects, relationships are frequently seen in the form of developers coding on multiple projects [7]. These developers are linchpins [8] between the different projects and are an important reason SECOs exists within FOSS. Since everyone can participate in coding for a specific project, large projects and with that, large interconnected SECOs come into play. Because of the size and level of transparency of these FOSS SECOs, they are ideal for statistical and network analysis, since a high sample size increases the significance of found results and improves the external validity [9]. Also, the use of large central repositories for storing all related projects enables a convenient way for gathering all data needed [10].

In this paper we first explain the main research question, including all sub questions related to this question in section 2. In (section 3) the Ruby case that was analyzed is described and the data gathering is explained. The research questions asked are analyzed into depth in section 4 and the results are presented in section 5. We conclude with the conclusion and discussion, followed by future research in section 6.

2 Research Questions

The main research questions answered in this paper is: **"What are the defining characteristics of a large scale open source software ecosystem?"**.

From the main research question, the following sub questions are derived:

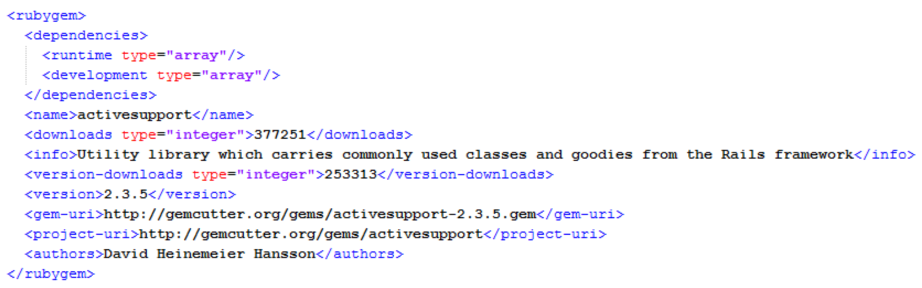
1. **What elements can be identified within a SECO?** - An ecosystem exists out of several different elements interacting with each other. We expect different *types* of elements to exist within a SECO, each having different *functions* within the SECO. This question will be answered by looking at the description of all elements in the repository.
2. **What are the characteristics of the identified elements?** - Each of the defining characteristics for the SECO are required for the analysis of an element. A characteristics is defined as a prominent attribute of a SECO element.
3. **What are the descriptives of a SECO?** - In order to get a good overview of the SECO we are analyzing, we first need to explore the elements and characteristics within a SECO into some more depth. We focus on the software supply network level [3] when answering this question. Descriptives describe the main characteristics of a collection of the SECO elements quantitatively [11].
4. **What roles can be identified within a SECO?** - The elements existing within a SECO have different roles depending on their position in the SECO. We will identify several distinctive roles within a SECO based on their interconnectivity.

These sub questions will be answered in the following sections, concluded with the answer to the main research question in section 5.

3 Case Description and Data Gathering

In this research the relationships, dynamics and characteristics of the Ruby Ecosystem are analyzed. Ruby is a popular programming language, using the MVC design principle [12] and trying to combine a scripting language's simple immediacy (e.g. PHP) with a strict object oriented architecture (e.g. Java) [13]. The framework consists out of thousands of possible parts that can be combined to get exactly the functionality your project needs. The 'parts' are called 'gems' in Ruby jargon and are hosted and maintained through Git. Git is a Decentralized Source Code Management (DSCM) system [14] that uses decentralized repositories, owned by developers, that can be mined for analysis in the case of Ruby. Because all gems related the the Ruby programming language are released under an open source license [15], the entire Ruby ecosystem is a collection of FOSS projects. Gems are developed by different developers and everyone can create as many gems as he wants. All gems are indexed on a central place and users of the Ruby programming language can decide which gems they need for their project development.

All data for this project was gathered by using an XML [16] overview of all gems present in the Ruby Git repository on *15-02-2010*. This XML contained information on each gem such as the name, dependencies, author, etc. Please see figure 1 for an excerpt of the XML file showing the XML data for one gem.



```xml
<rubygem>
  <dependencies>
    <runtime type="array"/>
    <development type="array"/>
  </dependencies>
  <name>activesupport</name>
  <downloads type="integer">377251</downloads>
  <info>Utility library which carries commonly used classes and goodies from the Rails framework</info>
  <version-downloads type="integer">253313</version-downloads>
  <version>2.3.5</version>
  <gem-uri>http://gemcutter.org/gems/activesupport-2.3.5.gem</gem-uri>
  <project-uri>http://gemcutter.org/gems/activesupport</project-uri>
  <authors>David Heinemeier Hansson</authors>
</rubygem>
```

Fig. 1. XML excerpt

By using the URI for the gem, the gem was downloaded from RubyGems[1] and stored in a a relational database. All additional meta information on the gem like the version of the gem release and developers who worked on the gem were stored in the database as well. This data gathering was done by using XSLT to get the right URI and the PHP scripting language to download the gems. The gems were downloaded to be able to analyze the source code of the gems and not only the meta information available of the gems. All data was stored in the database by using PHP scripting combined with SQL statements.

Before the data was usable for analysis, it had to be checked and reformatted. Due to errors in the XML source file and small glitches during data gathering, some data are incomplete or duplicated. We used the PASW analytics suite to identify errors and irregularities in the dataset[2] and corrected the gathered data based on this. The alterations to the original data are documented in our case study protocol [9].

4 Analysis

In this section an analysis of the Ruby dataset is provided, so each question posed in section 2 can be assessed. The numbering of the sub sections matches the numbering of the research questions in section 2 and each sub section analyzes the corresponding question.

4.1 Elements

The units of analysis in the Ruby SECO are gems and developers, with the possible relationships among them. If a developer has a relationship with a gem, he is a developer of that specific gem. If a developer has a relationship with another developer they have worked together on a certain gem and if a gem has a relationship with another gem, they are dependent on one another. The dependency relationship among gems can have two different types, as can be seen in figure 1. Dependencies between gems can either be runtime dependencies or

[1] Ruby community's official gem hosting service. Available at http://rubygems.org
[2] The complete dataset can be downloaded from http://softwareecosystems.org

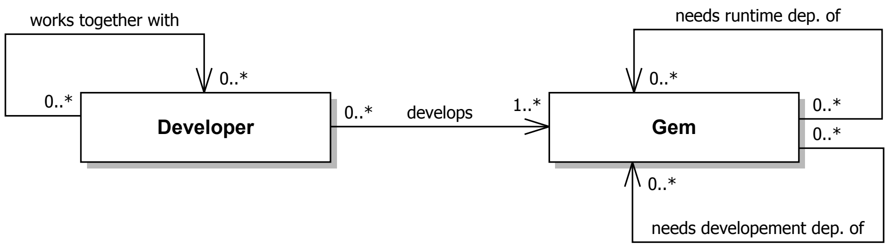


Fig. 2. Metamodel of interacting SECO elements

development dependencies. The first type is in place if a gem needs another gem in order to work properly for the end-user. The second type of dependency is relevant when a gem needs another gem only for development purposes, but not to work properly at runtime. An overview of all elements and their relationships can be found in figure 2.

4.2 Element Characteristics

Numerous characteristics are related to the SECO elements described above. For all gems we identified the following characteristics relevant for analysis, based on the information in the XML and the additional information available by downloading and examining the source code of the gem:

– **Name** - This is an unique identifier used to give a name to the particular gem, but also to be able to differentiate this gem from other gems. We use this characteristic only for identifying purposes.
– **Number of Downloads** - This number is an indication of the popularity of the gem. The more a gem is downloaded, the more popular it probably is. This popularity can for example be caused by the fact that this gem fulfils a core functionality and is used as a dependency by a lot of other gems.
– **Main Version** - This version number is a indicator of the maturity of the gem. Ruby uses sequence-based software versioning scheme in which a 0 as starting digit indicates a beta state and 1 or higher indicates a more mature state of the gem (for example 1.0.6). We only looked at the starting digit of the versioning sequence. Please note that the digits are not comparable with each other; version 2 on one gem can indicate another level of maturity than version 2 on another gem, since developers can decide themselves when to increase their version number.
– **Lines of Code (LOC)** - The LOC is an important indicator of the effort that was put in developing a gem. Ruby gems have two types of code that are present in the source of the gem. Besides the code that provides the functionality for the gem, also test code in embedded in the gem source. For the sake of our analyses we looked at the total lines of code. This includes both the functionality LOC and the test LOC.

Table 1. Ruby Ecosystem descriptives

Characteristic	Minimum	Maximum	Average	SD	Median
Downloads	3	377,251	1,159	10,710	123
Yahoo Hits	0	21,500,000	167,057	745,156	334
Size	0	25,851,477	76,362	422,091	16034
Lines of Code	0	427,736	2,059	8,544	513

- **Size** - This characteristic indicates the amount of bytes a gem uses. The characteristic can also be used as an indicator of the total effort put in developing the gem, but this can be deceiving due to the fact that gems can exist out of more than only code (i.e. images).
- **Yahoo Hits** - The amount of hits that were generated by giving "$name of the gem + 'ruby gem"' as input to the Yahoo search engine. This is also an indicator of the gem popularity, since a popular gem is more likely to be named or linked on multiple locations on the web instead of only from one location.

The characteristics listed above will be used as variables in further sections of the paper.

4.3 Descriptives

First the entire ecosystem including all developers, gems and their relationships is visualized within one big graph using the network visualization software Gephi [17], as can be seen in figure 3[3]. The visualization gives a clear indication of the size of the Ruby ecosystem and also shows a lot of small projects and some larger interconnected projects. The structure of the ecosystem clearly indicates some sort of ecosystem 'core' of active developers, as can be seen by the interconnected 'stem' in the middle of the visualization.

The ecosystem consists of *4,784* developers, *10,046* gems and *13,103* relationships between them. A gem is developed by an average of *1.23* developers with a standard deviation of *0.725* and each developer in the ecosystem has developed an average of *2.53* gems with a standard deviation of *3.95*. A selection of some of the most important descriptives of the ecosystem is reported in table 1.

In order to get insight in the Ruby ecosystem we first examined the boundaries of the ecosystem by analyzing the most popular gems and most active developers. We calculated the *eigenvector centrality* for all developers within the ecosystem to get a good overview of the level of importance of the developer. The eigenvector centrality is used within network analysis as a measure to indicate the importance of a node in the network [18]. For the sake of understandability of the graph, only the top 30 developers in terms of degree [19] were included.

[3] A high resolution version of the figure is available at http://softwareecosystems.org

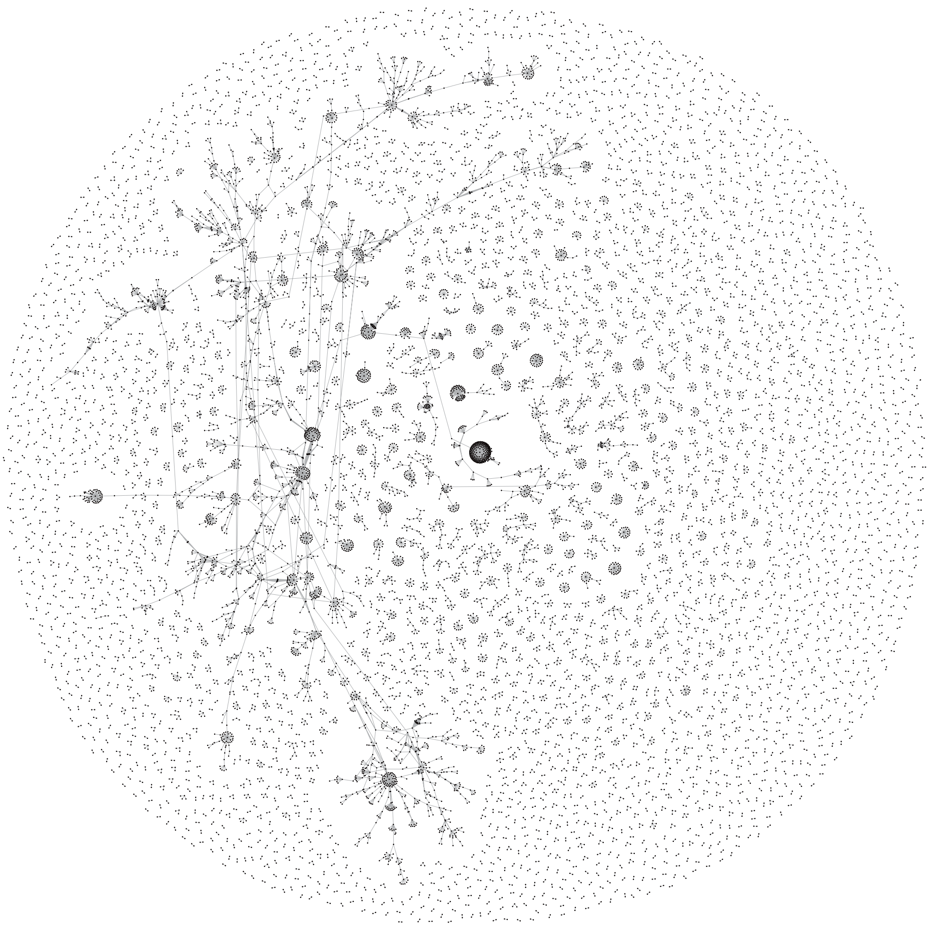
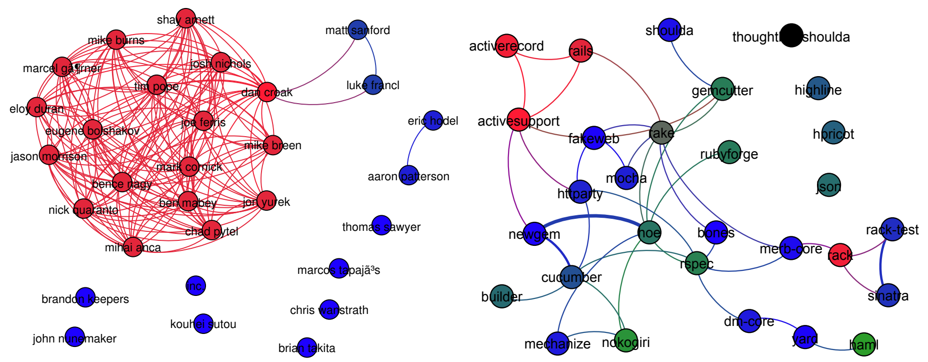


Fig. 3. Ruby Ecosystem Visualization

Figure 4a shows the network of interacting developers with a range of degree between 15 and 30. Notable is the concentration of nodes on the left, which are all interconnected and indicate a dense interaction of a large number of most active developers. Figure 4b shows the top 30 of interdependent gems. This top is composed by selecting the gems that have the highest number of gems depending on them. We added up both runtime and development dependency for calculating this top 30. As can be seen, most crucial gems also have interdependencies among each other, indicating a strong network of important gems within the Ruby ecosystem.

All of the developers shown in figure 4a are in the top 50 op highest eigenvector of centrality score and have an eigenvector centrality score of between 1 and approximately 0.1. The meaning of this will be explained in section 5.



(a) Top 30 of interacting developers (b) Top 30 of interdependent gems

Fig. 4. Overview of top 30 gems and developers

4.4 Roles

Within the Ruby ecosystem several different roles for developers are identified based on the degree of the developer (number of other developers he cooperates with), the number of gems made and the popularity of the gems.

The first role identified is the **Lone Wolf**. The role of Lone Wolf is based on the role of a 'Niche Player' [5], Specific to this context it is defined as a developer who has developed gems that are of importance within the Ruby ecosystem, but has almost no connections with other developers. He produces useful content for the ecosystem, but works solitary. The importance of a gem can be determined in different ways. First the number of downloads of all gems a developer made combined are of importance, after this the amount of gems that are dependent on a gem play a role. Finally the amount of gems created by someone is of importance in addressing the Lone Wolf role. Table 2 shows the top 5 of Lone Wolfs in the Ruby ecosystem. The ranking is based on the number of dependent gems as most important qualifier. These 5 developers mean a lot for the SECO, but are not related to any other developers.

The second role we identified is the role of **'Networker'.** The Networker role is based on the keystone role [5], but is defined for this specific case as someone who has a lot of developers he works with and also plays a large role in the

Table 2. Lone Wolf Top 5

Developer	Number of downloads	Dependent gems	Number of gems
David Heinemeier Hansson	2,056,351	2146	13
Loren Segal	7.123	295	10
Bob Aman	53,198	268	10
Makoto Kuwata	73,874	180	17
Zed A. Shaw	114,546	164	9

Table 3. One Day Fly Top 3

Developer	Gem	Version Number	Number of downloads
Wayne Meissner	FFI	0.6.1	46,222
Philip Ross	tzinfo	0.3.16	22,398
Benjamin Curtis	faker	0.3.18	13,522

SECO in terms of gem downloads. The top 30 of Networkers can be found in figure 4a. Besides Networkers we can also identify the so called **'One Day Flies'**. These developers have made one popular gem, but never made anything else. The criterion for being classified as a One Day Fly is: only one gem created with a version number starting with '0' and being in the top 5% of most number of downloads. A listing of the top 3 One Day Flies can be found in table 3.

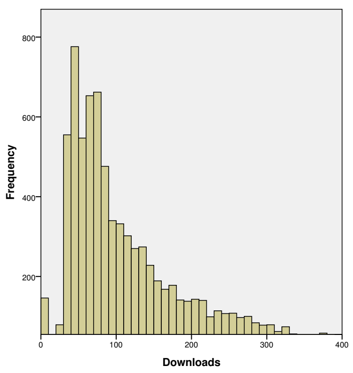
All different roles identified each have different functions in the Ruby ecosystem, as will be further discussed in section 6.

5 Results

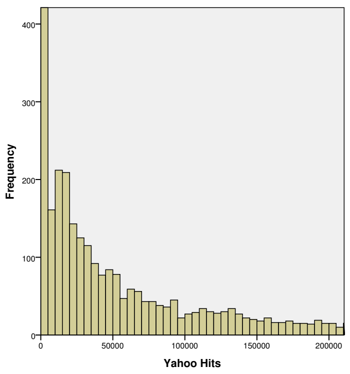
As section 4.1 showed, developers, gems and the relationships among them play a crucial role in the Ruby SECO. The characteristics playing a role in our analysis were listed and explained in section 4.2. For answering the question on SECO descriptives, table 4.3 shows the average number of downloads of a gem was *1,159*. This does not mean, in this case, that an average gem has approximately one thousand downloads. The height of the Standard Deviation (SD) shows us that the individual number of downloads per gem differs significantly from the average. This conclusion is shown clearly in figure 5a, in which a high number of gems can be seen with less than 100 downloads. Around 90% of all gems has a number of downloads below the average, meaning that there is only a 10% of all gems responsible for the high number of downloads.

A similar situation can be seen when looking at the amount of hits gems got in the Yahoo search engine. The SD of *745,156* again is much higher than the average number of hits of *167,057*, meaning that there is a skewed distribution. Figure 5b[4] shows that the distribution of hits on Yahoo is indeed skewed and the number of gems that is has an amount of hits below the average is again around 90%. Since we use the amount of hits on Yahoo as an indication of popularity of a specific gem, this statistic says a lot about the distribution of popularity among gems. Figure 5c[4] and 5d[4] both show a similar distribution and indicate the same effect as the amount of downloads and the number of Yahoo hits did; a small part of the Ruby SECO in this case take care of the largest gems in terms of bytes and lines of code. Finally in section 4.4 we could identify three distinct roles within the SECO, each being of high importance for the SECO.

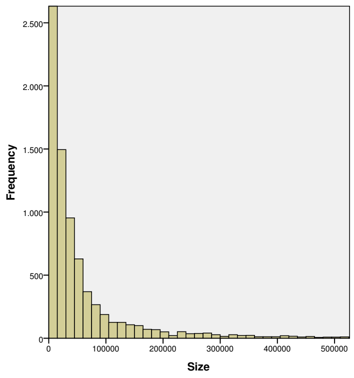
[4] One should note that the bar at '0' should actually be extended to a higher number, but is cropped due to readability.



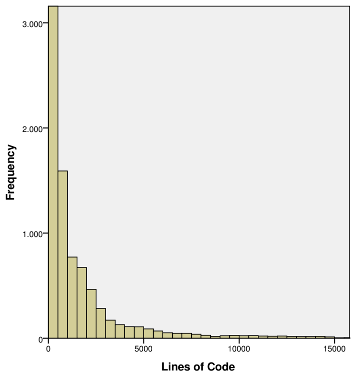
(a) Number of Downloads



(b) Number of Yahoo Hits



(c) Size



(d) Lines of Code

Fig. 5. Gem Characteristics Histograms

The existence of these roles could indicate that there is not one ‘holy grail’ in governing a SECO, since there are different types of developers.

6 Discussion and Conclusion

The conclusions of this work are as follows: **(1)** the Ruby SECO consists of developers, gems and relationships and **(2)** developers within the SECO fulfil several distinctive roles, each of different value to the ecosystem. **(3)** Also, within the SECO most activity is caused only by a small part of the ecosystem. the top 90% of the open source components used in Ruby development has been developed by only 10% of the total number of open source contributors.. The value of this knowledge lies in deciding how to better manage or steer SECO governance. **(4)** Trying to lure additional developers to your ecosystem in order to expand your ecosystem may not be the best way of managing a SECO; motivating existing developers to work together more on existing gems is a better way to get a solid and healthy ecosystem. On the other hand, some of the most popular gems are developed by lone wolfs, so this conclusion should be investigated in more depth in future longitudinal research.

For the dataset we analyzed, we were dependent on the 'snapshot' of gems available on Github. This dependency caused us to not being able to analyze everything the way we would have wished. The developments and growth within the SECO could not be researched by the static dataset we had. Because of this, no strong conclusions can be made on how to stimulate growth or health. Also we only performed one case study, so our results can not be generalized to draw conclusion on FOSS SECOs in general. On the topic of analysis tools, the large size of the dataset resulted in very long computation times, making analysis sometimes difficult and cumbersome.

Due to the lack of longitudinal data it is impossible to speculate about the dynamics of the Ruby SECO. In the future, we plan to follow specific developers and clusters in the ecosystem. Methods such as structural break analysis can be applied to monitor SECOs, to early discover the specific events that have happened in the SECO. Presently an extensible tool is being developed that is able to mine several different repositories, so future repositories can be mined as well. The tool will be able to add the element 'time' to our analysis and can also give additional case studies rather easily. This will enable us to do more longitudinal research on the topic of software ecosystem evolution in the future and generate more generalizable results on software ecosystems in general.

Acknowledgements. The authors would like to thank Bernard Verhoeven for supplying us with the data used for our analyses and his useful suggestions.

References

1. Xu, L., Brinkkemper, S.: Concepts of product software. European Journal of Information Systems 16(5), 531–541 (2007)
2. Messerschmitt, D.G., Szyperski, C.: Software ecosystem: understanding an indispensable technology and industry. The MIT Press, Cambridge (2005)
3. Jansen, S., Finkelstein, A., Brinkkemper, S.: A sense of community: A research agenda for software ecosystems. In: 2009 31st International Conference on Software Engineering Companion Volume, pp. 187–190 (2009)
4. Popp, K.M., Meyer, R.: Profit from Software Ecosystems. Books on Demand GmbH (2010)
5. Iansiti, M., Levien, R.: The keystone advantage: what the new dynamics of business ecosystems mean for strategy, innovation, and sustainability. Harvard Business Press, Boston (2004)
6. Feller, J., Fitzgerald, B.: Understanding open source software development. Addison-Wesley Longman Publishing Co., Inc., Boston (2002)
7. Scacchi, W., Feller, J., Fitzgerald, B., Hissam, S., Lakhani, K.: Understanding free/open source software development processes. Software Process: Improvement and Practice 11(2), 95–105 (2006)

8. Madey, G., Freeh, V., Tynan, R.: Modeling the free/open source software community: A quantitative investigation. Free/Open Source Software Development, 203–220 (2004)
9. Yin, R.K.: Applications of case study research, 2nd edn. Sage, Thousand Oaks (2003)
10. Kagdi, H., Collard, M.L., Maletic, J.I.: A survey and taxonomy of approaches for mining software repositories in the context of software evolution. Journal of Software Maintenance and Evolution: Research and Practice 19(2), 77–131 (2007)
11. Ross, S.M.: Introductory statistics. Academic Press, London (2005)
12. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design patterns: elements of reusable object-oriented software. Addison-Wesley, Reading (1995)
13. Bächle, M., Kirchberg, P.: Ruby on Rails. IEEE Software, 105–108 (2007)
14. Bird, C., Rigby, P., Barr, E., Hamilton, D., German, D., Devanbu, P.: The promises and perils of mining git. In: 6th IEEE International Working Conference on Mining Software Repositories, pp. 1–10 (2009)
15. Rosen, L.: Open source licensing: Software freedom and intellectual property law. Prentice-Hall PTR, Upper Saddle River (2004)
16. W3C: Extensible markup language (xml) 1.0 (fifth edition) (November 2008)
17. Bastian, M., Heymann, S., Jacomy, M.: Gephi: An open source software for exploring and manipulating networks. In: International AAAI Conference on Weblogs and Social Media (2009)
18. Bonacich, P.: Some unique properties of eigenvector centrality. Social Networks 29(4), 555–564 (2007)
19. West, D.B.: Introduction to graph theory. Prentice-Hall, Englewood Cliffs (2001)

Study of the Competition between Proprietary Software Firms and Free/Libre Open Source Software Firms Using a Simulation Model

Luisanna Cocco, Katiuscia Mannaro, Giulio Concas, and Michele Marchesi

University of Cagliari, DIEE - Dipartimento di Ingegneria Elettrica ed Elettronica, Piazza D'Armi, 09123 Cagliari, Italy
{luisanna.cocco,mannaro,concas,michele}@diee.unica.it

Abstract. In recent years, a very important structural change in the software industry took place, with an increasing number of firms that got involved in Free/Libre Open Source Software (FLOSS) development communities. FLOSS communities and products have been studied as complementary to proprietary software companies and products. In this paper we propone a business model for the software market, and in particular we analyze the competition between proprietary software firms and FLOSS firms. Our software market is a system where each agent is independent of each other in the choice about buying or selling software products or services. The proposed work aims to analyze the influence of FLOSS firms producing both software and services in vertical software markets, nowadays mostly dominated by large proprietary firms. The findings show that FLOSS firms are able to compete with proprietary firms, though in the end a monopoly or oligopoly of the latters emerges. The ousted FLOSS firms, however, survive longer than proprietary ones, when these are not able to compete in the market.

Keywords: Free/Libre Open Source Software, simulation model, business model, market strategy.

1 Introduction

In recent years the software market has been in a constant state of change. From an economic point of view, the entry of open source products in the software market has led to changes in the market structure. The situation of monopoly or oligopoly, with only one software production model that characterized the past years, was replaced by a competition between two different business models: proprietary software firms (PROPSf), whose primary goal is to develop proprietary software to make profits and increase the value of their shares [11], and Free/Libre Open Source Software (also abbreviated as FLOSS[1]) –based firms (FLOSSf), whose primary goal is to develop non-proprietary software given away

[1] We refer to the phenomenon known such as "libre software", "free software", or "open source software".

for free. FLOSS was originally developed by volunteer programmers who engage in such projects because they are not satisfied with the existing software or simply because the required software feature does not exist [2]. Nowadays, many commercial firms decided to enter the FLOSS market, devolving resources to develop FLOSS products. Their profit is gained through selling complementary goods and services. For this reason, their behavior of developing a software product that is going to be given away for free is not so strange. An example of how some producers of proprietary software support the development of open source software is given by the IBM company, that invests in developing and adapting FLOSS to their hardware and software, and sells consulting services and proprietary software that are complementary to the FLOSS software.

In this paper, we propone a business model for the software market, and in particular we analyze the competition between proprietary software firms and FLOSS–based firms. This work follows and complements many other papers appeared on the subject. Among others, we quote the papers by Mustonen [9], [10], Bonaccorsi et al. [3], Bitzer et al. [1], Leppämäki et al. [7] and Economides et al. [4] .

At first we define and compare these PROPS and FLOSS business models, then we describe some aspects of their strategies.

The paper is organized as follows: Section 2 briefly presents some of the research literature related to our work, in Section 3 we explain the characteristics of our model and in Section 4 we show the analysis of data obtained from our simulations. Finally, Section 5 summarizes the conclusions of our research, and ends the study with recommendations for future work.

2 Related Works

In this paper we propose a business model that follows and complements many other papers appeared on the subject, but our work differs in many ways. Lihui Lin in [8], studies how users' skill and networks effect may influence the software market, characterized by PROPS firms, and FLOSS firms. The work of Mustonen [9] explains the simultaneous existence of commercial alternatives to copylefted programs and why commercial alternatives to copyleft programs may not exist. In this model the monopolist firm invests in the quality of its program, that depends on the programming output, and thus on the programmers' ability. The programmers can choose to develop for the monopolist firms or can be engaged in copyleft work, receiving complementary income based on their ability. Finally, the firm sets a price for its program, and the consumers value the various programs.

Two research papers gave specific and precious insights to develop our model – they are the works of Haruvy et al. [6] and of Gosh [5], that have suggested some variables of our proposed model.

In particular, the work of Haruvy et al. [6] examines a model in a monopoly setting where the open source code is free but complements another product that is sold commercially. The authors characterize price, quality, and hiring paths for the firms under both the open source and closed source models. The optimal decision on opening the source depends on the importance of user contributions, and on the wages and effectiveness of in-house developers. In the case of closed source development, all quality improvement is due to in-house developers. Instead, in the case of open source development, all quality improvements come from the users, as a function of the size m of the network of users. So, in the first case the quality is defined as:

$$\left(\frac{dQ}{dt}\right) = KN - \delta Q \tag{1}$$

while, in the second case the quality is defined as:

$$\left(\frac{dQ}{dt}\right) = \alpha m - \delta Q \tag{2}$$

The parameter K denotes the productivity or effectiveness of the in-house programmers. The software quality becomes obsolete over time at a constant rate $0 < \delta < 1$. The parameter α represents the level of involvement of the open source user community, which includes users of both the software and of the complementary product.

Gosh [5] considers the software such as one of the key elements driving ICT role in the economy. This work identifies the role of FLOSS in the economy, its direct impact on the ICT sector, and its indirect impact on ICT-related sector. It uses an endogenous growth model in order to model and simulate the economic impact of FLOSS. In Gosh' work the growth rates of the following variables are analyzed and studied: FLOSS and PROPS prices, quality, technology (called *varieties* in [5]), human capital, effective capital stock, ICT-capital stock, and finally output and labour productivity. For the sake of brevity, we report only the equations defining the human capital, and the definition of the FLOSS and PROPS varieties, since they were also used in our model. In the model [5], the human capital h is defined as:

$$\left(\frac{dh}{dt}\right) = \pi(uh(t))^{\gamma}(\nu k_i(t))^{1-\gamma} \tag{3}$$

where π is a constant parameter reflecting the productivity of the human capital accumulation process, γ is a constant parameter equals to $0 \leq \gamma \leq 1$, and k_i is the ICT capital. The FLOSS and PROPS varieties are proportional to the labour force measured in human capital units, to the ICT-capital intensity, and to an exogenous term linked to R&D activity.

3 Model

In this section we describe our agent–based business model. Recently, in the software market an alternative to proprietary companies has been introduced

– the open source competitors. Analyzing the software market, we make the hypothesis that market trends are determined by the interaction between two kind of heterogeneous agents: users and software firms.

The aim of our model is to study a "vertical" software market, that is a segment of the whole software market where some software applications compete, giving functionalities to perform a specific job. To keep the model as simple as possible, we do not consider families of applications, network effects, or other kind of externalities.

In the proposed model, we consider an economy with two kinds of software firms:

1. Proprietary Software firms (PROPSf): firms that develop proprietary software, trying to make profit;
2. FLOSS–based firms (FLOSSf): firms whose business is built upon the selling of services complementary to the free software product.

The users of our model are heterogeneous in their expertise and they can freely choose between proprietary and FLOSS software. In our software market model, the software products are substitutable, meaning that they meet the same need, thus they have the same functionality, but they may differ in quality (for instance: Microsoft Office vs Open Office). Each proprietary software firm is characterized by a specific product that differs from the products of the competitors in quality, technology and cost. The PROPS firms develop a primary product (the software), and a complementary secondary product (the services). Complementary products are the products whose purchase cannot be made without the other one (e.g.: software product and specific services). They complement each other in common usage, and buying one of them would encourage buying the other. The primary and secondary PROPS products are commercial.

On the contrary, FLOSS firms tend to associate with each other in order to produce a single product. The FLOSS firms develop a primary product (the software), and a complementary secondary product (the services). The primary product is distributed freely, the secondary product is a commercial product. In our model all firms, PROPSf and FLOSSf, are characterized by specific investment policies, which influence the prices, the quality and the technology of the products. Starting from the definition of the product quality presented in the model of Haruvy et al. [6], the quality $Q_i(t)$ is defined using a differential equation, where *human capital* h refers to the model carried out by the team led by UNU-MERIT [5].

We describe the features of the products of our model. First, we present the products developed by PROPS firms. The quality $Q_{p,s,i}(t)$ of the primary and secondary PROPS products of the $i-th$ firm at time t is defined as in [6]:

$$\left(\frac{dQ_{p,s,i}}{dt}\right) = h_i(t)N_{p,s,i} - \delta Q_{p,s,i} \tag{4}$$

The technology $T_{p,s,i}(t)$ of the primary and secondary PROPS products of $i-th$ firm at time t is defined as in [5]:

$$T_{p,s,i}(t) = \Psi_0^p[h(t) * N_{p,s,i} + N_{p,s,i}] + \Psi_1^p[(1-\nu)I_{ICT,i}(t) - h_{TOT,i}(t) - N_{p,s,i}] + \Psi_2^p \tag{5}$$

The features of the primary FLOSS product are defined instead by the following equations:

$$\left(\frac{dQ_{p,i}}{dt}\right) = \alpha(t)m(t) + \sum_{i=0}^{N_{FLOSSf}} \lambda_q h_i(t) N_{p,i} - \delta Q_{p,i} \tag{6}$$

$$T_{p,i}(t) = \sum_{i=0}^{i=N_{FLOSSf}} \lambda_t * \\ * [\Psi_0^f(h(t) * N_{p,i} + N_{p,i}) + \Psi_1^f[(1-\nu)I_{ICT,i}(t) - h_{TOT,i}(t) - N_{p,s,i}] + \Psi_2^f] \tag{7}$$

The features of the secondary FLOSS products are defined by the following equations, as in [6] and in [5] :

$$\left(\frac{dQ_{s,i}}{dt}\right) = h_i(t)N_{s,i} - \delta Q_{s,i} \tag{8}$$

$$T_{s,i}(t) = \Psi_0^f(h(t) * Ns, i + N_{s,i}) + \Psi_1^f[(1-\nu)I_{ICT,i}(t) - h_{TOT,i}(t) - N_{s,i}] + \Psi_2^f \tag{9}$$

where the meaning of the variables is the following:

- $N_{p,i}$ and $N_{s,i}$: Number of developers of the $i-th$ firm that work in the primary and secondary product, respectively;
- Ψ_0^f: Contribution of human capital in FLOSS based number of varieties;
- Ψ_1^f: Contribution of ICT-capital[2] in FLOSS based number of varieties;
- Ψ_0^p: Contribution of human capital in PROPS based number of varieties;
- Ψ_1^p: Contribution of ICT-capital in PROPS based number of varieties;
- h: Productivity or human capital;
- I_{ICT}: ICT-capital stock;
- α: Level of involvement by the open source user community;
- $m(t)$: Size of open source community at time t;
- δ: Rate of depreciation of quality;
- λ_q, λ_t: Parameters that limit the quality and technology of the primary FLOSS product, with respect to the sum of contributions to the project of all FLOSS firms.

[2] ICT- capital is the money specifically devoted to increase competences and assets in information and communication technology (See [5]).

The equations 6 and 7 differ from the equations presented in [6] and in [5] because we have take into account the contributions of all FLOSS firms to product development in order to define the primary FLOSS product quality and technology. These contributions are weighted by the coefficients, λ_q, and λ_t, that have values less than one.

3.1 Firms' Behavior

Each company enters the market with an initial investment I_i and with two software products, one primary and one secondary, respectively characterized by a well-defined quality $Q_{p,i}(t)$ and $Q_{s,i}(t)$ which varies over time. Each company is endowed with a initial wealth $W_i > 0$ used as the initial investment for a new product or service. In order to enter the market each company invests at the initial time t_0 a fraction β of its initial wealth: $I_i(t_0) = \beta(W_i)$ with $\beta \in [0.8, 1]$. Now let's suppose that $N_{p,i}$ is the the number of work units employed by the $i-th$ firm in the primary product , $N_{s,i}$ is the the number of work units employed by the firm i in the secondary product, and ω is the per-capita salary for company employee. It follows that:

$$I_{(ICT,i)}(t_0) = I_i(t_0) - \omega(N_p + N_p)$$

is the quantity of wealth to be invested in ICT at the initial time t_0, and

$$I_{(notICT,i)}(t_0) = \omega(N_p + N_p)$$

is the quantity of wealth to be invested in non- ICT expenses (wages) at the initial time t_0.

By considering our initial definition that N is the total number of company employees, let's suppose that in the FLOSS case $N_p = N_s = \frac{N}{2}$ and in the PROPS case $N_p = \frac{2}{3}N$ and $N_s = \frac{1}{3}N$.

Briefly, we show the investments in the next time step. Each software firm updates its primary product at time intervals Δ, whose values are normally distributed, investing a quantity of money $I_{ICT,i}(t)$ that depends on its profits. We assume a longer time interval in PROPS firms than in FLOSS ones. In particular, $I_{ICT,i}(t)$ is a normal variable that assumes values in the interval $[0, \mu * G]$. The variable μ is a normal variable that assumes values in the interval [0.2,1] for firms with less than 50 developers, and in the interval [0.2, 0.4] for firms with more than 50 developers.

At monthly intervals the software firms invest a small quantity of money $I_{ICTmin,i}(t)$ in order to keep acceptable the software quality, and a quantity $I_{(notICT,i)}(t)$ for ordinary expenses.

Let us note that the ICT investments of a firm are equally shared by the developers of the firm, so the human capital per capita is equal for every developer, and in according to the definition of quality, an improvement of primary product quality implies an improvement of the secondary product. These improvements of the products' quality are proportional to the total human capital of the firm, $h * N$, and every ICT investment contributes to increase the price of both the primary and the secondary product.

Finally, let's note that the firms can make a new investment only if the following condition is verified: $G(t-1) > -[I_{ICT}(t) + I_{ICTmin}(t) + I_{nonICT}(t)] + R(t)$, otherwise the software firm may apply for funding. The credit value is equal to the difference between the profit G_i and the wealth to invest at time t $I_i(t)$. A company goes bankrupt when its debt exceeds its wealth W_i. For each new request for funding, the amount of time to reimburse the financing and its interest rate are calculated. In the case of business failure, the firm's customers will need to purchase a new product.

3.2 Price Clearing Mechanism

We assume that the price clearing mechanism is influenced by the investments I_i made by the i_{th} software firm, by the number of software firms on the market N_{fTOT}, by the total number of users in the market N_{uTOT}, and finally by a percentage of the profit g_i to be obtained. The price of the primary and the secondary PROPS products at time t is calculated taking into account all the investments made by a firm from time $t = 0$ to time t, and it is determined by the following equations:

$$P_{p,pi}(t) = g_i(t) \left(\frac{N_{fTOT}(t) \sum_{t=0}^{t} I_{p,i}(t)}{N_{uTOT}} \right) \tag{10}$$

$$P_{s,pi}(t) = g_i(t) \left(\frac{N_{fTOT}(t) \sum_{t=0}^{t} I_{s,i}(t)}{N_{uTOT}} \right) \tag{11}$$

The primary FLOSS products is freely distributed, while the price for the secondary FLOSS products is given by:

$$P_{s,fi}(t) = g_i(t) \left(\frac{N_{fTOT}(t) \sum_{t=0}^{t} I_i(t)}{N_{uTOT}} \right) \tag{12}$$

Finally, let's note that the firms reduce the price of their products when the number of their customers is decreasing.

3.3 User' Behavior

At the start time, in the simulated market there are N_u users. This quantity may vary during the simulation, because at random intervals a random number of users, equal to $N_{u,i}$ and $N_{u,o}$, respectively, may enter or exit the simulation. Each user is modeled as an autonomous agent, and is endowed with a given amount of cash that varies in time. Our users are heterogeneous in their skill level, that increases with time.

According to [8], let's suppose that all users, with different levels of skill, compare the various available software products taking into account their quality, their technology and their price. We assume that the skill Θ is drawn from a normal distribution with average 2.5, limited to the interval [1,4]. When $\Theta = 1$, the skill is considered low, while for $\Theta = 4$ the skill is high. The user's choice is made in the following way:

– for $1 \leq \theta < 2$, the user chooses the cheapest product;
– for $3 \leq \theta \leq 4$ the user chooses the product with highest quality and technology;
– for $2 \leq \theta < 3$ the user chooses a product randomly.

Moreover, at each time interval, a set of users is extracted in a random way, and only these users will be involved in an economic transaction. If the user chooses a proprietary product, then his portfolio will decrease by a quantity equal to the sum of $P_{p,pi}(t)$ and $P_{s,pi}(t)$ (see eq. 10 and eq. 11). Instead, if the user chooses a FLOSS product, then his portfolio will decrease of a quantity equal to $P_{s,fi}(t)$ (see eq. 12). Finally, if the user wants to update a software product, then he incurs in an update cost that is zero only in the FLOSS case.

4 Results

In this section we describe the results of the computational experiments we performed. In particular we analyzed the influence of FLOSS firms in the software market in order to assess whether there are conditions under which the FLOSS firms are able to compete with PROPS firms, or even if they are able to exclude the others from the market.

The proposed model describes an extremely ideal software market, where at the start time, all companies enter the market with their own offers, the users make their purchasing decisions, and only in the subsequent steps the users and the firms operate according to more realistic and specific strategies. Our model is studied in a time period T, and the time step used by our simulator is nominally equal to 1, corresponding to one month calendar time. As initial conditions, we assumed that the quality and the technology of products on the market and the productivity of the firms, have values equal to one at the beginning of the simulation. We assigned to each firm a given amount of initial capital, depending on its number of developers. The initial capital available to each firm is equal to 50,000 per developer in the firms with more than 50 developers, and it is equal to 40,000 per developer in firms with less than 50 developers. The number of developers is assigned at the start time and remains unchanged during the simulation.

We divided the simulations in two sets; for each set we considered different initial parameters. In the first set, we considered an initial number of users equal to 2000, an input rate of new users variable in the range [4-10], and an output rate of users variable in the range [1-4]. In the second set, we considered an initial number of users equal to 3000, an input rate of new users variable in the range [40-100], and an output rate of users variable in the range [10-40] We simulated four scenarios for each set. In these scenarios, we set the number of developers and the initial capital value considering that FLOSS firms have typically smaller size and make smaller investments with respect to PROPS ones. In the first scenario we studied the trend of some economic variables by considering 20 PROPS companies operating in the software market and studying the competition only among proprietary firms. They are characterized by a number of developers that varies in the interval [20;200] and by an initial capital with values in the range [800,000-10,000,000].

Subsequently, we analyzed the market trends when one PROPS company competes with 5 FLOSS companies. The PROPS firm is characterized by an assigned capital equal to 10,000,000 and by a number of developers equals to 200. The number of FLOSS developers varies in the interval [10-70] and their initial capital takes values in the range [200,000-1,000,000]. As mentioned in the model presentation, FLOSS firms are positively affected by the contributions of the FLOSS community and they enjoy the benefits arising from the peculiarities of this software to offer free access to the code. Consequently, we analyzed the influence of FLOSS competitors in the PROPS market by introducing a term that takes into account the increasing productivity of FLOSS firms. This fact is modelled through the parameter α, that represents the level of involvement of the open source user community. Note that the parameter α is a normally distributed variable, that assumes values in the range [1-4]. This range was set in agreement with the fact that the human capital of the in-house developers grows in time up to a max value equal to 3.

In the third scenario, we studied the iterations among 10 PROPS firms and 5 FLOSS firms. Also in this scenario, the initial capital and the number of developers of the PROPS firms are larger than those of FLOSS firms. In particular, in the PROPS firms, the number of developers varies in the interval [20-100] and the initial capital is valued in the range [200,000-5,000,000]. The number of FLOSS developers varies in the interval [10-70] and the initial capital takes values in the range [200,000-1,000,000].

Eventually, in the fourth scenario we examined the same quantity of PROPS and FLOSS firms (14 firms in total) that compete among each other. For each PROPS firm, there is a FLOSS firm with the same features, in particular the number of developers varies in the interval [20-70] and the initial capital in the range [800,000-3,500,000].

Note that the firm survival does not only depend on its initial capital and number of developers, but on many other factors, closely linked. In fact, the profit of a firm depends by the following quantities: initial capital, number of developers, number of users, ICT capital invested to upgrade its products, ICT capital invested to keep acceptable the quality, time interval to update the products, financing and interest rate, parameters that weight the human capital, ICT capital and research and development.

For FLOSS firms, we add three more factors: level of involvement by the open source community, and parameters that limit the quality and technology of the primary FLOSS product with respect to the sum of contributions to the project of all FLOSS firms.

Table 1. Initial Parameters of the proposed model

Parameters
I_{ICT}: capital invested to update the product
I_{nonICT}: capital invested in wages
I_{ICTmin}: capital invested to keep an acceptable quality
β: normally distributed variable with values in the range [0.8,1]
μ: normally distributed variable with values in the interval [0.2,1] for firms with less than 50 developers, and in [0.2, 0.4] for firms with more than 50 developers
δ_{PROPS}, δ_{FLOSS}: time intervals to update the product, with values in the range $[24, 48]_{months}$ and $[12, 18]_{months}$, respectively
quality and technology: main features of the products defined by a differential equation and by an algebraic equation, respectively
m: size of the open source community, and α: normally distributed variable with values in the range [1,4]
N_u: number of users (public and private companies), θ: normally distributed variable with values in the interval [1,4]
λ_q and λ_t :normally distributed variables with values in the range [0.1,0.7]
Ψ_0^F: normally distributed variable with values in the interval [0.8,1]
Ψ_0^P: normally distributed variable with values in the range [0.9,1]
Ψ_1^F: normally distributed variable with values in the interval [0.2,0.3]
Ψ_1^P: normally distributed variable with values in the range [0.1,0.2]
$\Psi_2^{P,F}$: parameter equals to 0.25]

In Table 1 we report the parameters we used for performing the simulations, that have a close link with the firm's ability to survive in the market. The reported values are taken from the literature, by analyzing market data, and by using our experience in software engineering. They have to be considered a first attempt to build such a complex model, and to verify its consistency.

In the following subsection 4.1 we report some results obtained running the simulations described above, assuming for the model's parameters the values reported in Table 1. In subsection 4.2, we report a sensitivity analysis to validate the results obtained.

4.1 Simulation Sets

In a first set of simulations we considered an initial number of users equal to 2000, an input rate of new users variable in the range [4-10], and an output rate of users variable in the range [1-4]. In the second set, we considered a bigger, more variable market, with initial number of users equal to 3000, an input rate of new users variable in the range [40-100], and an output rate of users variable in the range [10-40].

In Table 2 we report the results of the simulations by highlighting the number of survived firms for each kind, and the 25th, 50th and 75th percentiles

Table 2. Number of survived firms and statistics about exit time from the market in the two sets of four performed simulations

	Scenario 1		**Scenario 2**		**Scenario 3**		**Scenario 4**	
Set 1	*PROPS*	*FLOSS*	*PROPS*	*FLOSS*	*PROPS*	*FLOSS*	*PROPS*	*FLOSS*
Survived firms	1	- -	1	0	1	0	2	0
$P_{0.25}$	16.25	- -	0	102	7	120.5	7.75	65
$P_{0.50}$	36	- -	0	108	8	132	8	68
$P_{0.75}$	140	- -	0	159.5	8	170.5	9	80.25
	Scenario 1		**Scenario 2**		**Scenario 3**		**Scenario 4**	
Set 2	*PROPS*	*FLOSS*	*PROPS*	*FLOSS*	*PROPS*	*FLOSS*	*PROPS*	*FLOSS*
Survived firms	1	- -	1	0	1	0	2	0
$P_{0.25}$	15.25	- -	0	142.5	7	127	7	90.5
$P_{0.50}$	34	- -	0	168	8	134	7	96
$P_{0.75}$	212.75	- -	0	180.5	8.25	159.5	8.25	107.75

pertaining to the time istants in which the firms that go bankrupt exit the market. The results of the first Set confirm that in a free market with no technological breakthroughs or large economical variations, a monopoly or an oligopoly of PROPS firms tend to emerge. In particular, in the case of only PROPS firms (Scenario 1), the results show that the 75 % of the firms exit the market before of the 140th month, with 50% of the firms exiting within 36 months. A single monopolist emerges among these firms. The analysis of the competition between PROPS firms and FLOSS firms, under the initial conditions and the parameters chosen, show that in the long term a PROPS monopoly tends to emerge, driving out from the market all FLOSS firms. In the Scenario 4, an oligopoly of two PROPS firms emerges.

Note that the ousted FLOSS firms tend to survive on the market fairly longer than ousted PROPS firms. The latter ones are out of the market in about 9-10 months, while the former ones resist ten times longer, or more. This feature is closely linked to their smaller size and their smaller investments, so that they tend to loose less money, and thus are able to stay in the market for a longer time.

With a bigger market, and a bigger average inflow of new users every month (Set 2) , the survival times of PROPS firms do not change substantially, except in Scenario 1, when only PROPS firms are involved. In this case, the time when the last firms exit the market is substantially increased. On the contrary, FLOSS firms tend to last longer in almost all cases. In the 4th scenario, we have two PROPS firms surviving in the market.

In Figure 1 we show the number of users of the firms in this last scenario. Most PROPS firms collapse very quickly, while FLOSS firms proceed together a longer time, until they go bankrupt in about 80-130 months. Two PROPS firms survive with about the same market share. The number of users steadily increase with time.

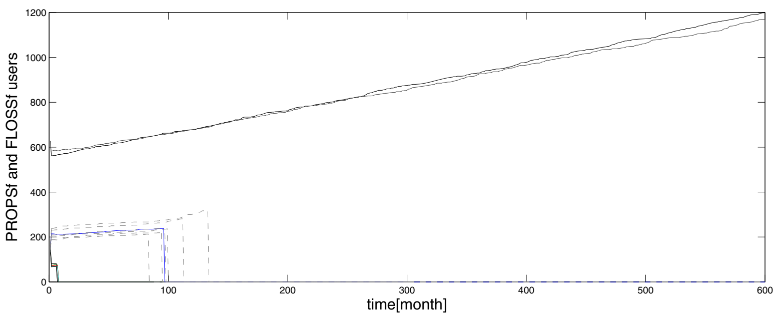


Fig. 1. Scenario 4: Numbers of firms' users versus time in the case of 7 PROPS firms (solid lines) and 7 FLOSS firms (dashed lines)

Table 3. Second simulation set: results of the Monte Carlo analysis

PROPS parameters	Scenario1	Scenario2	Scenario3	Scenario4
Average exit time for last ousted PROPS firm	336.34	-	14.21	12.52
$P_{0.05}$	273	-	9	8
$P_{0.95}$	384.5	-	27	22
FLOSS parameters	Scenario1	Scenario2	Scenario3	Scenario4
Average exit time for last ousted FLOSS firm	-	227.61	228.0833	132.55
$P_{0.05}$	-	159	165.7	107
$P_{0.95}$	-	308.5	292.8	172.50

4.2 Sensitivity Analysis: Monte Carlo Simulations

In the previous section, we reported results about just eight simulations. To assess the robustness of our model, we used a Monte Carlo approach. We repeated several simulations with the same initial conditions, but different seeds of the random number generator. We executed 100 simulations for the four scenarios of the second set, using the same parameters reported in Table 1. In particular, for each Monte Carlo run, we computed the exit time of the last firm exiting the market. In Table 3 we show the results of the Monte Carlo analysis and report the average exit time of the last ousted firm, and the 5th and the 95th percentiles of the exit time of the last ousted firm for all Monte Carlo runs, and for the two different kinds of firms.

In all the runs, we consistently observed that a monopoly, or an oligopoly of PROPS firms tends to emerge, but the ousted FLOSS firms survive on the market consistently longer than ousted PROPS ones. The 5th and 95th percentiles of the survival times of the last ousted firm are of the same order of magnitude of the average times, showing an overall consistency of the results.

5 Conclusions and Future Work

The presented simulations has the aim to present an heterogeneous agent-based approach to study the influence of the FLOSS firms in the software market by assessing whether there are conditions under which FLOSS firms are able to compete with PROPS ones, or even if they are able to exclude other firms from the market. We believe that the results of our agent-based model, whose parameters were taken from the literature, by analyzing market data, and by using our experience in software engineering, are interesting.

These results show that the FLOSS firms are able to compete with PROPS firms for a long time in the market, albeit the larger size and ability to make investments in technology in the end lead to a monopoly, or an oligopoly of the latters. Typically, FLOSS firms have a smaller size, and make smaller investments, compared to PROPS ones. This makes them more agile, and able to survive for a long time – of the order of ten years or more in our reference time.

Our model, as all simulation models, is a simplification of the real world. It represents each agent in an ideal way, and is based on several parameters in general difficult to estimate and whose values can heavily influence the output of the simulation. The aim of the presented work, however, is to show that an agent-based approach to analyze the software market is viable. To keep the model more focused, we limited ourselves to a "vertical" software market, with relatively few competitors and with no network or platform externalities.

Clearly, further research is needed to improve the model making it more realistic, and studying which parameters of the model have the larger effect on the results. In future works we will extend our model, in particular we intend to analyze the effects of network externalities, and to study product families, and their interplay. We will improve the price clearing mechanism, and we will include the possibility for the users to buy the primary and secondary products not necessarily from the same firm. Moreover, we will take into account that the customers' purchasing decision will vary in agreement with the maturity of the product and with the fact that FLOSS is often avoided because of the uncertainty of future support services. We will also take into account a more dynamic simulation setting, with the number of companies and customers following growth patterns.

We believe that this kind of research has the potential to help the study of future developments of various software market segments, including whether the monopoly of big firms will continue or will be challenged.

References

1. Bitzer, J., Schröder, P.J.H.: Competition and innovation in a technology setting software duopoly. Elsevier B.V, Amsterdam (2006)
2. Bitzer, J., Schröder, P.J.H. (eds.): The Economics of Open Source Software Development. Elsevier B.V, Amsterdam (2006)
3. Bonaccorsi, A., Rossi, C.: Why Open Source software can succeed. Research Policy 32, 1243–1258 (2003)

4. Economides, N., Katsamakas, E.: Two-Sided Competition of Proprietary vs. Open Source Technology Platforms and the Implications for the Software Industry. Management Science 52(7), 1057–1071 (2006)
5. Ghosh, R.A.: Study on the Economic impact of open source software on innovation and the competitiveness of the Information and Communication Technologies (ICT) sector in the EU, UNU-MERIT (2006)
6. Haruvy, E., Sethi, S.P., Zhou, J.: Open Source Development with a Commercial Complementary Product or Service. Production and Operations Management 17(1), 29–43 (2008)
7. Leppämäki, M., Mustonen, M.I.: Skill Signalling with Product Market Externality. The Economic Journal 119(539), 1130–1142 (2009), Available at SSRN: http://ssrn.com/abstract=1418177
8. Lin, L.: Impact of user skills and network effects on the competition between open source and proprietary software. Electronic Commerce Research and Applications 7(1), 68–81 (2008)
9. Mustonen, M.: Copyleft–the economics of Linux and other open source software. Information Economics and Policy 15(1), 99–121 (2003)
10. Mustonen, M.: When does a firm supportsubstitute Open SourceProgramming? Journal of Economics & Management Strategy 14(1), 121–139 (2005)
11. Pekka, H.: The Hacker Ethic and the Spirit of Information Age. Secker & Warburg, London (2001)

Adoption of Open Source Software and Software-as-a-Service Models in the Telecommunication Industry

Eetu Luoma[1], Nina Helander[2], and Lauri Frank[1]

[1] University of Jyväskylä, Department of Computer Science and Information Systems, P.O. Box 35 (Agora), FI-40014 University of Jyväskylä, Finland
{eetu.luoma,lauri.frank}@jyu.fi
[2] Tampere University of Technology, Dept. of Business Information Management and Logistics
P.O. Box 527, FI-33101 Tampere, Finland
nina.helander@tut.fi

Abstract. A case research is carried out on adoption of open source software (OSS) and software-as-a-service (SaaS) in the telecommunication industry. The study was conducted to examine the types of software deployed as OSS and SaaS and the conditions of adopting OSS and SaaS. Findings of the case study indicate that industry-specific software is not developed as OSS or deployed in SaaS mode. Based on the findings, we also arrive at conclusion: Adoption is hindered by specificity of processes and technology interfaces.

Keywords: Open source software, Software-as-a-Service, Telecommunication, Vertical Software Industries.

1 Introduction

Examining software business from the perspective of information systems (IS) science, software business is all about outsourcing the IS function. In the context of vertical software industry[1], software business takes place in dyadic relationship between a vertical industry enterprise and a vendor providing software products or services [1]. The vertical industry enterprise usually has its own unit or employees to produce certain parts of the IS function itself [2]. Alternatively, the enterprise may find it more efficient to outsource software development, deployment and operating to an external vendor. Nelson et al. [3] have provided an examination on which types of information systems are being outsourced. They found that common applications based on common technology are more likely to be acquired as packaged software, whereas specialized and unique applications require custom software development.

[1] Vertical software industry comprises of vertical industry enterprises (secondary software companies and software vendors (primary software companies), producing software products and services for the specific needs of the vertical industry. Vertical industry (e.g. telecommunication) has a clear specialization and limited transferability of skills and knowledge outside its own domain. Later, the term industry-specific software is used to describe software, which cannot be easily redeployed in other vertical industries than its original domain, as opposed to producing horizontal (general-purpose) software.

Further, unique applications based on common technology are more likely to be insourced and common application based on advanced technology are rather outsourced. The common development in the vertical software industries is that once unique and differentiating software depreciate into commodity [1, 4].

The software business setting in vertical software industries is also relevant when investigating the adoption of open source software and software-as-a-service. To facilitate such examination below, we define the concept based on contemporary literature as follows. *Open source software* (OSS) refers to product software, which is produced in collaborative manner and made available royalty free and with relaxed license terms. The terms allow to running, distributing and modifying the source code, for both commercial and non-commercial use. [5–8]. *Software-as-a-Service* (SaaS) is a type of software-based service where a service provided offers access to the functionalities of a commodity software over the Internet to several end-users, running a single instance of the particular software on top of multi-tenant infrastructure [9–11]. These definitions imply that OSS is a model for organizing software development and maintenance and, in turn, SaaS is a model for orchestrating software deployment and operating. Further, both OSS and SaaS can be treated as means to outsource software-related activities from vertical industry enterprise to external vendor.

The utilization of OSS has increased remarkably in the last decade. In the development of software programs and even large software systems, open source has become a serious alternative for the utilization of proprietary software [5, 6]. As one of the manifestations of cloud computing, even higher expectations are set to SaaS offerings. Primarily for established software vendors, SaaS presents an opportunity to add value in form of service offering, even as the product business declines [12]. Customers are also offered economic, flexibility and strategic benefits [11]. Against this background, it is interesting to examine why OSS and SaaS have not become widespread in industry-specific software. Conducted literature review reveals that this question has not been addressed before with regard to the two models.

Consequently, the following questions are of particular interest in this study: 1) Which types of software do the vertical industry enterprises deploy using OSS and SaaS models? 2) Do the vertical industry enterprises perceive value of OSS and SaaS differently? 3) Which factors drive and inhibit adoption of OSS and SaaS offering in vertical industry enterprises? These questions are set with the intention to generate an overview to the adoption of OSS and SaaS models and to compare these models to other business models in software business (e.g. with bespoke software) through their benefits and problems. We seek answers to the questions by adopting an exploratory approach and, therefore, the aim of our study was to arrive at a hypothesis on the factors affecting the OSS and SaaS adoption.

Our empirical investigation is conducted in the context of telecommunication software, where communication service providers (CSP) and software vendors serving them form the vertical software industry. Software specific to this industry supports CSP's day-to-day processes for service fulfillment, service assurance and billing as well as infrastructure development processes. Specifically, two types of software are considered to fall under this definition. First, operations support systems are software systems supporting telecommunication network management processes such as maintaining network inventory, provisioning services, configuring network components, collecting and mediating usage information and managing faults [13]. Second, business support systems are software systems supporting customer management processes including taking orders, providing customer service, processing bills and collecting payments [13]. This definition of telecommunication software excludes e.g. software used in mobile terminals, where OSS is used in different forms and to a different extent. Telecommunication software industry was chosen as the context for investigation for its plurality: operation support systems were assumed to incorporate specific knowledge that cannot be redeployed in other industries, whereas business support systems support processes common to many vertical industries.

The article has four further sections. Next section gives an overview on relevant literature concerning the OSS and SaaS adoption. Section three introduces the case study methodology applied. In section four, we present the findings made in the empirical study about the context of the particular vertical software industry, and about the current OSS and SaaS adoption in the industry. In the concluding section, we discuss the findings against the research questions and present two hypothesis for further studies.

2 Literature Review

2.1 Open Source Software Adoption

The research on OSS has progressed in the last decade. Most of the earlier literature focused on the motivations of individuals to contribute to the communities, open source project management issues and on general descriptions of the model [5]. More recent studies have additionally aimed at observing the adoption of OSS in organizations, through clarifying the benefits and problems of OSS adoption [5, 14–21] through examining which kind of OSS are deployed [5, 17, 18] and through studying the antecedents of adoption [5, 22, 23].

Ajila and Wu [14] suggest that the benefits of OSS are associated with reuse: customers receive added value from reduced time to market, reduced product development costs, improved process predictability and increased product quality. Their study also indicates that when organizations perform OSS component reuse in a systematic manner, organizations can attain economic benefits, increased productivity and increased quality. These observations are in line with many other similar research results, which often state lower costs, higher quality, adaptability and reduced dependency on vendor as the main benefits of OSS [14, 16, 18, 19]. The papers reporting on OSS adoption [21, 22] analogously ascertain that the decision making in organizations culminates on assessing potential cost benefits, on opportunities to exploit communities' resources and knowledge, and on functionalities and maturity of the software under consideration. Maturity is stressed by [20, 21] stating that both the customers and the vendors may hold their actions until dominant design emerges, and act only when peers are deploying OSS. To conclude, important determinants for adoption are, uniformly to general IS outsourcing, economics of the offering, capabilities and commodification.

There are fewer studies questioning the claimed benefits [5, 19, 22]. Ven et al. [19] raise several arguments, for instance OSS itself may be for free, but the switching costs as well as total costs are unclear. Performing customization and modifications to the source code may also turn out impractical in case the organization is missing required skills [5, 16]. Finally, whereas using OSS reduces vendor lock-in in software development, dependency on the vendor providing supporting services may increase [5, 19].

The advantages of OSS are examined in the literature mainly from the viewpoint of software intensive enterprise. To our knowledge, there are also studies targeting vertical industry enterprises [16, 22, 23] and a single study with focus solely on secondary software enterprises [15], in which the authors interviewed representatives of 13 companies from different vertical industries. They found that incentive to innovate and collaborate (by virtue of access to source code) reduced vendor lock-in and diminished costs were the most important business-related benefits. Technical benefits included various elements of software quality such as performance, security, flexibility and interoperability, and the respondents of the study indicated that these outweigh the drawbacks (lack of OSS expertise and poor documentation). As business-related problems, the study lists e.g. lack of ownership and support, which denotes that the vertical industry enterprises may find it difficult to find a service provider taking responsibility over support. While [5, 16, 22, 23] lists several other qualities in favor of and against OSS, and also some which do not usually appear in OSS literature, none of authors did explicitly compare motivations of secondary software enterprises to general findings across OSS literature. Overall, there appears to be a paucity of published empirical research on how motivations to deploy OSS differ in vertical industry enterprises.

Studies on industry-specific software developed in open source communities further seem non-existent. Current academic literature concerns horizontal infrastructure software [5, 17, 18, 23] almost exclusively and the industry-specific OSS may perhaps come forth in next wave of publications [8]. It seems that proper business models are missing in provisioning of OSS for vertical industries [16].

2.2 Software-as-a-Service Adoption

Contemporary academic literature is mostly limited to describing the architectural and technical properties of the SaaS offering and, in terms of adoption, suggesting advantages and downsides of the model. [11] and [24] were among the first to introduce claimed value propositions: less need for internal IT resources, and lower initial and total costs. These economic benefits are associated with the deployment and distribution model of SaaS, enabling service provider to achieve economies of scale [11, 25]. In addition to the economic value, customers may gain flexibility advantages such as prompt deployment, scalability, easily accessible updates and patches and, additionally, strategic benefits like increased bargaining power over vendors [11, 24–28]. Offering and using SaaS may also create problems compared to traditional means of deploying software systems. Using SaaS, customer is exposed to risks of losing control of business-critical data [24, 25, 29], thus not being able to access the service or experience inferior performance owing to the distribution of the service over the Internet [24, 27, 29]. Extensive integration and need for customizations may also reduce the attainable benefits compared to other business models in software business [30].

Similarly to OSS, well-known examples of SaaS are horizontal and employed in multiple industries [25, 29]. Excluding infrastructure software (which are rather part of platform-as-a-service offering), in services to business customers, SaaS model is mostly applied to email, customer relationship management, human resources management and financial management applications. In services to individual consumers, SaaS is applied to social media applications (e.g. Blogger, Facebook) and to storage and office applications (e.g. Dropbox, Google Apps).

Despite the enthusiasm towards SaaS model, relatively limited amount of research on the actual volume, reasons and experiences of adoption is available. However, few insightful studies can be found. Xin and Levina [31], reporting a research in progress, hypothesize that customization and need for client-specific functionalities, required service volume, internal IT capabilities among few other factors derived from IS outsourcing literature would be determinants for SaaS adoption. Benlian et al. [32] examined adoption of different types of applications using SaaS model. Applying transaction cost theory, resource-based view of firm, and theory of planned behavior, the authors found that in office applications attitude towards SaaS adoption and SaaS adoption can be explained by subjective norm, by low level of specificity and low level of adoption uncertainty. Correspondingly, when analyzing ERP systems, SaaS adoption is explained by higher adoption uncertainty, higher strategic value of application and higher application inimitability. This result can be interpreted in a way that standard applications and applications which are not supporting core processes of the enterprise may be offered and adopted using SaaS model.

3 Research Method

Using empirical data, this study examines the adoption of OSS and SaaS model in the telecommunication industry. Specifically, present study analyzes why adoption takes place, which factors drive and which inhibit adoption of OSS and SaaS. Moreover, we examine how adoption occurs in the companies of the specific industry, both on the client-side and on the vendor-side. This study applies exploratory approach [33] and case research [34] including a total of eight companies. Out of the total, six are communication service providers (CSP) and two are companies producing software products and services. The case study approach was chosen because of the lack of previous research and explanations on limited adoption of OSS and SaaS in industry-specific software. Therefore, motivation to conduct case research was to increase understanding on the specific context of vertical software industries [35]. Furthermore, case research has been argued to apply for initial identification of cause-effect relationships and forming hypothesis for further studies [34].

Telecommunication industry was selected as the target domain, since it clearly exhibits characteristics of a vertical industry that were thought also to affect the software business setting. First, software systems in this domain are required to interface with telecommunication networks. In addition, software systems are required to support processes specific to the industry. Furthermore, analyzing the properties of operations support systems and business support systems used in the domain, it was assumed that the domain would have both industry-specific and horizontal software systems, facilitating more insightful analysis on adoption of OSS and SaaS. The set of companies was selected to the case study through both purposive and convenience sampling [36]. For the former part, the sampling frame consisted of finding case companies of different sizes and breadth of operations and finding markets within telecommunication industry with different phases of maturity. Consequently, European and Chinese communication service providers were first selected as the target group of this study. Secondly, we also wanted to incorporate the software vendor's viewpoint into the study and, therefore, the inquiry was targeted to software vendors serving the communication service providers. These vendors are typically well aware of the customer needs and trends, and often push the adoption of new technologies and models.

The present study was executed in 2010, using two main sources of information: public documents and interviews. We initiated the study by gathering general background information on the case companies from their own and other public web pages. Case company details are summarized in Table 1, including company type, geographical area, and company size measured by revenue in the year 2009. With regards to the company type and revenue, it is noteworthy that they are defined based on the primary source of information, namely the respondents organizational unit.

The interviews were conducted as semi-structure interviews consisting of both fixed and open-ended questions. The questions covered operational environment, software acquisition strategies and adoption of OSS and SaaS in particular. Questions concerning operational environment attempted to prioritize between certain focus areas and capabilities: increasing customer base, network technologies and their development, operational efficiency and new services making possible new sources of revenue. It was hypothesized that business focus would affect software acquisitions strategies, i.e. whether software-related activities are insourced or outsourced, or whether CSP would prefer to acquire bespoke software or software product. These aspects were asked from CSPs through ratio of spending between internal versus outsourced development and bespoke versus product software, respectively. Additionally, the reasons for the selected strategy were asked. Further, both business focus and software acquisition strategy were seen as associated with OSS and SaaS adoption. Both the OSS mode of development and the SaaS mode of deployment assume outsourcing and relatively high level of commodification. Therefore, the more CSPs outsource there is function and utilize product software, the more OSS and SaaS should become a viable alternative. The questions on OSS and SaaS adoption simply comprised of open-ended questions on whether, how and why the models were adopted in the CSP software systems. All the respondents were asked essentially the same questions.

The interviews were mostly accomplished by the authors. The interviews were digitally recorded and transcribed. A Chinese scholar interviewed the service providers E and F. For these interviews, the questions were first translated into local language and Chinese scholar was instructed in performing the interviews. Later, responses were later translated to English. Due to confidentiality reasons, these interviews were not recorded, but the interviewer made notes on the questionnaire form.

As presented in the Table 1, the interviewees represented different positions in their organization. The main criterion for interviewed persons among the CSPs was that they were actively involved in their firm's decision-making regarding acquisition and deployment of software systems. In the software companies, we selected respondents who were frequently in contact with their customers and were consequently acquainted with their customers' needs, decision-making criteria and actions. Also, we interviewed those employees responsible of development of software products.

Table 1. Details of the case companies

	Company type	**Area**	**Revenue in 2009 (Euros)**	**Respondent**	**Mode**
Service provider A	Group of 28 regional operators	Europe	consolidated, 450 million	CEO	In-depth interview
Service provider B	Affiliate of global CSP	Europe	close to 1 billion	IT manager	Focused interview
Service provider C	National, incumbent CSP	Europe	over 12 billion	IT manager	Focused interview
Service provider D	Affiliate of global CSP	Europe	over 1,5 billion	Director, R&D	In-depth interview
Service provider E	Provincial branch of national CSP	China	estimated 450 million	IT manager	Focused interview
Service provider F	Provincial branch of national CSP	China	estimated 750 million	Business manager	Focused interview
Software vendor A	Global telecom software vendor	Europe	consolidated sales over 12 billion	R&D managers	Two focused interviews
Software vendor B	Global telecom software vendor	China	consolidated sales over 12 billion	Account and R&D manager	Two focused interviews
Software vendor C	Global system integrator	Europe	consolidated, close to 75 billion	Account manager	Focused interview

With the software vendor producing software especially for telecommunication, we had the possibility to carry out two interviews in both Europe and China. In China, these interviews complemented the answers by the CSPs and enabled verifying certain aspects regarding the operational environment. While most of the interviews were so called focused interviews, we also conducted two in-depth interviews with informants. By *focused interviews*, we refer to a single interview [34], which in the present study usually took approximately two hours. By *in-depth interview*, we refer to an interaction with the informant over longer period of time involving at least two interview sessions [34]. This enabled asking more detailed questions and confirming initial observations.

Data analysis followed the principles of qualitative research on parallel data reduction, data display and drawing conclusions [36]. First, the data was organized by identifying unique patterns in each case on the basis of interview themes and research questions. These themes were operating environment, software acquisition, and adoption of OSS and SaaS. Pattern matching [34, p. 136] enabled analyzing factors within the cases. Next cross-case synthesis technique was employed, enabling comparing the cases and aggregating the data [34, p. 156]. Overall, particular emphasis was on aspects explaining adoption of OSS and SaaS across the cases, on comparing customers and vendors viewpoints and on potential connections between the context (operating environment and software procurement) and adoption of OSS and SaaS.

4 Research Findings

In the following, the observations made in the empirical study are presented by categorizing them according to the interview themes. The operational environment and general alignments in developing, deploying and operating software should be treated as the context, where the contemporary models of software business may be examined.

Properties of the operational environment were realized through examination of communication service providers' business focus and required organizational capabilities. Surprisingly, there was much variety among European CSPs. Service providers A and B saw increasing customer base as most important focus area. In contrary, service providers C and D perceived operational efficiency and new service development as most critical. While this may be due to the positions of the companies in their market (market leaders and challengers), respondent in CSP C highlighted that the telecommunication market is already saturated and that developing new services is possible only through understanding customer needs. In China, the market is still growing and service providers focus on customer acquisition and improving quality of their network services.

Both the European and Chinese service providers suggested that the capability of being able to customize standard technologies to match the customer needs will be critical in the future. Overall, CSPs seem to be transforming from technology-orientated to customer-orientated companies. One of the interviewees from service provider C described this change:

"We started out as a true technicians' company. We had an advantage because we were the only operator so selling your services was easy. That changed with the competition from cable companies around two years ago. We said ok, the customer is the central of our world and technology is a way to attract the customer."

Software procurement activity in the service provider firms was investigated through outsourcing viewpoint. Questions on this topic focused on reasons to outsource and spending on software related activities. Currently, majority of software development and deployment is outsourced. Chinese CSPs estimated the ratio of expenditures between internal work and outsourcing expenses to be around one to nine. In European CSPs, the ratio varies. For example, CSP B told that these activities are solely in the hands of the vendors, whereas firms C and D estimated the outsourcing ratio to be between 60 to 70 percent.

The European interviewees stated cost-efficiency to be the most important reason for outsourcing. In China, outsourcing may additionally be explained by a lack of high-end capability. One informant from software vendor B explained this as follows:

"Chinese operators do not have capabilities to develop software themselves. CSPs and ISVs co-operate in developing and deploying their operations support systems and business support systems, making it almost all tailor-made... Operating is mostly organized by the CSP."

We also asked the ratio of spending between bespoke systems and software products. Chinese CSPs reported that their software systems, specifically used in producing telecommunication services, are fully bespoke. European CSPs (B, C, D) in turn attempt to employ software products as much as possible. However, the reality with all the CSPs is that company-specific legacy systems cannot be replaced. Reasons for this include complex network interfaces, company-specific procedures and sunk costs. The situation is different between business support systems (for customer management and billing) and in operation support systems (for provisioning, ticketing and mediation). Replacing business support systems with standard solutions is more straightforward; standards for processes of customer management and billing exist and deploying standard software products have become possible.

4.1 Open Source Software Adoption

In the telecommunication industry, OSS is mainly deployed in infrastructure software. Mentioned software included Linux, Apache and MySQL. The software vendor C informed that there are many initiatives, which drive open source adoption and CSPs are increasing use of OSS components in the future. However, it was found that open source is not in use in industry-specific software. The software vendor C expressed his opinion that OSS "does not fit" to software specific to the telecommunication industry and there are no communities to develop them. With regards to infrastructure software, open source is nowadays a common practice and OSS is used as part of the software system deliveries. Respondent from software vendor B described the use of OSS as follows:

"Operators are using open source, mainly in applications provided for customers. There is no preference in using either open source or proprietary solution, rather they want functioning and secure (and cheap) solution. OSS is more common in infrastructure software than in application software."

In contrary, one of the informants (in CSP D) underlined that the use of open source is avoided in business-critical systems and in services visible to their customers. He suggested that open source can be applied to systems supporting internal processes and to "enterprise-grade" systems, but in "carrier-grade" systems proprietary solutions are preferred.

OSS is mainly adopted because of the cost factors (CSPs B, C, D, F), although CSP D commented that OSS is not cheaper by an order of magnitude when looking at overall costs. Use of OSS is also motivated by the capabilities and resources available through the communities. With CSPs A and C, this is related to the lack of internal capabilities and to the efforts to generate new sources of revenue. Many service providers believe that flexibility is also an important benefit for OSS, including fast time-to-market. Service provider B sees flexibility in form of future capabilities allowing customization of standard building blocks.

Barriers for adopting OSS in telecommunication industry include lack of internal capabilities (CSPs B, C, D), fear of liabilities (CSPs A, C, Vendor A), associated control risks and uncertainties in business continuation (CSP A, B, C, D). According to the respondents, lack of internal expertise leads to situations where obtaining commercial supporting service becomes necessary and as a result cost advantages are diminished. Uncertainties and fear of liabilities are linked to the complexity of different open source licenses. In addition, service providers A and D mentioned that they are not using open source, as no viable offering is available.

4.2 Software-as-a-Service Adoption

Similarly to OSS, SaaS adoption is connected to the cost benefits (CSPs B, F) and principally to the flexibility of SaaS offering. The service providers A, B, C and D presented ease of procurement, ease of maintenance and swift time-to-market as components of flexibility. However, respondents were concerned with the total costs and for instance CSP C disclosed such uncertainty as inhibiting factor to SaaS adoption. The Chinese service providers are not applying SaaS, because their suppliers are not providing it. This was explained by the software vendors A and B; in software vendors currently have a strong customer lock-in (no incentives to offer SaaS) and systems are acquired as custom deployments (transformation to SaaS would be difficult). Service provider B called for industry standards in speeding up the development.

Common concerns related to SaaS mode included integration and security issues (CSPs B,C,D,E). For instance, CSPs are obliged by law to apply high data security measures on call data records, which SaaS vendors are not able to comply with. Problems with integration are related to the properties of SaaS offering. The mode of deployment assumes standard processes and interfaces, which does not match the attributes of industry-specific software. Informant in CSP C described the issues related to company-specific processes and network technologies:

"It is a combination of the two things. We've got a variety of network technologies in our network, for historical reasons. And that doesn't help in making it easier to outsource it because both of them are completely different. So try to outsource that to one and same company in itself it's a challenge. Try to rationalize and simplify the processes around it is also a challenge... Yeah, I would be inclined to say that it is more challenging to outsource in the OSS side of fence than BSS of fence."

However, the SaaS mode of deployment has already been adopted in several companies in the telecommunication industry (CSP A, B, C, D, F). Deployed software are horizontal, e.g. for financial management and customer relationship management. SaaS is also in use in the business support systems. However, in the companies interviewed, SaaS mode of deployment is not applied for industry-specific software. Service provider D described the adoption of SaaS in their organization:

"SaaS deployments have progressed and CRM system is in production in one business unit. New projects to deploy SaaS have been initiated in the area of business support systems... Attitude towards SaaS is more and more positive."

With regards to third-party software, the most of the operators (CSP B, C, D, F) see their role in the value chain as reseller and operator of the services, and have already taken such role. The CSP A's strategy in providing third-party software is to increase customer lock-in, by providing a combination of IT and communication services, and envisions operating in both intermediating and aggregating roles, and has already launched product concept to do so. The service provider D is aiming for an aggregator role, where CSP offers multiple SaaS products for end-users. Such role is seen natural, and CSPs are expected to take such role in its ecosystem.

5 Conclusions and Further Research

This study has focused on different aspects of OSS and SaaS adoption in the context of vertical software industry. This is a perspective, which has received limited attention in the contemporary literature, although a significant share of software business takes place in this context. Examination of the facets of the vertical industries may to bring into focus certain factors explaining the adoption or non-adoption, which do not manifest in the procurement of more generic software. In this study, the dynamics of software business in the telecommunication industry were examined. It was regarded as suitable target domain for analysis as it demonstrates characteristics of vertical industry enterprises that are both generic (like selling and using CRM) and industry-specific (like provisioning mobile subscriptions and managing network elements). Conducting a multicase study involving both communication service providers and software vendors serving them therefore facilitated insightful examination on software business in vertical software industry.

The interview data uncovers that OSS mode of development and SaaS mode of deployment are currently utilized by the communication service providers in horizontal software: OSS in infrastructure software and SaaS in customer relationship management and financial management software systems, which can all be used similarly in many vertical industries. Industry-specific software (i.e. operation support systems) is not developed as open source or deployed as a service. This observation addressing the first research question has two further consequences.

First, it signifies that the perceptions and experiences of interviewees on OSS and SaaS can only be associated with horizontal software. However, this allows us to position the empirical findings more easily against the prior literature. The respondents mentioned mostly similar benefits and disadvantages of OSS and SaaS as in earlier studies:

- The benefits of OSS include cost efficiency [15, 21, 22], resources and knowledge of the communities [21, 22], reduced time-to-market [14] and adaptability [14] of source code. Lack of internal capabilities to maintain OSS [5, 15, 16] and resulting increased dependency on support services [5, 15, 19] were considered as problems of OSS.
- SaaS model was regarded as beneficial in terms of flexibility [11, 24–28] in procuring, deploying and maintaining the software. Cost benefits [11, 24] were also mentioned, but the respondents also raised a question whether the total costs of utilizing SaaS would actually be lower over longer period of time compared to other deployment models. The problems with the model to be solved include issues related to security [24, 27, 29] and integration [30].

In this research, the value of OSS and SaaS was examined primarily through advantages and disadvantages of the models compared to more traditional business models incorporating bespoke software or software products. Taking into account that the specific attributes of industry-specific software most probably did not affect respondents assessment of OSS and SaaS, a partial answer to the second research question may be given: conducted case research indicates that the communication service providers see the value of OSS and SaaS consistently with companies in other domains.

The adoption of OSS and SaaS in only certain types of software, and non-adoption in certain others, moreover indicates that there are factors in the operating environment and in the software business setting, which simultaneously drive and inhibit adoption of OSS and SaaS models. As revealed by the case research, the decision-making on software procurement in communication service provider firms is presently business-driven. There are concurrent pressures to reduce expenditures on software and to deliver compelling services of highest quality. Such pressures drive e.g. acting as sales channel for third-party SaaS offering. This also informs us that certain technology, specifically horizontal business support systems, does not to any further extent provide significant competitive advantage to the firms. Instead, focus is on new technologies and services that further makes commoditized software subject to outsourcing and cost considerations. This observation is in line with previous studies, in which productization [1] and commodification [4] are seen as leading to increase in adoption of OSS and SaaS models.

On the other hand, it can be stated that SaaS mode of deployment is not harnessed in industry-specific software, i.e. operations support systems. This observation is somewhat contradicting to the models describing commodification development, since operation support systems (for provisioning, ticketing and mediation) are unlikely to act as source of differentiation in telecommunication either. Some of the case companies addressed the issue. Representatives of software vendors disclosed that there may not be incentives to offer SaaS or developing SaaS offering may turn out infeasible. Reasons mentioned by the CSPs for using the existing systems, instead of opting for SaaS mode, included specificity of processes and technology interfaces. These factors also appear in previous studies as determinants for vertical software industry evolution [1], but in the present study, company-specific processes and interfaces emerged as factors disallowing use of highly commoditized SaaS offering.

When software business is examined as outsourcing of the IS function, transaction cost economics may be employed to explain market failure, i.e. non-adoption of SaaS mode of deployment. Transaction cost theory [37] holds that transactions with high asset specificity are managed more efficiently within the boundaries of the firm. In the software business setting, this means that the more specific the requirements of software are, the more likely shall the clients choose to develop the software internally or as bespoke software. Further, in case of high asset specificity, software is less likely to be acquired as software product or as a service. In a prior study, Benlian et al. [32] analyzed the association of asset specificity as explaining factor for SaaS adoption. However, their focus was on more generic software systems and their operationalization of asset specificity constructs was therefore missing dimensions, which might be relevant to vertical software industries. Based on the case research, and in line with the transaction cost theory, the following hypotheses are put forth for further studies:

H1: Specificity of processes in client organization is negatively associated with SaaS adoption.

H2: Specificity of technology interfaces in client organization is negatively associated with SaaS adoption.

This paper has examined the adoption of OSS and SaaS models in telecommunication industry. Therefore, it contributes to the software business literature by recognizing the similarities and differences in adoption in vertical software industries. Conducting a case research, it was found that managers in communication service providers find similar benefits and problems in OSS and SaaS as suggested by the current literature. A conclusion can also be made on the types of deployed software: communication service providers use OSS and SaaS mode of deployment in software provided and used across industries. In this case research, no examples of industry-specific software developed as OSS or deployed as a service could be found. For theory development in the field of software business, the findings indicate different patterns of adoption on different types of systems. This study arrived at two hypotheses, which are subject to further research.

References

1. Tyrväinen, P., Warsta, J., Seppänen, V.: Evolution of Secondary Software Businesses: Understanding Industry Dynamics. In: León, G., Bernardos, A., Casar, J., Kautz, K., DeGross (eds.) IFIP International Federation for Information Processing, Open IT-Based Innovation: Moving Towards Cooperative IT Transfer and Knowledge Diffusion, pp. 381–401. Springer, Heidelberg (2008)
2. Hirschheim, R.A., Lacity, M.C.: The Myths and Realities of Information Technology Insourcing. Communications of the ACM 43(2), 99–107 (2000)
3. Nelson, P., Richmond, W., Seidmann, A.: Two dimensions of software acquisition. Communications of the ACM 39(7), 29–35 (1996)
4. van der Linden, F., Lundell, B., Marttiin, P.: Commodification of Industrial Software: A Case for Open Source. IEEE Software 26(4), 77–83 (2009)

5. Nagy, D., Yassin, A.M., Bhattacherjee, A.: Organizational adoption of open source software: barriers and remedies. Communications of the ACM 53(3), 148–151 (2010)
6. Sen, R.: A Strategic Analysis of Competition Between Open Source and Proprietary Software. Journal of Management Information Systems 24, 233–257 (2007)
7. Von Krogh, G., Von Hippel, E.: The Promise of Research on Open Source Software. Management Science 52(7), 975–983 (2006)
8. Fitzgerald, B.: The Transformation of Open Source Software. MIS Quarterly 30(3), 587–598 (2006)
9. Vaquero, L.M., Rodero-Merino, L., Caceres, J., Lindner, M.: A break in the clouds: towards a cloud definition. ACM SIGCOMM Computer Communication Review 39(1), 50–55 (2008)
10. Mell, P., Grance, T.: Draft NIST Working Definition of Cloud Computing, Version 15, 8-21-09. National Institute of Standards and Technology. Information Technology Laboratory (2009)
11. Jacobs, D.: Enterprise Software As Service. Online Services are Changing the Nature of Software. ACM Queue, 36–42 (July/August 2005)
12. Cusumano, M.A.: The Changing Software Business: Moving from Products to Services. IEEE Computer 41(1), 20–27 (2008)
13. Luoma, E., Frank, L., Pulkkinen, M.: Overview of Telecom Operator Software Market. In: Tyrväinen, p., Mazhelis, O. (eds.) Vertical Software Industry Evolution: Analysis of Telecom Operator Software, pp. 35–42. Springer, Heidelberg (2009)
14. Ajila, S., Wu, D.: Empirical study of the effects of open source adoption on software development economics. Journal of Systems and Software 80, 1517–1529 (2007)
15. Morgan, L., Finnegan, P.: Open Innovation in Secondary Software Firms: An Exploration of Managers' Perceptions of Open Source Software. DATABASE for Advances in Information Systems 41(1), 76–95 (2010)
16. Ågerfalk, P.J., Deverell, A., Fitzgerald, B., Morgan, L.: Assessing the Role of Open Source Software in the European Secondary Software Sector: A Voice from Industry. In: First International Conference on Open Source Systems. Springer, Heidelberg (2005)
17. Hayes, J.: Open to Growth. Engineering & Technology 4(15), 54–55 (2009)
18. Ven, K., Mannaert, H.: Challenges and strategies in the use of Open Source Software by Independent Software Vendors. Information and Software Technology 50, 991–1002 (2008)
19. Ven, K., Verelst, J., Mannaert, H.: Should You Adopt Open Source Software? IEEE Software 25(3), 54–59 (2008)
20. Serrano, N., Sarriegi, J.M.: Open Source Software ERPs: A New Alternative for an Old Need. IEEE Software 23(3), 94–97 (2006)
21. Ebert, C.: Open Source Software in Industry. IEEE Software 25(3), 52–53 (2008)
22. Glynn, E., Fitzgerald, B., Exton, C.: Commercial Adoption of Open Source Software: An Empirical Study. In: International Symposium on Empirical Software Engineering. IEEE, Los Alamitos (2005)
23. Dedrick, J., West, J.: An Exploratory Study into Open Source Platform Adoption. In: 37th Hawaii International Conference on System Sciences. IEEE, Los Alamitos (2004)
24. Greschler, D., Mangan, T.: Networking lessons in delivering, Software as a Service' Part I. International Journal of Network Management 12(5), 317–321 (2002)
25. Gonçalves, V., Ballon, P.: An Exploratory Analysis of Software as a Service and Platform as a Service Models for Mobile Operators. In: 13th International Conference on Intelligence in Next Generation Networks. IEEE, Los Alamitos (2009)

26. Choudhary, V.: Software as a Service: Implications for Investment in Software Development. In: 40th Hawaii International Conference of System Sciences. IEEE Computer Society Press, Los Alamitos (2007)
27. Erdogmus, H.: Cloud Computing: Does Nirvana Hide behind the Nebula? IEEE Software 26(2), 4–6 (2009)
28. Gold, N., Mohan, A., Knight, C., Munro, M.: Understanding Service Oriented Software. IEEE Software 21(2), 71–77 (2004)
29. Luoma, E., Mazhelis, O., Paakkolanvaara, P.: Software-as-a-Service in the Telecommunication Industry: Problems and Opportunities. In: Tyrväinen, P., Jansen, S., Cusumano, M.A. (eds.) ICSOB 2010. Lecture Notes in Business Information Processing, pp. 138–150. Springer, Heidelberg (2010)
30. Ma, D.: The Business Model of "Software-As-A-Service". In: IEEE International Conference on Services Computing, SCC 2007 (2007)
31. Xin, M., Levina, N.: Software-as-a-Service Model: Elaboration Client-Side Adoption Factors. In: 29th International Conference on Information Systems (2008)
32. Benlian, A., Hess, T., Buxmann, P.: Drivers of SaaS-Adoption – An Empirical Study of Different Application Types. Business & Information Systems Engineering 1(5), 357–369 (2009)
33. Myers, M.D.: Qualitative research in information systems. MIS Quarterly 21(2), 241–242 (1997)
34. Yin, R.K.: Case Study Research: Design and Methods, 4th edn. Sage Publications, Thousand Oaks (2009)
35. Benbasat, I., Goldstein, D.K., Mead, M.: The Case Research Strategy in Studies of Information Systems. MIS Quarterly 11(3), 369–386 (1987)
36. Miles, M.B., Huberman, M.: Qualitative Data Analysis: An Expanded Sourcebook. Sage Publications, Thousand Oaks (1994)
37. Williamson, O.E.: The Economics of Organization: the Transaction Cost Approach. American Journal of Sociology 87(3), 548–575 (1981)

Examining the Effects of Agile Methods and Process Maturity on Software Product Development Performance

Mikko Rönkkö, Juhana Peltonen, and Christian Frühwirth

Software Business Lab, Aalto University
FI-00076 Aalto, Finland
{Mikko.Ronkko,Juhana.Peltonen,Christian.Fruehwirth}@Aalto.fi

Abstract. This paper examines the effects of agile methods and software process maturity on software product development performance. Through a mail survey, we obtained data from 72 small and medium-sized software firms that predominantly were not CMMI-certified. Findings from our partial least squares analysis suggest that the use of agile methods has a positive impact on product development efficiency and effectiveness, but CMMI practices do not have this effect. Our results suggest that software process improvement initiatives in software product firms create the highest benefits through first adopting agile methods and only then moving on to implementing CMMI-like process improvement initiatives.

Keywords: Product development, agile methods, process improvement, software.

1 Introduction

The performance of agile information systems development methods has received considerable interest among information systems scholars [e.g., 1-3]. However, the current level of empirical support for the arguments about performance of agile methods can be considered to be inadequate [4]. In this paper, we compare the effects of agile software development practices and process maturity on the firm software product development performance.

Research has provided us with strong evidence of the positive effect of process maturity on software development performance [5, 6]. These studies have predominantly focused on process maturity from a more traditional CMMI perspective. However, recently the use of agile methods has gained popularity in the software industry. These methods have been advocated as a solution to problems in software development [7-10] in a dynamic environment such as the software product industry [11, 12]. Regardless of the relative popularity of this alternative development approach, only a limited amount of research compares agile methods with other development approaches [13]. Clearly, more studies comparing the effectiveness of different development approaches are needed. Moreover, a large share of the studies testing the performance of CMMI uses data from CMMI appraisals that are used predominantly by larger firms [14]. While the CMMI based approach on software process improvement is well suited for large project-based development organizations, the effectiveness of different development approaches in small firm context is currently under-studied.

In this paper we investigate the following research question: "What is the impact of process maturity and use of agile practices on software product development performance?" This paper addresses two important gaps in the present body of knowledge: First, we conduct a statistical comparative test between the effects of process maturity and agility on software product development performance. To our knowledge, only a few comparative studies like this exist [e.g., 13]. Second, in contrast to previous studies using projects in larger firms, our sample consist of small and medium sized firms that develop software products hence extending the generalizability of previous studies.

The rest of the paper is organized as follows. In the second section we briefly review the key literature and formulate hypotheses for testing. The next sections introduce our sample, constructs and measures that are followed by data collection and analysis. Thereafter, the sixth section introduces the results of hypothesis testing using structural equation modeling. The final section presents a discussion on our results and on their broader implications.

2 Literature Review and Hypothesis Development

Compatibility of agile development and process maturity has been of interest for the information systems community. Agile methods were initially seen as alternative, contradicting ways to develop software that are based on profoundly different values and principles [15]. While agile development and process maturity were initially often considered to be incompatible targets, information systems and software engineering communities have recently started to consider these as more compatible than conflicting targets [15-18].

Process maturity, often measured through frameworks such as CMMI, refers to the degree of use of well-established software engineering methods [19]. Development organizations often strive for maturity to improve the quality of the systems developed, to improve the manageability of the process, and to make the development projects more predictable. During the recent more than a decade, the CMM family – including the most recent CMMI model – has established itself as the leading software process improvement framework, through which increased maturity is pursued.

When combined with traditional plan-driven development methods, increasing process maturity generally emphasizes and accommodates issues such as planning of the system and development organization, repeatability predictability, and relatively rigid control procedures [20, 21]. Combining software engineering best practices into development frameworks has been a big success in both terms of popularity and results. Indeed, there is no lack of research supporting the existence of a link between maturity and development process performance [5, 6, 22, 20]. However, software process improvement with CMMI or any other leading framework does not come without a cost. First, CMMI adoption and appraisals required for official certification are costly and require significant effort thus making it more accessible for larger than smaller firms [14]. Second, opponents of these development frameworks argue that they can lead to too much overhead making the process less efficient and responsive to changes [23] While these potential weaknesses can be justified especially in large software development projects, the relative cost in terms of lost opportunities can be high in more market driven software product development [24, 25].

Agile methods were initially presented as an alternative to the increased overhead in the development process as well as to decrease the effort required for adopting a new development framework. In contrast to several of the major SPI frameworks that are commonly documented as books [26], agile methods commonly define only minimum amount of processes required for a development team to function efficiently and can be codified as a relative small number of principles [27]. Another distinctive feature of agile methods is that they emphasize simplicity, speed, and flexibility to accommodate changes in requirements. For achieving the pursued characteristics, developers of the agile methods feel the planning – including documentation – and control should be, to certain degree, discarded [28]. Agile methods emphasize self-organization of the development team, quick and frequent delivery of working software in an incremental manner, and intensive communication both within the development team and with the customer. Not surprisingly, agile methods have been recently advocated as a solution to problems in software product development [7-10] as a response to dynamic environment where technologies change fast and often in an unpredictable manner [11, 12].

The topic of the compatibility of agile methods and process maturity has been recently under investigation in the information systems community [2]. Within the field of software engineering, insights have been gained through case studies of companies that have adopted agile methods into traditional development organization [16], and of companies using agile development processes that have implemented CMMI, even up to maturity level 5 [29]. In general, these efforts are usually not without challenges, but most studies report positive outcomes. In general, recently more evidence for than against the compatibility of agile development and process discipline has been presented [15-18]. The current understanding is that agile process can be disciplined and mature, and that the development process should be selected base on situation characteristics [15, 30].

Since most studies focusing on the performance of different software development approaches focus on software projects, this poses a limitation on how well these approaches generalize to non-project oriented settings. Particularly, cases where software is developed as products has received limited attention in the studies investigating the performance of different development approaches. We start the discussion on the product development effects of these frameworks by looking at how product development performance is defined in the literature. While information systems and software engineering research communities view the development performance as a two dimensional construct composed of product quality and development project performance [31], product development research has developed several alternative conceptualizations, most notably one consisting of efficiency, effectiveness and innovativeness [32], and an alternative separating product development and product management as distinct dimensions [33]. The differences on these conceptualizations can be best understood by examining them in the context: Incremental product development - the improvement of existing products or product lines - and radical product development - the development of completely new products or product lines – require different capabilities and hence it is natural that performance of these processes are considered as separate constructs [34, 35]. In all, the operationalization of incremental product development include the dimensions of efficiency and effectiveness, while the goodness of radical product development performance is often measured through an additional dimension of innovativeness. Efficiency refers to the goodness of rate that the resources are transformed into outputs [32], and is composed of two main factors, lead-time and cost-efficiency. This dimension has a close fit to the process performance dimension of software development performance [20, 36]. Effectiveness refers to the goodness of the product not only in terms of product quality – a common measurement of software development performance [20, 36] – but also how well it fits the needs of the markets. Finally, innovativeness refers to the ability to conceive new ideas and develop these into commercially successful products or product features.

Traditional plan-driven methods have been designed to for developing well-defined software solutions in time and on budget. These objectives are normally achieved through solid planning and by avoiding wasted programming work [19]. Considering the strong evidence of process maturity on development project performance [5, 6, 20] and the close fit between the performance dimensions of incremental product development and software project performance presented earlier, we hypothesize:

Hypothesis 1: Process maturity increases product development efficiency.
Hypothesis 2: Process maturity increases product development effectiveness.

While agile methods are not about efficiency in the same sense as plan-driven methods [7], they aim to reduce the amount of administrative work and control for wasted programming work through frequent integration, testing and delivery of program versions. In this way, agile methods provide an improvement in efficiency compared to having less defined processes. Considering the suggested fit between agile methods to software product development [7-10] we hypothesize:

Hypothesis 3: Process agility increases product development efficiency.

Product development effectiveness refers to how well the developed product matches the markets. While software product industry is obviously one of the more turbulent industries, it is likely that the market requirements change during the development cycle of a new product. In traditional project and contracting oriented software engineering requirements changes cause schedule and budget overruns. However, in product oriented development these can result in product with bad market fit or delay, significantly affecting how the product performs in the markets [37-39]. Hence it is imperative for firms to maintain flexibility in product development. We hypothesize:

Hypothesis 4: Process agility increases product development effectiveness.

Regarding the last dimension of product development performance, innovativeness is has received relatively little attention in the software process research. In management research, one of the most enduring findings about innovativeness is that innovation occurs at organizational interfaces [40]. Considering that several of the SPI processes are organizational level tools that define interfaces also between engineering and other units within the firm, process maturity should improve innovativeness. In addition, several of the frameworks specify that organization should have processes for managing innovations [7].

Hypothesis 5: Process maturity increases product development innovativeness.

It has been suggested that agile methods are associated with innovation [41-43]. Moreover, agile methods give significant responsibilities to individuals, and this empowerment of individuals is a prerequisite for innovative behavior [28]. Moreover, agile methods emphasize the importance of communications with the customer interface and emphasize building the work motivation at an individual level. Hence we hypothesize:

Hypothesis 6: Process agility increases product development innovativeness.

3 Empirical Study Design

We use a subset of a larger data collection that was collected as a part of a research project surveying software companies in Finland [44]. We use the term “software product firm” to refer to any firm that owns and markets a software product – regardless of the official company classification. The sampling frame for the primary survey, which is described in details elsewhere [44], contained 2616 firms mainly under the NACE (rev 1.2) codes 72.21 and 72.22, but also a small number of other firms that conduct software product business as a secondary area. For this sample, the CEO or other high ranking employee of a firm was used as an informant. After filling in the general information including questions about product development performance the informant nominated another person that was intimately familiar with the software development process of the firm, most commonly the manager of the software development function. These nominates acted as the population for the secondary survey where the software development related questions were asked. Asking the independent and dependent variables from a different person helped us to avoid common method bias, which is commonly present in survey research.

The dependent variable, product development performance, was measured with a scale developed by Kusunoki et al. [32]. It contains three dimensions; product development efficiency, product development effectiveness and innovativeness. These were measured with 10 items using 7-point Likert scales. Principal factor analysis revealed only one dominant factor and a weaker three-factor solution. However due to the theoretical foundations of the scale, we decided to use the three dimensional structure of efficiency, effectiveness, and innovativeness for which we standardized and calculated the Cronbach’s alphas of .76, .75, and .85, respectively.

The constructs for development process maturity and agility were adapted directly from a paper by Rönkkö, Järvi, and Mäkelä [45]. Although their scale was originally developed as a Rasch scale [cf., 46], it consists of simple agree-disagree questions and can hence be adapted to a study using more traditional measurement approaches. The scale for measuring the process maturity was based on a well-known Capability Maturity Model Integration (CMMI) software process maturity framework: CMMI for Development, Version 1.2 [26]. To improve the discriminant validity of the construct, i.e. its ability to measure software process maturity as a distinct construct from more general product development capability, we decided to exclude the so-called integrated product and process development (IPPD) additions to the CMMI. In total this scale consisted of 21 items. The adoption of agile methods was measured using a scale derived from on the basis of the two most well known agile methods and consisted of 16 items.

All scales that were adapted from previously published research were translated to Finnish using the double blind translate and back-translate procedure [47]. The translations were performed by the first authors of this paper, one researcher not related to this paper, and two research assistants. The translating protocol included translating also the context of the items, if the content validity of the original item was considered poor by the translator. The clarity and general validity of the items were checked by pre-testing all survey instruments with several experts and practitioners that had roles closely matching the ones of the informants. Based on these reviews, a number of items were reformulated.

The primary survey was implemented in two stages as follows. First, a pilot survey with about 11 percent of the sampling frame of 2616 firms was launched in April 2007. The pilot survey, mailed to 291 firms, was used to test the primary survey instrument. Minor modifications were made to the questionnaire. The main primary survey was thereafter sent to 2550 firms (2616 firms, from which we have excluded 66 firms that had responded to the pilot survey).

Both stages were implemented following a modified version of the tailored survey design method [48]. Mailings began with a pre-notice letter followed by the main survey package using postal mail. Two email reminders and one round of telephone calls were used to improve the response rate. A printed questionnaire and an online form were offered as alternative options to the informants. This phase of our data collection lasted from April through July 2007. Altogether, this phase produced 287 usable responses. After excluding firms with less than five employees as too small, the secondary survey was sent to 123 managers of software development or person in similar knowledgeable position with regards to software development in the firm. After several reminders by email and telephone, 83 firms provided responses to both surveys and were used in the main analyses. 9 cases were dropped due to the amount of missing data resulting in final sample of 72 companies.

We screened and prepared the data with Stata version 10. The effect of non-response was tested using two different methods. First, we compared the means of the key study variables between early and late respondents, as suggested by Oppenheim [49]. For the primary survey, we found that late respondents were larger ($p < .001$) and older ($p < .05$) than the early respondents. No significant differences were found for the secondary survey. Second, we compared the sampling frame with the respondents. For the primary survey, we found that industry codes 72.21 and 72.22 were overrepresented, which was expected considering that the survey targeted the software product industry and oversampling was used. We did not find any significant differences between geographical distribution and age of the sampling frame and the respondents. When comparing the respondents of the secondary survey to the sampling frame, no differences were found. In all, the results of our non-response analysis indicate that the results of this study are likely to be more valid for smaller than larger software product firms.

We used Partial Least Squares Path Modeling (PLS) as our main data analysis method. The reason for using this approach is the convenience it provides by testing all hypotheses and the validity of the measures in a single set of analyses. Although structural equation modeling would perform similar tests, we chose PLS because it has less stringent sample size and indicator distribution requirements than traditional SEM approaches [50]. Moreover, the method has increased popularity in IS recently [51] and hence we considered it more appropriate.

4 Results

We start examining the results of PLS by examining the measurement model. Table 1 shows the summary statistics for the constructs and Table 3 shows the construct-indicator cross loading matrix. Indicator loadings for the product development performance constructs are all above the recommended .707 threshold except the first two items. The first item is substantially below the limit for acceptable, but we decided to keep it nevertheless. The reason for this is that PLS model works better when the number of indicators is larger, all reliability indicators in Table 2 were good for this construct, and this poor item is weighted less than others so it should not cause bias in the results. All indicators in these scales load higher on their respective constructs than other constructs indicating discriminant validity of the three-dimensional product development performance scale.

The process agility and process maturity scales are more problematic since they have so many items. We considered parceling of the items, but this was not done due to the fact that to get unbiased results in PLS, the number of indicators needs to be large [52]. However, when the ratio of number of indicators to the number of cases is large, the indicator loadings tend to be unstable. Nevertheless, the reliability indices for these two constructs were high (Process Maturity) and acceptable (Process Agility) and hence we concluded that the overall degree of quality of measurement is sufficient.

Figure 1 shows the results of PLS estimation in the form of a path diagram and Table 2 shows the results of bootstrapping. All path coefficients are positive except for the path from Process Maturity to Efficiency. None of the paths between Process

Table 1. Construct reliability and validity

	AVE	Composite Reliability	R Square	Cronbachs Alpha	Commun-ality	Redund-ancy
Agility	0,1932	0,7772	0,0000	0,7531	0,1932	0,0000
Effectiveness	0,4225	0,7356	0,1649	0,5329	0,4225	0,0665
Efficiency	0,6093	0,8236	0,1929	0,6850	0,6093	0,0997
Inno-vativeness	0,7640	0,9066	0,0856	0,8454	0,7640	0,0521
Maturity	0,3742	0,9237	0,0000	0,9147	0,3742	0,0000

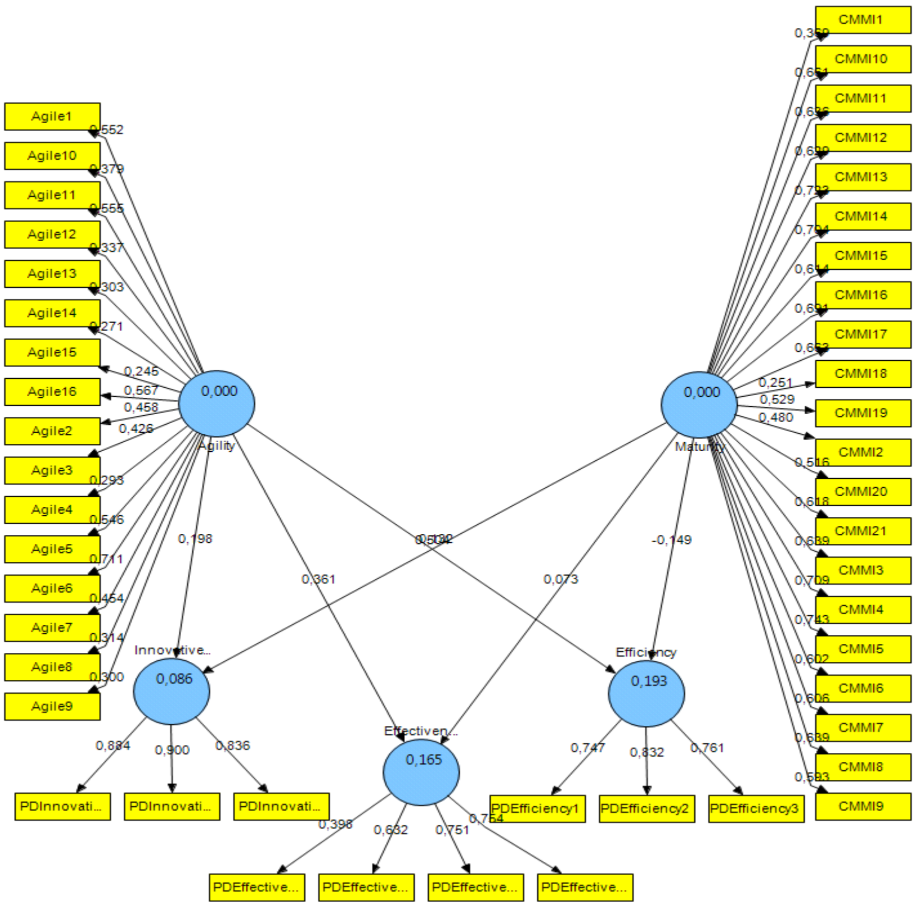


Fig. 1. Results of PLS estimation

Table 2. Bootstrapping results

	Original Sample	Sample Mean	Standard Error	T Statistics	p
Agility -> Effectiveness	0,3614	0,4046	0,1401	2,5787	0,05981
Agility -> Efficiency	0,5037	0,4787	0,1527	3,2974	0, 0007589
Agility -> Innovativeness	0,1981	0,2532	0,1535	1,2908	0,1005
Maturity -> Effectiveness	0,0726	0,1223	0,1272	0,5706	0,2850
Maturity -> Efficiency	-0,1485	-0,0693	0,1773	0,8378	0,2025
Maturity -> Innovativeness	0,1320	0,1590	0,1295	1,0189	0,1556

Table 3. Construct-indicator cross-loading matrix

	Agility	Effectiveness	Efficiency	Inno-vativeness	Maturity
Agile1	0,5515	0,3127	0,1299	0,2173	0,5430
Agile10	0,3789	0,1255	0,2487	-0,0273	0,0404
Agile11	0,5548	0,1869	0,2705	0,1668	0,4009
Agile12	0,3367	0,0768	0,2214	-0,0083	0,0376
Agile13	0,3031	-0,0318	-0,0773	-0,0028	0,5196
Agile14	0,2710	-0,0128	0,0527	-0,0955	0,3336
Agile15	0,2448	-0,0225	-0,0154	0,0362	0,3367
Agile16	0,5672	0,1005	0,3635	0,0891	0,3932
Agile2	0,4581	0,1095	0,0276	0,0608	0,3085
Agile3	0,4256	0,0781	0,0076	0,0688	0,3247
Agile4	0,2935	-0,0001	0,0229	0,1066	0,5750
Agile5	0,5457	0,2462	0,1487	0,1581	0,3604
Agile6	0,7113	0,3264	0,3295	0,2493	0,2257
Agile7	0,4541	0,1525	0,0068	-0,0359	0,4008
Agile8	0,3141	0,2139	0,0911	0,1171	-0,0456
Agile9	0,2999	0,1206	0,1487	0,1379	0,2503
CMMI1	0,2075	0,0245	0,0326	0,0685	0,3686
CMMI10	0,5046	0,2144	0,2208	0,0453	0,6608
CMMI11	0,4052	-0,0227	-0,1179	-0,0375	0,6358
CMMI12	0,4344	0,1157	0,1575	0,0425	0,6293
CMMI13	0,3204	0,1073	0,0087	0,1387	0,7229
CMMI14	0,3130	0,1407	0,0694	0,2226	0,7038
CMMI15	0,1868	0,1349	-0,1231	0,0695	0,6136
CMMI16	0,2867	0,0802	-0,1057	0,0536	0,6906
CMMI17	0,1609	0,2231	-0,1458	0,1694	0,6626
CMMI18	0,1469	0,1367	0,0535	0,0439	0,2506
CMMI19	0,6020	0,2361	0,2840	0,0237	0,5287
CMMI2	0,1281	0,1380	-0,0326	0,2104	0,4804
CMMI20	0,4241	0,1507	0,1119	0,0457	0,5156
CMMI21	0,2926	0,1742	-0,0233	0,0957	0,6183
CMMI3	0,3116	0,0983	0,0007	0,2018	0,6391
CMMI4	0,2585	0,2190	0,0707	0,2861	0,7086
CMMI5	0,5284	0,1554	0,2045	0,2393	0,7431
CMMI6	0,4737	0,2091	0,2277	0,1620	0,6023
CMMI7	0,3866	0,0686	-0,0611	0,0388	0,6062
CMMI8	0,1414	0,0767	-0,0478	0,1276	0,6392
CMMI9	0,2022	0,1720	-0,0713	0,1647	0,5934
PDEffectiveness1	0,0764	0,3985	0,1221	0,1919	0,1971
PDEffectiveness2	0,2938	0,6317	0,2349	0,2655	0,1857
PDEffectiveness3	0,3106	0,7509	0,4392	0,5817	0,0940
PDEffectiveness4	0,2969	0,7539	0,4742	0,6027	0,2428
PDEfficiency1	0,3087	0,2378	0,7466	0,2607	0,0174
PDEfficiency2	0,3045	0,4715	0,8316	0,4067	0,0402
PDEfficiency3	0,3594	0,4760	0,7609	0,4216	0,2093
PDInnovativeness1	0,2505	0,6057	0,4483	0,8841	0,1661
PDInnovativeness2	0,2519	0,6588	0,5216	0,9004	0,2425
PDInnovativeness3	0,2073	0,4617	0,2596	0,8365	0,2203

Maturity to the product development constructs are significant and hence we can conclude that hypotheses 1, 2, and 5 are not supported. When we look at the significance tests for the paths from Process Agility to the product development constructs, we can see that the path to Efficiency is significant, path to Effectiveness is marginally significant and path to Innovativeness is close to marginally significant. Hence we conclude that Hypothesis 3 is supported, Hypothesis 4 is weakly supported and Hypothesis 6 does not receive support from our data. The lack of support for the last hypothesis is probably due to lack of statistical power due to a small sample size.

5 Discussion and Conclusions

In this paper we studied the product development performance effects of agile software development practices and mature software process. To our understanding, this undertaking is one of the first studies investigating the relative merits of these models using survey research. Our key findings are that agile methods have a positive effect on product development performance, but process maturity does not seem to have an effect. This finding is a bit surprising, but can be explained by the fact that the companies in the sample are small firms and the process development frameworks are designed for predominantly large organizations that do larger software projects.

The fact that software process and innovativeness seem unrelated can be interpreted as meaning that there are other organizational factors that affect innovativeness much more than what software development methods are in use.

Regarding the limitations of the study, non-response analysis and descriptive statistics indicate that the results of the study are probably rather well generalizable among small and the smaller medium-sized software product firms. However, it needs to be pointed out that we have studied only Finnish firms. A number of issues of national culture or the peculiarities of Finland as a small economy that may influence business operations can quite possibly affect the results..

Also, as with any paper relating to self-administered surveys and related to composite variables is that our paper relies on self-reports. Although we spent considerable efforts to make the survey items as clear as possible [45] and tried to make the items such that also persons not familiar with CMMI or the particular agile methods used could reliably answer the questionnaire, it is possible that some of the items were too difficult for the respondents. However, our more detailed analysis [45] indicated that these problems should not be more serious than what is typical for a paper based on survey data. Finally, there have recently been some concerns related to the validity of the PLS approach [53-54], and should these concerns be validated in follow-up studies, they can have implications for this paper.

This paper has two managerial implications: First, at least in small firm context, using agile methods can substitute for process maturity. Although our sample includes only firms from CMMI levels four and below, this is an important finding that can, if supported by further research, help guide initial software process improvement efforts in smaller firms. Second, our data indicated that agile methods are probably more appropriate for product development than non-agile.

References

1. Kautz, K., Madsen, S., Norbjerg, J.: Persistent problems and practices in information systems development. Information Systems Journal 17, 217–239 (2007)
2. Vinekar, V., Slinkman, C.W., Nerur, S.: Can Agile and Traditional Systems Development Approaches Coexist? an Ambidextrous View. Information Systems Management 23, 31–42 (2006)
3. Balijepally, V., Mahapatra, R., Nerur, S., Price, K.H.: Are Two Heads Better Than One for Software Development? The Productivity Paradox of Pair Programming. MIS Quarterly 33, 91–118 (2009)
4. Dybå, T., Dingsøyr, T.: Empirical studies of agile software development: A systematic review. Information and Software Technology (2008)
5. Agrawal, M., Chari, K.: Software effort, quality, and cycle time: A study of CMM level 5 projects. IEEE Transactions on Software Engineering 33, 145–156 (2007)
6. Galin, D., Avrahami, M.: Are CMM program investments beneficial? Analyzing past studies. IEEE Software 23, 81–87 (2006)
7. Highsmith, J., Cockburn, A.: Agile Software Development: The Business of Innovation. IEEE Computer 34, 120–122 (2001)
8. Meso, P., Jain, R.: Agile Software Development: Adaptive Systems Principles and Best Practices. Information Systems Management 23, 19–30 (2006)
9. Poppendieck, M.: Lean Software Development. In: Companion to the Proceedings of the 29th International Conference on Software Engineering, pp. 165–166. IEEE Computer Society, Los Alamitos (2007)
10. Taylor, P.S., Greer, D., Sage, P., Coleman, G., McDaid, K., Lawthers, I., Corr, R.: Applying an agility/discipline assessment for a small software organisation. In: Proceedings of Product-Focused Software Process Improvement, pp. 290–304. Springer, Berlin (2006)
11. Baskerville, R., Pries-Heje, J.: Short cycle time systems development. Information Systems Journal 14, 237–264 (2004)
12. MacCormack, A., Verganti, R., Iansiti, M.: Developing products on “Internet time": The anatomy of a flexible development process. Management Science 47, 133–150 (2001)
13. Germain, É., Robillard, P.N.: Engineering-based processes and agile methodologies for software development: a comparative case study. The Journal of Systems & Software 75, 17–27 (2005)
14. Staples, M., Niazi, M., Jeffery, R., Abrahams, A., Byatt, P., Murphy, R.: An exploratory study of why organizations do not adopt CMMI. Journal of Systems and Software 80, 883–895 (2007)
15. Boehm, B., Turner, R.: Using risk to balance agile and plan-driven methods. Computer 36, 57–66 (2003)
16. Baker, S.W.: Formalizing agility: an agile organization’s journey toward CMMI accreditation. In: Proceedings of Agile Conference, pp. 185–192 (2005)
17. Merisalo-Rantanen, H., Tuunainen, T., Rossi, M.: Is extreme programming just old wine in new bottles: A comparison of two cases. Journal of Database Management 16, 41–61 (2005)
18. Paulk, M.C.: Extreme programming from a CMM perspective. IEEE Software 18, 19–26 (2001)
19. Pressman, R.: Software Engineering: A Practitioner’s Approach. McGraw-Hill Science/Engineering/Math. (2009)

20. Jiang, J.J., Klein, G., Hwang, H.G., Huang, J., Hung, S.Y.: An exploration of the relationship between software development process maturity and project performance. Information & Management 41, 279–288 (2004)
21. Schach, S.R.: Object-oriented and classical software engineering. McGraw-Hill, Boston (2002)
22. Herbsleb, J., Zubrow, D., Goldenson, D., Hayes, W., Paulk, M.: Software quality and the Capability Maturity Model. Communications of the ACM 40, 30–40 (1997)
23. Turner, R., Jain, A.: Agile meets CMMI: Culture clash or common cause? Extreme Programming and Agile Methods—XP/Agile Universe 2002, 153–165 (2002)
24. Carlshamre, P.: Release Planning in Market-Driven Software Product Development: Provoking an Understanding. Requirements Engineering 7, 139–151 (2002)
25. Jantunen, S., Smolander, K.: Towards global market-driven software development processes: an industrial case study. In: Proceedings of the 2006 International Workshop on Global Software Development for the Practitioner, pp. 94–100. ACM, Shanghai (2006)
26. SEI: CMMI for Development, version 1.2 (2006)
27. Beck, K., Andres, C.: Extreme programming explained: embrace change. Addison-Wesley Professional, Reading (2004)
28. Turk, D., France, R., Rumpe, B.: Assumptions underlying agile software-development processes. Journal of Database Management 16, 62–87 (2005)
29. Sutherland, J., Jakobsen, R., Johnson, K.: Scrum and cmmi level 5: The magic potion for code warriors. In: Proceedings of the 41st Annual. Hawaii International Conference on System Sciences, p. 466 (2008)
30. Cockburn, A.: Selecting a project's methodology. IEEE Software 17, 64–71 (2000)
31. Subramanian, G.H., Jiang, J.J., Klein, G.: Software quality and IS project performance improvements from software development process maturity and IS implementation strategies. Journal of Systems and Software 80, 616–627 (2007)
32. Kusunoki, K., Nonaka, I., Nagata, A.: Organizational capabilities in product development of Japanese firms: a conceptual framework and empirical findings. Organization Science 9, 699–718 (1998)
33. Kahn, K.B.: Market orientation, interdepartmental integration, and product development performance. The Journal of Product Innovation Management 18, 314–323 (2001)
34. McDermott, C.M., O'Connor, G.C.: Managing radical innovation: an overview of emergent strategy issues. Journal of Product Innovation Management 19, 424–438 (2002)
35. Veryzer, R.W.: Discontinuous Innovation and the New Product Development Process. Journal of Product Innovation Management 15, 304–321 (1998)
36. Nidumolu, S.R.: Standardization, requirements uncertainty and software project performance. Information and Management 31, 135–150 (1996)
37. Citrin, A.V., Lee, R.P., McCullough, J.: Information use and new product outcomes: The contingent role of strategy type. Journal of Product Innovation Management 24, 259–273 (2007)
38. Karlsson, L., Dahlstedt, A.G., Regnell, B., Nattoch Dag, J., Persson, A.: Requirements engineering challenges in market-driven software development - An interview study with practitioners. Information and Software Technology 49, 588–604 (2007)
39. Slaughter, S.A., Levine, L., Ramesh, B., Pries-Heje, J., Baskerville, R.: Aligning software processes with strategy. Mis Quarterly 30, 891–918 (2006)
40. Tushman, M.L.: Special boundary roles in the innovation process. Administrative Science Quarterly 22, 587–605 (1977)
41. McDonough, E.F.: Investigation of factors contributing to the success of cross-functional teams. Journal of Product Innovation Management 17, 221–235 (2000)

42. Spreitzer, G.M.: Psychological empowerment in the workplace: Dimensions, measurement, and validation. Academy of Management Journal 38, 1442–1465 (1995)
43. Tierney, P., Farmer, S.M.: Creative self-efficacy: Its potential antecedents and relationship to creative performance. Academy of Management Journal 45, 1137–1148 (2002)
44. Rönkkö, M., Eloranta, E., Mustaniemi, H., Mutanen, O., Kontio, J.: Mustaniemi, H., Mutanen, O., Kontio, J.: Finnish Software Product Business: Results of National Software Industry Survey 2007. Helsinki University of Technology (2007)
45. Rönkkö, M., Järvi, A., Mäkelä, M.M.: Measuring and comparing the adoption of software process practices in the software product industry. In: Proceedings of Internationl Conference on Software Process, Leipzig, Germany, pp. 407–419 (2008)
46. Dekleva, S., Drehmer, D.: Measuring software engineering evolution: A rasch calibration. Information Systems Research 8, 95–104 (1997)
47. Brislin, R.W.: Back-Translation for Cross-Cultural Research. Journal of Cross-Cultural Psychology 1, 185–216 (1970)
48. Dillman, D.A.: Mail and Internet surveys: the tailored design method. Wiley, New York (2007)
49. Oppenheim, A.N.: Questionnaire Design and Attitude Measurement Heinemann, London (1966)
50. Chin, W.W.: The partial least squares approach to structural equation modeling. In: Marcoulides, G.A. (ed.) Modern Methods for Business Research, pp. 295–336. Lawrence Erlbaum Associates Publishers, Mahwah (1998)
51. Ahlemann, F., Urbach, N.: Structural Equation Modeling in Information Systems Research Using Partial Least Squares. Journal of Information Technology Theory and Application (JITTA) 11 (2010)
52. Dijkstra, T.: Some comments on maximum likelihood and partial least squares methods. Journal of Econometrics 22, 67–90 (1983)
53. Evermann, J., Tate, M.: Testing Models or Fitting Models? Identifying Model Misspecification in PLS. In: Proceedings of the ICIS 2010 (2010)
54. Rönkkö, M., Ylitalo, J.: Construct Validity in Partial Least Squares Path Modeling. In: Proceedings of the ICIS 2010 (2010)

Online Distribution of Packaged Software

Shuangzeng Hu and Rod B. McNaughton

Conrad Business, Entrepreneurship and Technology Centre
University of Waterloo
295 Hagey Blvd., Suite 240
Waterloo, Ontario Canada N2L 6R5
rmcnaughton@uwaterloo.ca

Abstract. Increased bandwidth and user sophistication make it practical for software developers to sell and distribute packaged software to customers online. This paper develops a transaction cost model of the conditions under which software developers are more likely to distribute their packaged software online rather than through traditional channels. The model is tested using data from a survey of Canadian software firms. Almost three-quarters of the respondents use the Internet at least in part to distribute their products. Firms are more likely to distribute their packaged software online in less diverse markets and where channel volume is increasing rapidly. However, the results are not consistent with other conditions posited to be associated with online distribution. Implications for understanding how the Internet is changing the transaction costs of distributing digital products are discussed.

Keywords: packaged software, distribution channels, transaction cost analysis.

1 Introduction

The research reported in this paper tests a model of the conditions in which software vendors are likely to sell and distribute their packaged products to end-users through the Internet. As software is a digital product, it is easily transferred from developer to user on the Web. Thus, the Internet has become a key element in the marketing strategy of software firms throughout the selling cycle from promotion, sales and payments, downloading, after-sale service and updates, and monitoring accounts. Online software distribution benefits both developers and customers in terms of time and place utility. The effective use of the Internet enables software developers to provide anytime/anywhere access to their products. With online software distribution, customers can purchase and immediately download software products to their computers. In addition, online software distribution reduces logistics, hardware and supply costs, and allows software firms to address more points of sale worldwide at low cost. This is facilitated by the penetration of high-speed broadband access and high performance desktop computers. The Internet is a particularly potent marketing medium for smaller software firms, as it can help them to compete on a more "level playing field" with larger firms.

Packaged software is an important part of the software industry. The worldwide sales of packaged software for all platforms was $US 297 billion in 2008 (IDC, 2009). Packaged software is traditionally purchased "off-the-shelf" and is often referred to as "shrink-wrapped" after a common form of packaging. However, packaged software can also be downloaded, or even remotely hosted by a service provider. The Software & Information Industry Association (2005) defines packaged software as that which is "…written for mass distribution, not for the specific needs of a particular user, and may be distributed in any format – electronic download, physical media such as disk or CD, or a web-based service." We explicitly exclude software designed solely for mobile devices as this lacks the size and complexity generally associated with packaged software, and is only distributed through downloads or original equipment manufacturers (OEMs) (i.e., it is included with a mobile device).

We view the decision to distribute online through the lens of transaction cost analysis (TCA). Several researchers have used transaction cost theory to address the issues of marketing channel choice and software outsourcing in the software industry (e.g., McNaughton, 1996, 2002; Eric and Wang, 2002). However, this literature pre-dates the widespread commercial use of the Internet, especially for downloading and hosting software products. Thus, the fundamental question addressed by this research is how and when packaged software developers use the Internet to deliver their products.

The following section reviews transaction cost theory, and existing empirical studies of software distribution. We then develop hypotheses of the conditions under which developers are more likely to sell and distribute their products online. Further sections describe the collection of data from the senior executives of Canadian packaged-software developers, and statistical tests of the hypotheses. The paper concludes with a discussion of the implications of the findings for using TCA to understanding the choice of online distribution, and practical guidance for firms that face the decision of whether to sell their packaged software online.

2 Literature Review

There is a large literature on transaction cost analysis, much of which was spawned by the writings of Williamson (1975, 1979, 1981, 1985, 1996). As the theory is well-known and reviews are available elsewhere, we focus on previous applications of this theory to explain the decision about the distribution of software products. In essence, TCA argues that because markets are frequently inefficient, the costs associated with activities such as searching for information, negotiating terms, and monitoring contracts are an important part of ensuring a favorable deal. Williamson (1981) argued that there are three critical dimensions of transactions that influence these costs:

1. Asset specificity: durable investments that are undertaken in support of particular transactions, and that are difficult or costly to transfer to other uses.
2. Uncertainty: the cost associated with unexpected outcomes and asymmetry of information. Uncertainty arises due to opportunism, bounded rationality, and asymmetry of information.
3. Transaction frequency: how often a particular type of transaction recurs.

Transaction cost theory can be used to identify conditions in which firms are more likely to extend forward into distribution rather than outsource those activities to third parties. Whether internal organization or market exchange is preferred depends on the relevant transaction costs. TCA argues there is little incentive to integrate in a competitive market because transaction costs are relatively low. In contrast, firms prefer to internalize their distribution channels when transaction costs are high and difficult to control through pricing and/or behavioral constraints. From a TCA perspective, distribution channel choice involves decisions about how to minimize the total cost of transferring a product from developer to customer, including both production and transaction costs.

McNaughton (1996, 2002) and McNaughton and Bell (2001) used TCA to explore several aspects of how software firms distribute both their packaged and customized software. McNaughton (1996) used TCA to analyze the impacts of product and market attributes on the selection of distribution channels by Canadian software firms when they sell to export customers. He concluded that channel volume is positively associated with the use of foreign sales subsidiaries and negatively associated with the use of shared control modes such as distributors; and that asset specificity is negatively associated with the use of shared control modes and positively associated with the use of foreign sales subsidiaries. Firms selling primarily packaged software were more likely to have overseas sales subsidiaries or joint ventures, and were less likely to use market-based channels compared with firms that sell customized software services.

McNaughton and Bell (2001) tested seven hypotheses about the conditions in which software firms use a lower control mode in an export market than in their domestic market. Their model includes variables for asset specificity, external uncertainty, production cost efficiency, specific market (the U.S. market or other), and customization (packaged software products versus customized software products). They found that firms tend to use the same channels in both domestic and foreign markets: only 23% of the respondents used a different channel when entering a foreign market. Amongst those that did switch, the most common change was from a higher to a lower control mode, and this was negatively associated with knowledge-based asset specificity, physical asset specificity, and market size; but positively with market diversity. The authors conclude that software businesses need to be conscious of the momentum of their domestic channels and carefully evaluate whether or not it is appropriate to extend their domestic channels into a foreign market.

In a follow-up study, McNaughton (2002) developed a TCA model of channel choice to identify conditions that increase the likelihood that multiple channels are used to serve a foreign market. He found that neither channel volume nor growth rate is associated with the use of multiple channels, leading to the conclusion that multiple channels are used to increase sales volumes, rather than being a consequence of them. The study also found that multiple channels are more likely to emerge in mature markets that are experiencing slower growth.

Finally, two other researchers, Eric and Wang (2002), studied how transaction attributes and post-contractual opportunism affect the success of customized software outsourcing. In this model, the dependent variables were outsourcing success and post-contractual opportunism, while the independent variables were contractor reputation, uncertainty, and asset specificity. Data to test the model were collected from medium to large-sized software firms in Taiwan. The results show that asset specificity and uncertainty both significantly reduce a contractor's post-contractual opportunism, and increase outsourcing success.

3 Transaction Cost Model

The traditional distribution channels available to developers of packaged software include:

1. Retail distribution channels. For example, computer vendors, big-box stores, office equipment vendors, stores specializing in software sales, and warehouse clubs.
2. Distributors. These include horizontal distributors (who typically carry many software titles) and vertical distributors that concentrate on a few categories of software and target a vertical market. Distributors generally do not offer a high level of technical support, and sell primarily to retail stores and other resellers (Wilson, 2001).
3. Original equipment manufacturers. These are vendors or manufacturers of computer hardware. By integrating specialized products, hardware, and services, OEMs sell turnkey products intended for a specific use.
4. Systems integrators. These are often large consulting companies or OEMs, and usually specialize in a particular vertical market.
5. Service partners (or value-added resellers - VARs). These firms typically add value to software products through consulting, customization, and/or training services. Most VARs work with distributors and do not keep their own stock.
6. Direct marketing channels. In this case, developers sell their packaged software directly to clients. This can include in person selling, direct-mail advertising, and telemarketing.

Selling and downloading software from the Internet is a form of direct marketing when the software developer uses their own Website. In some cases, however, software is sold through a distributor's Website, transforming the Internet into an indirect channel. While either case is less expensive than traditional marketing channels, and incurs no shipping or handling costs, most vendors sell and distribute directly through their own Website, thus avoiding having to share margin with a channel partner.

Online distribution also enables numerous alternative business models and additional services, for example, remote hosting and per-use or subscription-based pricing. In addition, multiple distribution channels can be managed through secure affiliate sites. As a consequence, online selling and distribution is quickly becoming the norm for packaged software. However, the basic TCA argument is that a market mode is the default choice as it is more efficient. Thus, we begin with the assumption that firms will distribute their software through a traditional indirect channel unless there is a compelling reason to do otherwise, and our model develops expectations about the conditions under which firms are more likely to sell and distribute directly to customers through their own Website.

3.1 Asset Specificity

Prior research uses two dimensions to reflect the degree of asset specificity associated with software: physical and knowledge-based asset specificity. Physical asset specificity refers to investment in special equipment, such as computers, servers or routers for the purpose of software distribution, while knowledge-based asset specificity refers to unique knowledge and expertise that is required to sell or use the software. TCA argues that asset specificity increases transaction costs as both parties may have to invest in specific assets, such as skilled sales persons or unique equipment (e.g., Thompson et al, 2004). In such conditions, integrated (i.e., direct) distribution channels are more efficient as the developer will typically already have the required equipment and expertise, and it will be difficult to find third parties (i.e., channel partners) who are willing to make the required investment. Thus, to the extent that online distribution is typically done by the vendor:

H1. The use of the Internet in the distribution of packaged software is positively related to the physical asset specificity of the software.

H2. The use of the Internet in the distribution of packages software is positively related to the knowledge-based asset specificity of the software.

3.2 Uncertainty

We identify two sources of uncertainty in the market for packaged software: diversity and volatility. In a volatile market, software firms have difficulty predicting customer demands and competitor actions. In this situation, a direct channel is more likely to be used as few third parties will be willing to assume the risk of environmental volatility, or would demand a premium for doing so. Therefore:

H3. The use of the Internet in the distribution of packaged software is positively related to environmental volatility.

Diversity stems from multiple sources of uncertainty in a market. A diverse market is one in which there are many customers and competitors and they are heterogeneous in their characteristics. In such a market, a firm will need to develop multiple strategies to meet varied and specialized demands. A more complex and flexible channel structure can be created by including channel partners that help to gather and process the information required to deal with a heterogeneous market. As the diversity of the environment increases, multiple channels may be used, and the relative importance of direct distribution of software through the Internet may decline. Thus:

H4. The use of the Internet in the distribution of packaged software is negatively related to environmental diversity.

3.3 Transaction Frequency

Two dimensions are used to measure transaction frequency: sales volume and sales growth. TCA argues that firms have more incentive to integrate their distribution channels for high volume transactions as the costs of developing and maintaining the channel are spread over more transactions. In addition, there is an incentive to retain the margin that would otherwise be shared with channel partners. Compared with other forms of direct distribution, online distribution has relatively low set-up costs, and almost negligible unit costs. Consequently, it is financially feasible for software firms to integrate their distribution online at much lower sales volumes than is the case for traditional direct channels. In contrast, large sales volumes provide more financial resources that can be used to explore relatively more expensive channels that may be more effective in reaching additional customer segments and in gathering information to further refine the product. McNaughton (2002) observed that in more mature markets where sales have slowed, developers often use multiple channels to refine the way they target segments. In such cases, firms are likely to continue distributing online to some segments, while serving others with personal sales or VARS, and supporting customers online in more refined ways (e.g., targeted or personalized online sales sites, or support through live chat). Therefore:

H5. The use of the Internet in the distribution of packaged software is positively related to growth in sales volume.
H6. The use of the Internet in the distribution of packages software is negatively related to gross sales.

4 Method

To test these hypotheses, data were collected from Canadian software developers in an online survey. The target population of CEOs or other executive leaders of software firms was identified from the Canadian Company Capabilities online directory (CCC) maintained by Industry Canada. Recruitment e-mails were sent to executives at 1142 software development firms that are Canadian owned (i.e., not subsidiaries of firms based in the US or elsewhere). Complete responses were received from 82 firms, for a response rate of 7%. Unfortunately, low rates are becoming the norm in online surveying. Replies to the recruitment and follow-up e-mails suggests that about one third of the targeted firms did not have a software product (i.e., they provide customized programming services only) and thus did not respond. The number of e-mails reaching their intended recipients was also reduced by errors and changes in e-mail addresses, and messages being trapped by spam filters.

Krosnick (1999, 540) argued that "surveys with very low response rates can be more accurate than surveys with much higher response rates," and that "having a low response rate does not necessarily mean that a survey suffers from a large amount of nonresponse error". This conclusion is based on Visser et al.'s (1996) study showing that data accuracy, measured as the difference between predicted and actual outcomes, was higher in a low-response rate mail survey than in a higher response rate telephone survey. The most important consideration is if the reason for low response is related to the topic of the survey, which is unlikely in our case. To check for nonresponse bias we compared the first and last quartile of respondents, and found no statistically significant differences in demographics or mean responses to the key dependent and independent variables.

Consistent with the structure of the Canadian software industry, the majority of respondents represented small firms: 30% had between 1 and 5 employees, and 75% employed fewer than 30 employees. Only 10% had more than 100 employees. The firms were generally young, with two-thirds having been founded since 2000. Respondents were asked to provide data for their best-selling software product in its largest market. This combination accounted for an average of 63% of the gross sales of respondents. Many of the firms only had one software product. For most Canadian software firms, the largest market for their best-selling product is either Canada (48%) or the United States (44%). Only 8% cited a different national market. Respondents characterized their software product as a horizontal application (16%), vertical market application (78%), or games and educational software (6%).

4.1 Dependent Variable

The dependent variable measured whether or not a firm uses the Internet for distribution of packaged software. Respondents were asked a series of questions to determine if they use the Internet to distribute their best-selling product, and details about how it

Table 1. Primary Channel for Distribution of Packaged Software

Primary channels	Frequency	Percent
Internet	32	39.0
Offline direct marketing channels (personal sales, telesales, direct mail, etc.)	29	35.4
Distributors, publishers or wholesalers	14	17.1
OEM	4	4.9
Value added resellers	1	1.2
Other	2	2.4
Total	82	100.0

is used or their other channels. Table 1 shows the primary distribution channel reported by the respondents. Thirty-nine percent use the Internet, while 35% sell directly using a traditional direct method. The remainder use a traditional indirect method. However, The classification of distribution channel type is more complex, as it is possible for firms to use the Internet in conjunction with another method (i.e., complete only part of the selling and distribution function online), alongside another method (i.e., multiple channels), and for the Website to be controlled by either the software developer or a partner (e.g., an online software distributor). Table 2 shows that almost three-quarters of the firms incorporate the Internet in some way as part of their distribution channel, while Table 3 provides detail on the frequency with which channel activities are conducted online.

4.2 Independent Variables

The independent variables were measured using Likert scales adapted from previous research (McNaughton 1996, 2002 and McNaughton and Bell, 2001). Respondents were asked to indicate their level of agreement ("Strongly disagree"=1, "Strongly agree"=7) about a series of statements that related to physical and knowledge-based asset specificity, volatility and diversity. Respondents were also asked to estimate the current sales of their best-selling software package in its largest national market, and the annual growth rate of those sales. Table 4 provides descriptive statistics for the independent variables. Gross sales and growth in sales are highly skewed, so were rank transformed before the analysis. We also include a control variable to indicate if the largest market is domestic (0) or foreign (US or international = 1).

Table 2. Use of the Internet for Distribution of Packaged Software

Online Distribution	Frequency	Percent
Internet is the primary method of distribution	32	39.0
Internet is part of the distribution channel	29	35.4
No part of distribution is online (except viewing promotional material)	20	24.4
Missing	1	1.2
Total	82	100.0

Table 3. Channel Activities Conducted Online

Channel activity	Frequency
View promotional material	82
After-sale service and support (including documentation)	34
Download patches and/or updated versions	43
Download a beta or trial version	23
Download full software package	32
Make payment online	30
Online training	2
Missing	1

Frequencies sum to more than the number of respondents as respondents could indicate all activities that apply.

Table 4. Descriptive Statistics for Independent Variables

Variable	N	Mean	S.D.	Min.	Max.
Knowledge-based Asset Specificity	82	5.5	1.6	1.0	7.0
Physical Asset Specificity	82	2.1	1.6	1.0	7.0
Diversity	82	4.3	2.1	1.0	7.0
Volatility	82	4.2	2.0	1.0	7.0
Gross annual sales ($CAN millions)	79	41.9	191.7	0.05	100.0
Annual growth in sales (%)	82	60.0	130.0	0	500.0

5 Findings

To test the hypotheses developed in Section 3, we first fit a binary logistic regression model in which firms that use the Internet as part of their distribution channel (i.e., distribute completely online or incorporate the Internet as part of the channel) are compared with those that do not. Table 5 reports the result of this analysis. The overall model is statistically significant, and has a moderate fit and classification rate. However, the parameters for several of the variables are very small and not statistically distinguishable from 0.0. The sign is reversed from the expectation for H2, and there is support for hypotheses H5 and H6. The control variable is not significant.

To better understand the relationships, we fit an additional polychotomous model in which the dependent variable distinguishes between those firms that use the Internet as their primary distribution channel, those that include the Internet along with offline components, and those that do not use the Internet as part of their distribution channel. Table 6 reports the results. We only include the variables that were statistically significant in the first model to reduce the number of parameters that needed to be estimated. This additional analysis has similar results to those reported in Table 5, except that knowledge-based specificity is not statistically significant in either case (at $p=0.05$), and gross sales is not significant for partial use of the Internet.

Table 5. Relationships Between Transaction Cost Variables and Use of the Internet in Distribution Channel

Parameter	Expected sign	Estimate	Chi-Square	Pr > Chi Sq
Intercept		7.44	6.81	0.01
Physical asset specificity	H_1 (+)	--	--	1.00
Knowledge asset specificity	H_2 (+)	-0.51	3.80	0.04
Volatility	H_3 (+)	--	--	1.00
Diversity	H_4 (-)	-0.58	5.02	0.03
Growth rate	H_5(+)	0.08	10.05	0.00
Gross sales	H_6 (-)	-0.08	13.10	0.00
Largest market	Control	--	--	1.00

Goodness of fit (residual test) chi-square = 3.22 (p=0.78 and df=6); Hosmer and Lemeshow Goodness-of-Fit test =15.24 (df=7, P=0.03); Correct classification rate = 82.9%, R^2 = 0.56.

Table 6. Relationships Between Transaction Cost Variables and Use of the Internet as Primary Distribution Channel or Part of Distribution Channel

Parameters for online distribution as the primary channel	Estimate	Chi-Square	Pr > Chi Sq
Intercept	8.09	8.19	0.00
Knowledge specificity	-0.58	3.01	0.08
Diversity	-0.63	5.85	0.02
Growth rate	0.12	16.84	0.00
Gross sales	-0.14	18.72	0.00

Parameters for partial use of Internet in channel	Estimate	Chi-Square	Pr > ChiSq
Intercept	5.85	5.00	0.03
Knowledge specificity	-0.55	3.61	0.06
Diversity	-0.65	5.85	0.02
Growth rate	0.06	6.92	0.01
Gross sales	-0.04	2.97	0.09

Goodness of fit (residual test) chi-square = 10.79 (p=0.70 and df=14); Correct classification rate = 79.3%, R^2 = 0.63.

6 Conclusions

About three-quarters of the firms in our sample incorporate the Internet into the distribution channel for their best-selling software product. However, only about 40% complete the entire distribution process online. Of the six hypotheses developed in Section 3, only two are supported by the empirical evidence. Contrary to the expectations and evidence from prior research (e.g., McNaughton 1996), physical asset specificity does not play a role in the choice of distribution channel. The parameter estimate for knowledge-based asset specificity is statistically significant, but the sign is opposite to our expectation. Software firms are less likely to incorporate the Internet into their distribution channel when knowledge-intensity is high. (However, this effect is not statistically significant in our second model.) Environmental uncertainty in the form of volatility has no influence on the use of the Internet in the channel, while diversity is a positive influence as expected. Annual growth in sales volumes also has a significant positive effect, and sales volume a negative effect as expected. However, gross sales has no significant relationship in the case of incorporating the Web as part of the channel (p=0.05).

7 Discussion

As our expectations are not strongly supported by the empirical evidence, we looked to the answers respondents provided to open-ended questions in the survey for additional insight. The absence of a relationship between physical asset specificity and use of the Internet may be explained by the wide availability of high performance computing. Few software packages require an investment in specialized equipment. As one respondent wrote: "Our software is a large system that we formerly marketed through sales agents. Now we market our product on the Internet, and offer updates online for download." The results for knowledge-based specificity are counter to the expectation, with more knowledge intense products being less likely to be distributed online.

An explanation may lie in the desire to protect unique intellectual property, or the notion that cutting-edge and advanced knowledge may be difficult to codify and transfer in an online environment. Vasiu (2003, 1), for example, found that online distribution "may also render organizations more vulnerable to electronic fraud (e-fraud)." E-fraud, including license breaking, can have significant financial implications, and adversely affect the decision to distribute online. Some developers of packaged software may keep their products off-line as a means of protecting them. This could extend to competitive information about pricing, rather than the software itself. For example, two of the 32 firms that make their full software package available for download do not take payments online. Instead, once they pay offline, clients are given access to a secure Website to download the software. As one respondent commented: "The Internet is a valuable sales channel, but pricing is discussed privately via e-mail and telephone. Therefore, there are no online financial transactions."

An additional consideration is that some developers (28%) do not sell their software through their own Website, rather they sell through the site of an online distributor or catalogue. In these cases, online distribution is a market rather than an integrated mode of distribution. Unfortunately, the number of these cases was too small for us to model them separately. Nor is it obvious in these cases that the distribution process is entirely outsourced. For example, one respondent commented: "We use our own site and those of various partners for marketing, but there is always a phone or in-person component of the sales process. Technically, online distribution is easy, but we sometimes mail our software on CD, or even install it in person."

Diversity is negatively associated with the use of the Internet for distribution in both models. In more diverse software markets, developers turn to traditional distribution channels to target specific segments and gain valuable feedback. As two of the respondents commented: "At this time we use the Internet primarily as an information portal to our products. As our software is scientific and can be used in various applications, assistance from our company or its partners is needed to determine the best piece of software", and "Our packaged software is a very sophisticated product that requires training and face-to-face advice from our sales agents."

The variables relating to gross sales and sales growth measure production costs rather than transaction costs. The argument relates to the ability to spread costs of channel development over a large number of sales. However, online distribution fundamentally changes the economics of distribution. Online distribution is relatively inexpensive compared to alternatives, so developers are able to integrate their channel at much lower volumes. Online distribution contributes to growth in sales. However, when sales volumes are higher and growth slows, firms turn to more diverse channels to reach additional segments, and create value for those customers who are willing to pay a premium.

This study is one of very few that explores the channel choices of software developers, and possibly the only one to address the issue of the conditions in which developers are more likely to use online distribution. This research extends the early work of McNaughton (1996, 1999, 2002) and McNaughton and Bell (2001) which largely predated the online distribution of software. Transaction costs appear less relevant in determining whether developers deliver their software online or not, than they were in distinguishing between the use of traditional market and integrated forms of distribution. As software is a digital product that can be transferred online, the Internet is fundamentally restructuring the economics of this industry. While Internet use is almost universal for some aspects of the software distribution process, there remain nuances in the extent to which all aspects of the selling and distribution process are handled online. In particular, offline components play a role in protecting knowledge-based assets, in dealing with diversity in markets, and later in the product life-cycle when volumes are larger but growth slows. In these cases, offline components can help to identify and serve additional segments that require more advice or service, and that are willing to pay a premium for value-added channels.

References

1. Barnett, V.: Sample survey: Principles and methods, 3rd edn. Oxford University Process, London (2002)
2. Eric, T., Wang, G.: Transaction Attributes and Software Outsourcing Success: An Empirical Investigation of Transaction Cost Theory. Information Systems Journal 12, 153–181 (2002)
3. Krosnick, J.A.: Survey Research. Annual Review of Psychology 50, 537–567 (1999)
4. Liang, T.P., Huang, J.S.: An Empirical Study on Consumer Acceptance of Products in Electronic Markets: A Transaction Cost Model. Decision Support Systems 24, 29–43 (1998)
5. Lohrke, F.T.: The Internet as an Information Conduit: A Transaction Cost Analysis Model of Small Business Internet Use (2002), `http://www.usasbe.org/knowledge/proceedings/2002/38.pdf` (retrieved December 22, 2005)
6. McNaughton, R.B.: Foreign Market Channel Integration Decision of Canadian Computer Software Firms. International Business Review 5(1), 23–52 (1996)
7. McNaughton, R.B.: Disk by Mail for Industrial Survey Research: a Review and Example. Industrial Marketing Management 28(3), 32–47 (1999)

8. McNaughton, R.B., Bell, J.: Channel Switching between Domestic and Foreign Markets. Journal of International Marketing 9(1), 24–39 (2001)
9. McNaughton, R.B.: The use of multiple export channels by small knowledge-intensive firms. International Marketing Review 19(2/3), 190–203 (2002)
10. The Software & Information Industry Association (TSIIA): Packaged Software Industry Revenue and Growth (2005), `http://www.siia.net/software/pubs/growth_software05.pdf` (retrieved September 10, 2005)
11. Thompson, S.H., Wang, P., Leong, H.C.: Understanding online shopping behavior using a transition cost economics approach. Internet Marketing and Advertising 1(1), 62–84 (2004)
12. Vasiu, L.: A conceptual framework of e-fraud control in an integrated supply chain (2003), `http://csrc.lse.ac.uk/asp/aspecis/20040168.pdf` (retrieved March 26, 2006)
13. Visser, P.S., Krosnick, J.A., Marquette, J., Curtin, M.: Mail Surveys for Election Forecasting? An Evaluation of the Columbus Dispatch Poll. Public Opinion Quarterly 60, 181–227 (1996)
14. Williamson, O.E.: Markets and hierarchies: Analysis and antitrust implications. The Free Press, New York (1975)
15. Williamson, O.E.: Transaction-Cost Economics: The Governance of Contractual Relations. Journal of Law and Economics 22, 233–262 (1979)
16. Williamson, O.E.: The Economics of Organization: The Transaction Cost Approach. American Journal of Sociology 87, 548–577 (1981)
17. Williamson, O.E.: The Economic Institutions of Capitalism: Firms, Markets, Relational Contracting. The Free Press, New York (1985)
18. Williamson, O.E.: The Mechanisms of Governance. Oxford University Press, New York (1996)
19. Wilson, L.H.: Software development industry study. Business & Research Services. The U.S. Small Business Administration (2001), `http://www.spa.org/sharedcontent/press/2000/6-6-00.html` (retrieved March 25, 2006)

Scenarios on Adoption of Open Source Software in the Communications Software Industry

Eetu Luoma[1], Mikko Riepula[2], and Lauri Frank[1]

[1] University of Jyväskylä, Department of Computer Science and Information Systems, P.O. Box 35 (Agora), FI-40014 University of Jyväskylä, Finland
{eetu.luoma,lauri.frank}@jyu.fi
[2] Aalto University, School of Economics, P.O. Box 21210, FI-00076 Aalto, Finland
mikko.riepula@aalto.fi

Abstract. We examine the trends and developments affecting the adoption of open source software in the communications software industry. Based on expert interviews and scenario analysis, four alternative and possibly co-existing scenarios are derived. The analysis suggests that communication service providers will mostly deploy open source software in infrastructure software. Alternative developments include use of open source software in launching new services and in cloud computing. The present study is relevant particularly for technology managers considering open source strategies.

Keywords: open source software, scenarios, communications industry, communications software, telecom software, vertical software markets, technology management.

1 Introduction

Software produced for vertical industries is a considerable business for software vendors and a significant expense for its buyers in the industry. Our focus is on the *communications software market*[1], i.e. both the software vendors and *communications service providers*[2] (CSPs). As expected, also the communications software market has been found to be developing according to a pattern where software at first provides

[1] The *communications software market* is formed by the communications software vendors and the communication service providers (CSPs) as their customers. It is a *vertical software market,* producing software products and services mainly applicable within the communications industry, as opposed to producing horizontal (general-purpose) software.

[2] The *communications industry* is formed by the CSPs and their customers. As a vertical industry, it has a clear specialization and limited transferability of skills outside its own domain. By avoiding the more traditional "telecommunications" term we want to account for the changing industry landscape where not only traditional telecom operators, but a variety of actors ranging from ISPs and cable operators to the likes of Google and Amazon, are at play and capable of making attractive end-user offerings in at least parts of the communications domain.

competitive advantage to the companies in the form of technological or process innovation, but later, as software becomes a commodity, it is mainly a cost issue [1]. In the latter and currently observable phase of price erosion and cost cutting, the actors in the market can benefit from two low-cost models, namely open source software (OSSw) and cloud computing. At present cloud computing, and particularly software as a service (SaaS) as a part of it, are being adopted widely, also in the communications software market. OSSw in turn has already achieved a prominent share in horizontal infrastructure software, as in operating systems, databases, and application servers.

This article concentrates on a) the utilization of OSSw by communication service providers and b) use of OSSw by the software vendors in their offerings. Past research has indicated that the CSPs are reluctant to adopt OSSw, owing to the perceived associated risks [2]. Assuming that more recently the attitudes towards OSSw have changed with the positive experiences from Linux, MySQL and Apache, the use of and contributions to OSSw may increase also in the communications software market.

Open source software generally refers to software, which source code is made available to all users along with the usual run-time code and which has licenses with relaxed or very few restrictions. Using analytical and case study approaches, the contemporary academic literature around the phenomenon has focused on the incentives to produce and deploy OSSw, in the management issues of OSSw projects as well as comparison of open source and proprietary software from a business perspective [3–5]. While much has also been written about the antecedents of OSSw adoption [6–8], there are only few articles available where OSSw is studied in the context of vertical industries [10, 11].

This article reports a *scenario elaboration* on the trends and alternative development paths, which may affect the adoption of OSSw in the communications software market in the future. We also draw conclusions on how OSSw adoption would increase in the domain. We applied a set of scenario elaboration techniques, including domain expert interviews and workshops, morphological analysis, and the Global Business Network (GBN) method.

In the next section, the current rate and form of OSSw utilization in CSPs is examined. Section 3 gives an overview to the scenario method and techniques applied in this study. Sections 4 and 5 deal with the trends and change drivers of the communications software market and develop alternative or co-existing scenarios for OSSw in the communications industry. Before the conclusions, we discuss the benefits and limitations of the scenario approach we used.

2 Scenario Approach

Scenarios are defined by Ogilvy and Schwartz as "narratives of alternative environments in which today's decisions may be played out" [12]. In assessing possible developments for OSSw in the domain, we chose an exploratory design where future scenarios were developed based on a combination of interviews and a workshop with industry experts and a set of scenario techniques. The aim was not to forecast one future state, but to describe alternative developments providing a view on *how* adoption of OSSw would increase in the communications industry. Such information enables setting a technology strategy, which matches the emerging needs and

Table 1. Scenario method in the current study

Steps [14]	**Current study**
1. Identify focal issue	How would OSSw adoption increase in the communications industry?
2. Identify key forces in the environment	Current state of OSSw adoption and business trends in the communication industry were examined through expert interviews.
3. Rank factors by importance and uncertainty	A baseline scenario depicting the likely development path was established in a workshop. Based on the interviews, authors identified the main trends and uncertainties affecting the OSSw adoption in the vertical software market.
4. Select scenario logics	Alternative development paths were elaborated utilizing GBN technique.
5. Flesh out scenarios	Scenario descriptions were written by the authors, based on the baseline and alternative developments.
6. Analyzing implications of scenarios	Not applied in this study so far. Learning scenarios were introduced to workshop participants as a tool for further work.
7. Selection of indicators and signposts	Not applied in this study so far.

requirements of the specific market. Assessment of possibly co-existing scenarios consequently makes available a tool for anticipating the developing needs and thus aids in technology forecasting and roadmapping.

Scenario elaboration includes a number of identifiable tasks relating to the generation of ideas and gathering of data, integrating the ideas, and checking the consistency of scenarios [13]. The main steps included in the scenario method by Schwartz [14, p. 241] were adapted for the current scenario elaboration. Table 1 below introduced the tasks and the outcomes in this study.

We first conducted a set of interviews with the objective of assembling information on the CSPs' operating environment and OSSw adoption among them. Here, interviewees represented a mix of large software vendors (4 interviews) and communication service providers (6 interviews). In the semi-structured interviews, these industry experts were asked to describe their business focus, critical capabilities, software acquisition strategies employed by CSPs and reasons for usage or non-usage of OSSw. The initial findings from the interviews were documented and organized to facilitate the current state and trend analysis.

We then arranged a workshop among industry experts in order to establish a baseline scenario with the main drivers and barriers for OSSw adoption therein. Representatives from three software companies and two research organizations participated in the workshop. The baseline scenario was created applying a morphological analysis using a "future table", in which the columns represent the dimensions of uncertainty and each dimension may include several alternative future states [15, 16]. The scenario logic is identified by choosing one alternative state from each column. This technique is efficient in capturing several future attributes concurrently, and practical in cases where a group of domain experts is available.

Although scenarios usually realize as simple and apprehensible narratives, there are multiple means of creating good and internally consistent scenarios. Bishop et al. [17] examined an extensive set of the techniques for developing scenarios. They found the techniques to vary in complexity and rigor, from judgmental techniques relying solely on the individual describing the future to cross-impact analyses for calculating relative probabilities for future events. To identify sources of uncertainty as the basis for alternative futures, forecasting techniques, the GBN technique, and morphological analysis can be used. In the GBN technique, a matrix of dimensions is created and the combinations of the dimensions incorporate the scenario logic [12]. It is common that the dimensions include the current trends and anticipated future developments.

To elicit the scenarios, we applied a deductive GBN approach, in which the trends and uncertainties were prioritized to form scenarios, each combining the extremes of the two uncertainties. By comparing and contrasting the main drivers for the CSPs and software vendors as identified in the two rounds of data gathering, the forces were reduced into two dimensions. First, the CSPs have traditionally been technology driven, focusing on building and operating the communications networks with technology from vendors. Here, the change may be towards strategies with service and business innovations, as the technologies are becoming commoditized and it is no longer the speed and extent of deploying new technology that determines the winning CSPs. This forms our *nature of competences* dimension ranging from capabilities to produce and benefit from technical product innovations to capabilities to produce and benefit from business and service innovations of non-technical nature. Second, the communications software market was born as CSPs started to outsource their software development in the 1990s, and outsourcing had increased ever since [1, 2]. However we now see a change in this trend and instead, in the near future, some CSPs may on the contrary increase in-house development for flexibility. This uncertainty affects both CSPs and software vendors serving them, and forms our *technology sourcing* dimension.

Any scenario consists of three elements [18]: interpretation of current events and their propagation into the future, internally consistent description of future developments and description of future end state. In our approach, interviews and the workshop provided insight on the current state and on the likely developments on OSSw adoption in the communications industry. Further, the identified dimensions facilitated elaboration of alternative developments, based on combination of the extremes of the dimensions. Accordingly, after the dimensions were identified, we focused on creating descriptions of the future developments and arriving at prospective end state based on causal logic from the present to the future. Combining the extremes resulted in four consequences, which were written into a form of learning scenarios [19]. Learning scenarios were later introduced to the workshop participants and serve as inputs for further company-specific examinations.

3 Open Source Adoption in the Communications Software Industry

In order to examine the current state of OSSw adoption in the communications industry, a set of thematic interviews was conducted. The interviews were obtained according to judgment sampling and included two large Chinese CSPs and four European

Table 2. Overview of the respondents' position and their companies

Respondent(s) role	Company
CEO	Small regional CSP in Europe (later CSP1)
IT manager	Affiliate of global CSP in Europe (CSP2)
Director of R&D	Affiliate of global CSP in Europe (CSP3)
IT manager	National incumbent CSP in Europe (CSP4)
IT manager	Provincial branch of national CSP in China (CSP5)
Business manager	Provincial branch of national CSP in China (CSP6)
R&D managers, Account manager	Global telecom software vendor (Vendor1)
Account manager	Global system integrator (Vendor2)

CSPs of different sizes in order to enable a comparison of mature and developing CSP markets. In addition, two large software vendors operating in a global scale were interviewed. The interviewees had varying positions in their organizations. However, they were chosen so as to have a good view of how software systems are acquired and deployed. Table 2 provides an overview on the positions in their organization and the types of companies they represent.

The interviews were semi-structured. Each interviewee was presented with essentially the same questions, with certain differences in wording depending on whether the interviewee represented a CSP or a software vendor. The questionnaire applied had both fixed and open-ended questions to gain a deeper understanding on the adoption of OSSw. The questionnaire was organized around three themes: operating environment referring to both the communications industry and the communications software market, software acquisition strategies by the CSPs, and adoption of OSSw. In the following, the analysis on the latter topic is reported, while the results from the two former are included in the scenario elaboration. Under the adoption theme, the volume, the kind of and reasons for OSSw usage were discussed. Specifically, the respondents were asked: i) "Are you using OSSw and in which systems?" ii) "Why have you chosen or not chosen to use OSSw?" and iii) "What are the risks of OSSw?" The interviews were analyzed following the thematic analysis principles as proposed by Aronson [20]. The key findings were as follows.

Generally, the respondents commented that perception and attitude towards OSSw is getting more positive and that the OSSw alternative is treated with same principles and analyzed against similar criteria as proprietary alternatives when acquiring software systems (indicated by CSP3 and CSP4). Nevertheless, the current usage of OSSw among CSPs could be described as moderate or even as low. The OSSw alternatives are used mainly in infrastructure software, including Linux, Apache, and MySQL. Comparing the circumstances in European and Chinese markets, there is no notable difference in the adoption of OSSw by CSPs. In both markets, the development and deployment of software systems is occurring in fixed co-operation with vendors. This signifies that OSSw will likely be deployed in case the vendor provides it as part of their offering.

We found that only a few applications specific to this industry exist, namely in service monitoring tools and in voice communication servers built on IP networks

(Vendor 2). Also, the service providers are cautious in using OSSw components in systems visible to masses of subscribers and in systems that are otherwise critical to the business, i.e. in systems that require carrier-grade quality and performance. In such systems, the proprietary alternative is often preferred (CSP4).

The main reasons cited for opting for proprietary software systems were the lack of capabilities, the required support services, and the perceived legal and business risks (CSPs 1–6). Especially the European CSPs informed that they had outsourced most of their software-related activities. This has led to the organizations not being able to maintain internal capabilities on OSSw. If CSPs deploy OSSw, they will need to contract comprehensive support services from the vendor (CSP2). In fact, also the vendors are forced to do the same. As a result, a major share of the cost advantage of OSSw is lost. One of the CSP interviewees also brought up the of overall cost structure of software systems: a large part of the total costs incurs from deployment, integration, maintenance and operating, and the share of software licensing fees is only about 20 percent (CSP3). As the CSPs review the overall costs, the zero licensing cost of OSSw does not bring a weighty cost advantage. Instead, according several interviewees, the other contemporary phenomenon, cloud computing, may generate more measurable and wide-ranging benefits and thus CSPs are currently more focused on those. In addition, there are risks related to OSSw, which may cause CSPs to favor proprietary software: the interviewees specifically mentioned the difficulty of understanding OSSw licenses, continuity of OSSw businesses, and control over the OSSw communities.

4 Trends in the Communications Software Market

As part of the scenario elaboration, a workshop was organized with the aim of identifying the main trends in the communications industry and the communications software market as well as the main drivers behind OSSw adoption by CSPs and OSSw offerings by vendors. Managers from three software companies serving the communications industry (total of 5 persons) and scholars studying OSSw from two universities (total of 4 persons) were invited to a one-day workshop. In the workshop, the researchers presented the observations made based on the interviews and together with managers from software companies a business-as-usual scenario was constructed. The results of the workshop and, thus, the main trends can be summarized as follows.

The communications industry has been consolidating mainly through mergers of established CSPs. Simultaneously, new entrants from IT and software industries (e.g. Google and Amazon) have penetrated the market, along with virtual network operators and smaller ventures competing with innovative business models (e.g. Blyk before its transformation into a vendor). As a consequence of declining revenues from the traditional operating business, especially in the mature markets, the CSPs' business focus is partly on operational efficiency and partly on efforts to generate new revenues through introduction of new services. On the other hand, traditional MNOs are still committed to building and developing their basic network infrastructure, as mobile Internet is increasing the demand for capacity.

With regards to software systems, a clear trend has been to outsource development, deployment and operating of systems. The network element manufactures have a strong position in selling software closer to the network interfaces. However, the market for these operations support systems is slowly declining and commoditising. For business support systems, closer to the customer interface, where CSPs manage sales and billing processes, new providers have appeared to compete with the existing vendors. Thereby, analogously to the host industry, the communications software market is affected by imminent price competition leading to cost pressures. As a result, the vendors are striving to keep their positions and searching for new ventures in service-based businesses. To strengthen their positions, network element manufacturers and software vendors are engaging in long-term managed service contracts with the CSPs. New sources of revenues are being sought from cloud computing and related services.

The workshop agreed on the cost efficiency as the main driver for adoption and provisioning of OSSw. End-user innovation, another often quoted benefit of OSSw, did not come up as a driver in the discussion. Primary barriers for adoption were thought to be the perceived inferior quality of OSSw compared to proprietary alternatives – even if generally OSSw has been demonstrated to be of relatively high quality due to testing by the masses, in the communications software market OSSw is usually not considered carrier-grade, probably explained by the lack of those masses of testers and users (CSPs) – as well as control costs and risks. The potential liabilities for misusing the source code was seen as a secondary barrier.

From the interviews, several different alternative developments or pairs of opposite forces emerged, which could each have a potential impact on the adoption of OSSw in the communications software market. Out of these we identified the following two as most significant and took them as the basis of our scenario creation: *technology sourcing* and the *nature of competences* (whether technical or business-oriented).

Below we first define the concepts using extant literature and also consider how they manifest themselves in our data. Depending on the how the actors in the communications software market organize their sourcing and competences development, they may become either active or passive in employing and developing software technologies. Both circumstances have further effects on the future of OSSw in the market.

4.1 Technology Sourcing

Dibbern et al. [21] use the term *technology sourcing* to refer to an organization's arrangements regarding which parts of software development, deployment, operating and maintenance are executed with internal resources and which parts are produced by external providers. It thus designates the decision to either *insource* or *outsource* tasks and responsibilities related to software systems. Outsourcing decisions and antecedents have been studied extensively in the context of information systems. The reasons for outsourcing include cost advantages [22, 23], providers' special capabilities and resources [24, 25] and commodification of certain types of systems [26]. Alternatively, producing the IS function with the organization's own resources has been explained by asset specificity and by control costs and risks [24, 25].

Certain issues regarding outsourcing were revealed by the interviews with domain experts. Both in mature and developing parts of the communications industry, outsourcing of activities related to software systems has increased over time. In the mature parts, the main reason for contracting out is the aim to lower operational costs. In the developing parts, CSPs instead need the capabilities of the vendors as the internal competences are missing. However, some of the interviewees expresses that their organization seeks to increase internal development. The motivations included a need to increase flexibility, avoiding dependency on the vendors and ensuring the quality of the software being deployed.

One of the interviewees, a representative of a CSP in Europe (CSP3), described the ramifications of cost-pressures to software quality as described in Figure 1. By outsourcing software activities, the CSP intended to lower costs by a certain amount. The software vendor obviously must produce the services at an even lower cost than what the price paid by the CSP is. The CSP's costs will in fact increase in the beginning when outsourcing is commenced due to the structural change and other transaction costs of one-time nature.

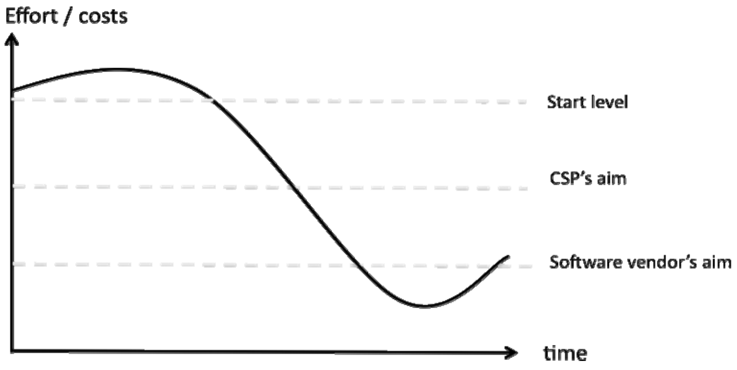


Fig. 1. Cost pressures in software outsourcing. The curve represents the CSP's total cost over time from commencement of the outsourcing arrangement.

If this comes as surprise to the CSP, the CSP may expect the software vendor to compensate for this by offering the services at an even lower price, i.e. with even less effort than what the original aim was. According to the interviewee, these cost pressures influence the quality of the software negatively. The CSP may thus lose in competitiveness due to substandard software and services, whether it shows as more limited functionality, a less faultless implementation or poorer performance. CSPs may therefore make a more prudent evaluation on which activities are outsourced and rather insource those that they see critical for keeping customers and creating new competitive benefits.

4.2 Nature of Competences

Hamel and Prahalad [27] define company's *core competence* as a set of skills and technologies that enable a company to provide a particular benefit to customers. Further, customer value can be achieved through product innovations or business innovations, which again reflect the skills and orientation of the provider. According to Kampas [28], *product innovations* materialize as a result of breakthrough engineering, whereas *business innovations* are grounded in enhancing processes and deploying commodity-marketing strategies. He thus suggests that these types of innovations are applicable in different phases of the technological development. Once a dominant design emerges in any technology domain, commodification begins and companies need to innovate with complementary assets such as marketing, efficient operations, after-sales support etc. in order to be competitive [29]. Focus moves from product innovation to business innovation. Whether competencies are generated within the company or by combining capabilities of multiple vendors in the supply chain is considered in the established theories of the management discipline. According to the resource based view [30] a competence can be achieved using internal resources that are "valuable, rare, difficult to imitate and non-substitutable". As an alternative, the dynamic capabilities framework [31] suggests that competences are created by developing an ability to "integrate, build and reconfigure internal and external competences to address rapidly changing environments."

In our data, some of the CSP respondents stressed the skills and capabilities related to different network technologies, while others emphasized understanding customer needs and customizing technologies to match the present needs. Adding capabilities for providing new services was also mentioned. Technological competences were considered important by the representatives of the European CSPs (CSP 2–4). Customization and innovations were deemed equally important by both the European and Chinese CSPs.

It is consequently natural to observe the CSPs' attempts to create competences through both product innovations and business innovations: in the future certain CSPs may build their competences by developing network assets, technologies and internal resources. These companies may seem inactive, but may maintain their positions by offering network capacity efficiently. Also, certain CSPs may instead choose to compete with contemporary and diligently segmented services built on existing infrastructure.

A CSP's total revenue can be formulated as (number of users) x (average revenue per user, or ARPU). Improving ARPU through new services is particularly of importance to CSPs operating in mature markets, given that gaining new customers is very difficult. Until recently, the CSPs have focused on the media and entertainment services, but CSPs are also well positioned to act as channels for delivering software vendors' offerings to end customers. For instance, Lucas [32] suggests the following to be strengths for CSPs in the emerging software as a service (SaaS) market: Network assets, expertise in service assurance and billing, and marketing channels. The interviews revealed that the CSPs are interested in taking their position in the SaaS value chain as aggregators of services. A typical way is to package the various SaaS services (computing and storage, e-mails, entertainment, photos, and business applications like CRM, financial management, etc.) and to automate the access to them via a portal. The end user gets all the services via a single access and is billed on a single bill. Strategic competence lies in brand creation, marketing and packaging. Such bundling is used to increase ARPU and decreases churn.

5 Four Scenarios on Open Source Adoption

In this section the two dimensions presented in the previous section are used to develop future scenarios. The scenarios illustrate how the adoption of OSSw in the communications industry might increase in the future. As described above, the GBN

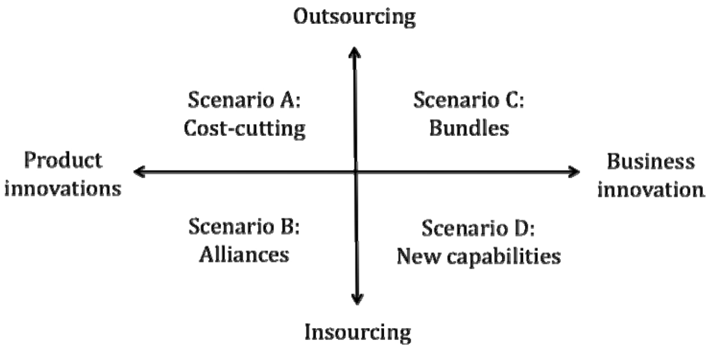


Fig. 2. The four scenarios of communications software market development

approach was applied to prioritize industry trends and to form four scenarios, each combining the extremes of the two uncertainties. Figure 2 provides an overview of the four different scenarios.

The baseline scenario A, which was established in the workshop, signifies the current trends and development path. While the baseline scenario itself may have interesting outcomes, some of the communication service providers may also aim at, for instance, strengthening their position in their industry by increasing internal software development and by developing new capabilities. What follows is that the development path and outcomes of scenario D also become possible, simultaneously with the baseline scenario. Thus, in the future we might see alternative and coexisting developments, which all have potential to increase adoption of OSSw in the domain. Table 3 below summarizes the scenario logics in terms of factors affecting the communications industry and communications software market, dimensions of uncertainties, drivers for OSSw adoption and the outcomes of developments.

Scenario A: Cost-cutting. This scenario assumes a continuous increase of outsourcing with simultaneously occurring product innovation (new technical services, such as a location-based application for smartphones). This scenario might occur as a result of consolidation in both the host industry and the communications software market. Consolidation enables economies of scale, and both CSPs and software vendors are focusing on lowering operational expenditures. It is characteristic to both kinds of actors to concentrate on developing and deploying the network technologies. In this scenario, the main driver for OSSw adoption is cost efficiency, although OSSw is mainly used in infrastructure software. Utilization of OSSw mainly takes place on the side of the software vendors', who include it as part of their offering. However, aggressive price competition may lead to an unexpected outcome, in which the player suffering the most engages in a loss-leader strategy and releases its business support system as OSSw. The intention of the vendor is to retain its existing customers, possibly create new revenues from complementary services, and to avoid competition with new entrants. In case the actor is large enough or it succeeds in obtaining more customers with this strategy, other actors in the communications software market may be forced to follow.

Scenario B: Alliances. This scenario is based on similar premises with regards to the operating environment as the previous scenario. The activities of both the CSPs

Table 3. Summary of the scenario logics

	Scenario A: Cost-cutting	**Scenario B: Alliances**	**Scenario C: Bundle**	**Scenario D: New capabilities**
Factors in communications industry (CSPs side)				
- structure	Consolidated	Consolidated	Diversificated	Diversificated
- focus	Decrease costs	Decrease costs	Increase ARPU	Increase ARPU
- capabilities	Operational efficiency	Operational efficiency	New services development	New services development
Factors in the communications software market (Vendors' side)				
- structure	Consolidated	Consolidated	Diversificated	Consolidated
- focus	Decrease costs	Protect market share	Service-based business	Decrease costs
- capabilities	New technol. development	New services development	New services development	New technol. development
Technology sourcing	Outsourcing increases	Insourcing increases	Outsourcing increases	Insourcing increases
Nature of competences	Product innovations	Product innovations	Business innovations	Business innovations
Driver for OSSW adoption	Costs	Costs	New services	Flexibility and new services
Patronages of OSSw devel.	Software vendors	Both	Both	CSPs
Uses of OSSw	Infrastructure software	Vertical software	Used in several layers	Used in several layers
Outcomes	Loss-leader dumps BSS	Common platform devel.	Adopt Cloud OSSw	CSPs engage in communities

and the vendors are focused on technological development and both are affected by the cost pressures of commodified older services. However, in contrast to the first scenario, operators become more active by increasing internal software development and related capability in order to benefit from the available standard solutions and OSSw – or models resembling it. To match the competition and to secure their market share, one of the software vendors will attempt to gather its customers into an alliance, in which an application based on the vendor's source code is being developed in co-operation with the CSPs. Such a mode of collaboration is described under the term Client-Shared Source [33]. It refers to an arrangement where the customer pays the vendor to participate in a restricted community, and is granted a license and access to the common source code repository. In this arrangement, both customers and vendor(s) contribute to the development. Such a mode is beneficial for the vendor in that the source code is not accessible to its competitors, the vendor may build its business model based on managed services or alike, and in that the vendor may maintain its lock-in to the customers.

Scenario C: Bundles. This scenario is a combination of outsourcing and business innovating trends. In it the service providers will evolve into to two basic types: ones focusing on providing network capacity ("bitpipe operators") and ones exploiting their brand to sell value-adding services on top of their existing infrastructure. Technological (i.e. product) innovation is much less important than the way the commercial offering is formulated, priced, and marketed (the business innovation). Those CSPs profiling them with such services will organize their service delivery platforms so that third parties, software vendors for instance, may utilize the platform in a flexible and agile manner. In this scenario, in line with the current trend, the CSPs outsource most of their software development activities, which means it is up to the network element manufacturers and software vendors to supplying necessary incremental improvements to the CSP's system infrastructure. *Cloud computing* technologies will be in a central role in providing new services efficiently. In cloud computing, mature and widely-adopted OSSw solutions already exist. Present considerable projects include e.g. Eucalyptus, a software platform for the implementation of private cloud computing, and Hadoop enabling the creation of data-intensive distributed applications. In this scenario, the adoption of OSSw in the communications industry is consequently associated with the developments of cloud computing. At a high adoption rate, OSSw solutions may become the dominant design in cloud technologies.

Scenario D: New capabilities. In this scenario vertical integration and divergence in the communications industry increase. Owing to the need to improve flexibility, speed to market and the quality of services, some CSPs increase their competences and capabilities in software development. Decreasing time-to-market on new services enables CSPs to monetize the current needs of their customers. Development of network technologies and IT infrastructure is left for the existing vendors to take care of. Main drivers of OSSw adoption in this development are the modifiability of standard OSSw components to specific needs and the access to a pool of resources and knowledge of the OSSw communities. As an outcome of the development, certain CSPs will become patronages of selected communities, they actively engage in these communities, and OSSw will be used in value-adding services offered to both business customers and consumers.

6 Discussion and Further Research

This paper has introduced four future scenarios for the communications industry and assessed the role of OSSw in each. We argue that the adoption of OSSw cannot be explained only through analysis on drivers and barriers of adoption. In contrast to the current studies on adoption of OSSw, this paper examines the operating environment that is influenced by several trends and forces, some of which are opposing each other. Above we have implicitly suggested that either the communication software market continues to follow the current development path with emphasis on new technology and outsourcing, or discontinuity transpires in two possible ways. First, diversification of both CSPs and software vendors may result in coexisting developments and outcomes, where e.g. certain CSPs insource in software-related activities, while others continue outsourcing these activities. Such diversity would allow software vendors to employ different technology strategies and business models, also ones incorporating OSSw as part of the offering. Second, we might also see radical change in the communications industry (e.g. widespread adoption of cloud computing technologies and service-orientation), which would further affect the whole vertical software market and its developments.

Making predictions about OSSw developments or any future events is uncertain and difficult. The scenario approach does not forecast the future; rather it can be seen as a tool for elaborating plausible future situations [14, 19]. The developments are not necessarily mutually exclusive at the industry level; they might turn out co-existing with different players evolving in different directions. This suggests that this scenario study complements the analysis on the current state. In addition, the approach produces information, which helps practitioners to identify development paths, to focus their R&D efforts and even to influence the development towards desired direction [13]. The strength of the scenario approach is in the information the interviewed domain experts can provide. However, as with all studies collecting qualitative data, there are downsides in accuracy and questions whether propositions can be generalized. While the workshop was used to elaborated baseline scenario and seek confirmation on the identified trends of the communications software market, we recognize that further interviews with domain experts might have revealed further trends. Further studies on the domain could therefore select different dimensions for the analysis and assess further development paths and outcomes.

It is also recognized that the quality of the data might have been affected by the unwillingness of some experts to share their information on current state and likely developments. In assessing the scenarios it also should be remembered that the scenarios are based more on the researchers' views and interpretation than on the experts' views. Further, it is noteworthy that the analysis may have revealed possible development paths, but authors' conclusions on outcomes may not materialize. A further limitation of the study is that we have only addressed the traditional telecommunications industry directly, not the whole communications industry. However, insofar as the landscape is changing with forces external to them, this should be felt by the traditional CSPs and their software vendors at least indirectly.

The exploratory approach, which applies adapted scenario method, facilitated examination on how OSSw is currently used and how adoption of OSSw could increase. The results of the analysis suggest that OSSw will mainly be adopted in infrastructure software. In addition, the use of OSSw in new services and as basis for applying cloud computing technologies seem both plausible. Alternative developments include an increase in OSSw supply by unusual business models (loss-leader strategy) and by alliances of software vendors and communication service providers.

The presented approach is relevant particularly for technology managers as a tool to examine uncertainties and expand thinking on the technology strategies. Moreover, the scenario elaboration is useful in examining the interplay of multiple factors affecting the environment and developments. It is therefore a viable tool for researchers conducting exploratory research in the domain. In further research, it would be interesting to feed the scenarios back to the experts, to further discuss the likelihood and impact of the scenarios to their business, and refine the scenario based on experts' feedback. What was also left for further studies was the creation of indicators enabling the technology managers to observe and project the development paths in this domain.

References

1. Tyrväinen, P., Warsta, J., Seppänen, V.: Evolution of Secondary Software Businesses: Understanding Industry Dynamics. In: León, G., Bernardos, A., Casar, J., Kautz, K., DeGross (eds.) IFIP International Federation for Information Processing, Open IT-Based Innovation: Moving Towards Cooperative IT Transfer and Knowledge Diffusion, vol. 287, pp. 381–401. Springer, Heidelberg (2008)

2. Frank, L., Luoma, E.: Future Issues in a Software Market: A Delphi Study on the Telecommunications Industry. In: 8th Conference on Telecom, Internet & Media Techno-Economics (2009)
3. Sen, R.: A Strategic Analysis of Competition Between Open Source and Proprietary Software. Journal of Management Information Systems 24, 233–257 (2007)
4. Von Krogh, G., Von Hippel, E.: The Promise of Research on Open Source Software. Management Science 52(7), 975–983 (2006)
5. Fitzgerald, B.: The Transformation of Open Source Software. MIS Quarterly 30(3), 587–598 (2006)
6. Glynn, E., Fitzgerald, B., Exton, C.: Commercial Adoption of Open Source Software: An Empirical Study. In: International Symposium on Empirical Software Engineering. IEEE, Los Alamitos (2005)
7. Nagy, D., Yassin, A.M., Bhattacherjee, A.: Organizational adoption of open source software: barriers and remedies. Communications of the ACM 53(3), 148–151 (2010)
8. Ven, K., Verelst, J., And Mannaert, H.: Should You Adopt Open Source Software? IEEE Software 25(3), 54–59 (2008)
9. Ogilvy, J., Schwartz, P.: Plotting Your Scenarios. GBN Global Business Network (2004)
10. Morgan, L., Finnegan, P.: Open Innovation in Secondary Software Firms: An Exploration of Managers' Perceptions of Open Source Software. DATABASE for Advances in Information Systems 41(1), 76–95 (2010)
11. Ågerfalk, P.J., Deverell, A., Fitzgerald, B., Morgan, L.: Assessing the Role of Open Source Software in the European Secondary Software Sector: A Voice from Industry. In: First International Conference on Open Source Systems. Springer, Heidelberg (2005)
12. Ogilvy, J., Schwartz, P.: Plotting Your Scenarios. GBN Global Business Network (2004)
13. Börjeson, L., Höjer, M., Dreborg, K.-H., Ekvall, T., Finnveden, G.: Towards a user's guide to scenarios - a report on scenario types and scenario techniques. Royal Institute of Technology (2005)
14. Schwartz, P.: The Art of the Long View: Planning for the Future in an Uncertain World. Currency Doubleday, New York (1991)
15. Rhyne, R.: Whole-pattern futures projection, using field anomaly relaxation. Technological Forecasting and Social Change 19, 331–360 (1981)
16. Godet, M.: The Art of Scenarios and Strategic Planning: Tools and Pitfalls. Technological Forecasting and Social Change 65, 3–22 (2000)
17. Bishop, P., Hines, A., Collins, T.: The current state of scenario development: an overview of techniques. Foresight 9(1), 5–25 (2007)
18. Burt, G., Chermack, T.J.: Learning With Scenarios: Summary and Critical Issues. Advances in Developing Human Resources 10(2), 285–295 (2008)
19. Schoemaker, P.J.H.: Multiple scenario development: Its conceptual and behavioral foundation. Strategic Management Journal 14(3), 193–213 (1993)
20. Aronson, J.: A Pragmatic View of Thematic Analysis. The Qualitative Report 2 (1994)
21. Dibbern, J., Goles, T., Hirschheim, R., Jayatilaka, B.: Information Systems Outsourcing: A Survey and Analysis of the Literature. ACM SIGMIS Database 35(4), 6–102 (2004)
22. Ang, S., Cummings, L.L.: Strategic Response to Institutional Influences on Information Systems Outsourcing. Organization Science 8(3), 235–256 (1997)
23. McLellan, K.L., Marcolin, B.L., Beamish, P.W.: Financial and Strategic Motivations Behind IS Outsourcing. Journal of Information Technology 10, 299–321 (1995)

24. Poppo, L., Zenger, T.: Testing Alternative Theories of the Firm: Transaction Cost, Knowledge- Based, and Measurement Explanations for Make-or- Buy Decisions in Information Services. Strategic Management Journal 19, 853–877 (1998)
25. Loh, L.: An Organizational-Economic Blueprint for Information Technology Outsourcing: Concepts and Evidence. In: 15th International Conference on Information Systems, pp. 73–89 (1994)
26. Nelson, P., Richmond, W., Seidmann, A.: Two dimensions of software acquisition. Communications of the ACM 39(7), 29–35 (1996)
27. Hamel, G., Prahalad, C.K.: Competing for the Future. Harvard Business School Press (1994)
28. Kampas, P.J.: Shifting Cultural Gears in Technology-driven Industries. MIT Sloan Management Review, 41–48 (winter 2003)
29. Teece, D.J.: Profiting from Technological Innovation: Implications for Integration, Collaboration, Licensing and Public Policy. Research Policy (15), 285–305 (1986)
30. Wernerfelt, B.: A resource-based view of the firm. Strategic Management Journal 5, 171–180 (1984)
31. Teece, D.J., Pisano, G., Shuen, A.: Dynanic capabilities and strategic management. Strategic Management Journal 18(7), 509–533 (1997)
32. Lucas, M.: Software as a Service (SaaS) and the Telecoms, http://www.billingworld.com/articles/editorial/Editorial-Software-as-a-Service-SaaS-and.html
33. Riepula, M.: A Licensing and Business Model for Sharing Source Code with Clients - Leveraging Open Client Innovation in the Proprietary World. In: Tyrväinen, P., Jansen, S., Cusumano, M.A. (eds.) ICSOB 2010. Lecture Notes in Business Information Processing, vol. 51, pp. 13–25. Springer, Heidelberg (2010)

Improving Quality and Cost-Effectiveness in Enterprise Software Application Development: An Open, Holistic Approach for Project Monitoring and Control

Luigi Buglione[1], Ernesto Damiani[2], Fulvio Frati[2], Sergio Oltolina[1], and Gabriele Ruffatti[1]

[1] Engineering Group
Via S.Martino della Battaglia 56, 00185 Rome, Italy
{luigi.buglione,sergio.oltolin,gabriele.ruffatti}@eng.it
[2] Dipartimento di Tecnologie dell'Informazione, Università degli Studi di Milano
Via Bramante 65, 26013 Crema (CR), Italy
{ernesto.damiani,fulvio.frati}@unimi.it

Abstract. The availability of integrated software tools can help organizations to easily and quickly achieve higher maturity and capability levels in process improvement and change management initiatives, by effectively supporting easy data and information sharing. However, despite their usefulness, their implementation costs still often represent a not trivial constraint for their adoption. In order to overcome such constraints, Open Source Software (OSS) can represent the right solution. Among the plenty of OSS freely available on the Net, only a very reduced set deals with measurement and monitoring & control processes, which instead represent two core processes in well-known SPI models. This paper proposes a case study showing how to efficiently detect possible project improvements using a combination of software engineering measurement-related techniques supported by the OS platform Spago4Q, keeping the focus on the need of organizations to strengthen its historical data gathering process.

Keywords: Open Source, Project Monitoring & Control, CMMI, Process Improvement, Performance Measures, QEST, LIME.

1 Introduction

The relevance of Open Source Software (OSS) has been rapidly increasing during the last few years for several reasons, including the OSS capability of fostering the open knowledge sharing across organizations, the Total Cost of Ownership (TCO) [1] reduction and a higher Return on Investment (ROI) [2] for successfully implemented projects. Nowadays, a large amount of OSS is freely available and covers plenty of informative and business goals. However, even if major forges contain hundreds of thousands projects (e.g. SourceForge – www.sourceforge.net - has more than 260,000 projects), only a few OSS projects are listed under the "Software Development" or "Enterprise" categories.

Again, very few OSS projects deal with goal-oriented measurement gathering data directly from the organization's information systems. Most available tools adopt a traditional view on software measurement, deriving static measures from source code, such as One Point Project (http://sourceforge.net/projects/opproject/) and Open Workbench (http://sourceforge.net/projects/openworkbench/).

The aim of this paper is to present a new, more comprehensive approach to software project management, including a roadmap to set up and manage a reliable and efficient measurement framework. The paper is organized as follows: Section 2 presents the measurement techniques needed to detect, filter, organize a measurement plan and measure the overall value of a project from the viewpoints of concurrent stakeholders, namely QEST-LIME. Afterwards it introduces the open source platform Spago4Q (Spago for Quality), showing its integration and joint usage with the above-mentioned measurement framework. Then, Section 3 proposes a case study applying this measurement framework to two Italian projects. Finally, Section 4 draws our conclusions and provides information about future steps towards further improvements of quality and cost-effectiveness in enterprise software application development.

2 The Puzzle of Project Monitoring and Control

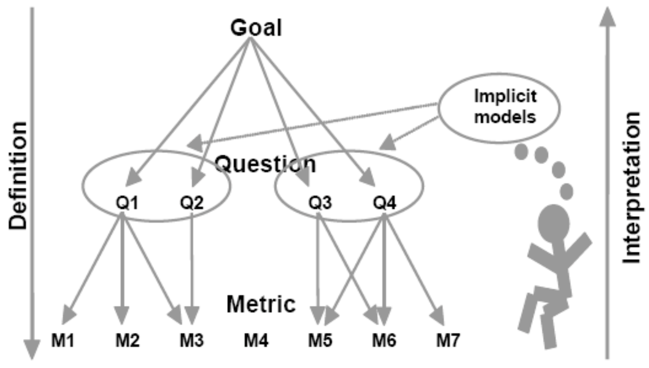
When setting up a measurement plan, organizations often erroneously choose the most adopted "standard measures", following a sort of "adoption by analogy" approach. However, in the medium term, the habit of under-analyzing the internal informative needs and the impact that measurements have on their economic situation may lead to a "*domino effect*." The reduction of the budget devoted to measurement, monitoring, and control processes usually leads to a lower level of control on the project. This effect is described in the most popular SPI process reference models, such as CMMI [4] and ISO/IEC 15504 (aka SPICE) [5]. Lack or scarcity of reliable data can make it hard to bind the mean relative error to the phenomena to be predicted. Here a sound analysis on the informative needs and the careful choice of measurements and metrics as a basic recipe for successful process monitoring is provided.

The following sub-sections include a short description of the "ingredients" for the above-mentioned recipe, focusing on the added value that an OSS suite can bring to it.

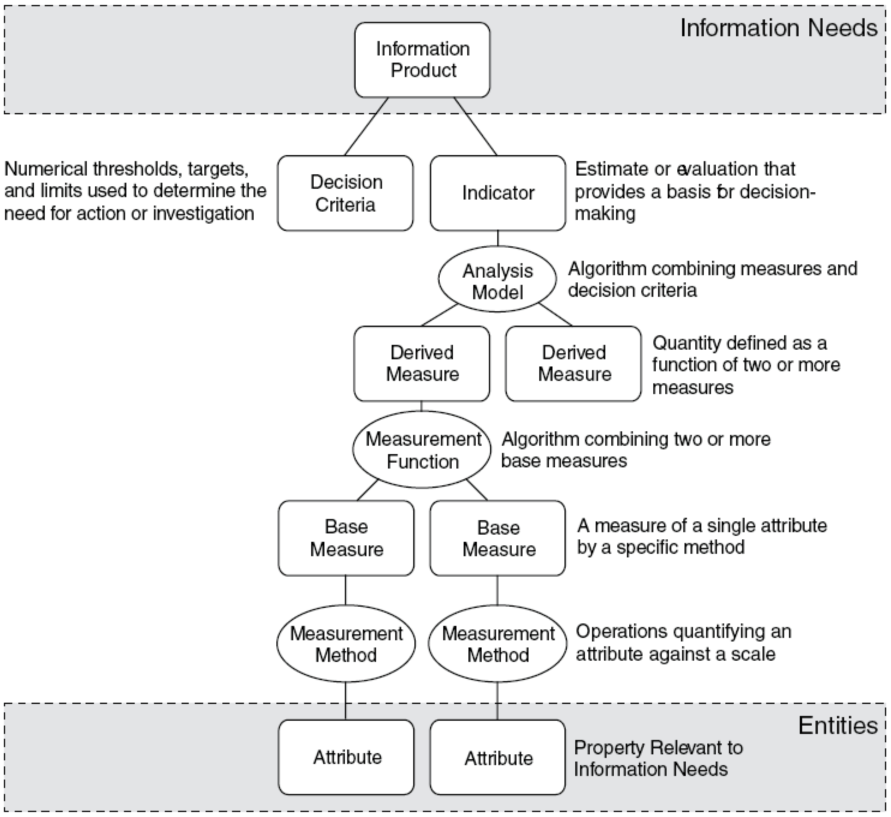
2.1 Deriving Measures from Information Needs: The GQM Paradigm

The first, common-sense technique for deriving a measure is GQM (Goal-Question-Metric), originated by Basili & Weiss in the early '80s [3] and refined during the years. Such approach uses a three-tier decomposition, deriving *measures* (M) from the related *questions* (Q) to be posed for answering the information *goals* (G) of the interested stakeholders. In fact, one of the strong points of GQM is being multi-faceted supporting multiple perspectives. Fig. 1 (a) shows GQM bi-directional information flow, from the definition of measures until the interpretation of results obtained by applying those measures.

Plenty of variants from the basic technique have been proposed, including GQ(I)M [6], GQM++ [7], and M3P [8], in the '90s and more recently V-GQM [9], MIM, Lightweight GQM [10], and GQM+Strategies [11].



(a)


Information Needs
Information Product
Decision Criteria
Numerical thresholds, targets, and limits used to determine the need for action or investigation
Indicator
Estimate or evaluation that provides a basis for decision-making
Analysis Model
Algorithm combining measures and decision criteria
Derived Measure
Derived Measure
Quantity defined as a function of two or more measures
Measurement Function
Algorithm combining two or more base measures
Base Measure
Base Measure
A measure of a single attribute by a specific method
Measurement Method
Measurement Method
Operations quantifying an attribute against a scale
Entities
Attribute
Attribute
Property Relevant to Information Needs

(b)

Fig. 1. GQM (a) and MIM (b) specifications

Using also a three-tier decomposition schema, ISO/IEC 15939 proposed the Measurement Information Model (MIM), linking the information needs to its measurable *entities* and related *attributes* (Fig. 1 (b)) [12]. MIM refines and reinforces the basic GQM idea, stressing the central role of information needs and the instrumental role of measures as a tool for addressing them [13].

2.2 Determining the Right Number of Measures: The BMP Technique

An important set of questions to be answered concerns the number of measurements to be collected and the cost of the measurement campaign.
Relevant questions include: How many measures should be adopted? Are they frequent enough/properly balanced in order to intercept the different stakeholders' goals? And how much do they cost as a percentage of the project budget? Finally, are measurement costs aligned and well integrated within the cost for the 'project management' process?

BMP (Balancing Multiple Perspective) [14] is a technique for answering these questions, using concepts taken from the popular Balanced ScoreCard (BSC) approach[1].

The four steps allowing to deploy an efficient BMP framework are:

1. Determine the dimensions of interest in the project;
2. Determine the list of the most representative measures associated with each dimension;
3. For each of the selected measures, identify which other control variables might be impacted negatively (e.g. higher quality often means greater initial costs or longer project duration);
4. Figure out the best combination of indicators and the causal relations between them in order to build a measurement plan for the project.

2.3 Determining the Performance Value: The QEST-LIME Family

The QEST-LIME models and frameworks have been designed to tackle the entire decision-making process from a multi-perspective viewpoint [15]. QEST (Quality factor + Economic, Social and Technical dimensions) is a multi-dimensional, software performance measurement model: it provides a multidimensional shell that can be filled according to the management objectives of each specific development project.

For this reason, it is often referred to as an "open model". Its basic purpose is to express performance into a unique, single value, as the combination of the specific measures (or sets of measures) selected for each of the three dimensions, these values being derived from both an instrument-based measurement of productivity and a perception-based measurement of quality.

In the last decade, the original QEST three-dimensional model has been extended to handle *n* dimensions/perspectives, using the simplex algorithm to compute the top

[1] More information and references with BMP case studies are available at the BMP webpage (www.semq.eu/leng/modtechbmp.htm).

performance point. Therefore, QEST nD can be also used as a generic n-dimensional measurement model, according to the features and advantages listed above [16]. At the same level, the LIME (LIfecycle MEasurement) model [15] represents an extension of QEST model concepts to a dynamic context.

The iterative definition, collection and analysis of multidimensional measures at each software life cycle (SLC) phase offer the feedback required to make adjustments to the project processes in a timely fashion, both for the next phase and for designing future improvements to the process of the previous phase.

2.4 Automating Project Monitoring and Control: Spago4Q

Whatever the technique used for measuring and collecting data, the evaluation must be executed in a fully transparent way, without requiring any further action by programmers and/or designers.

In relation to this, Spago4Q (www.spago4q.org) is an open source multi-process and multi-project monitoring platform. It allows to measure and monitor the quality improvement during the project lifetime, software projects development, ICT services supply and facility management, according to specific service level based on the various actors' point of views.

Spago4Q relies on a suitable meta-model, described in [17], for the definition of the process and measurement framework, supporting its customization to different process paradigms and measurement frameworks. Refer to [17][18] for a more detailed description of valuable features of the tool (e.g. supporting assessment frameworks such as CMMI and QEST). As shown in Fig. 2, Spago4Q includes multiple extractors for the tools used during the software lifecycle (development environments, text-editing tools, requirements management frameworks, etc.). Those extractors collect data directly from process work-products (e.g. java classes or logs).

Spago4Q meta-model is composed of four modules:

- the *Process* meta-model, which provides a description of the generic software development process;
- the *Measurement Framework* meta-model, used to represent specific measure models from most popular development processes;
- the *Assessment* meta-model, which allows the modeling of a generic evaluation structure;
- the *Extractors* meta-model, which formalizes and defines the extractors used to retrieve data from the process entities and supplies it to the measurement framework.

The exploitation of the platform in working environments showed that automating the gathering of measurement data and its computation considerably reduced the impact of costs in Software Process Improvement (SPI) activities.

Even if a number of open source and proprietary process monitoring tools are available, to the best of our knowledge only Spago4Q can handle multi-process, multi-project measuring. Commercial well-known tools such as Polarion (www.polarion.com) and 6th Sense (www.6thsenseanalytics.com) do not support cross-process, cross-project comparisons. The Holkar project [19] developed a prototype to monitor quality which used the XML data model for collecting data from sources and storing them in a repository. A limited set of budget performance metrics can then be applied on the XML database. The Hackystat project (http://csdl.ics.hawaii.edu/research/hackystat) provides a measurement framework for non-intrusive project metric collecting and analysis.

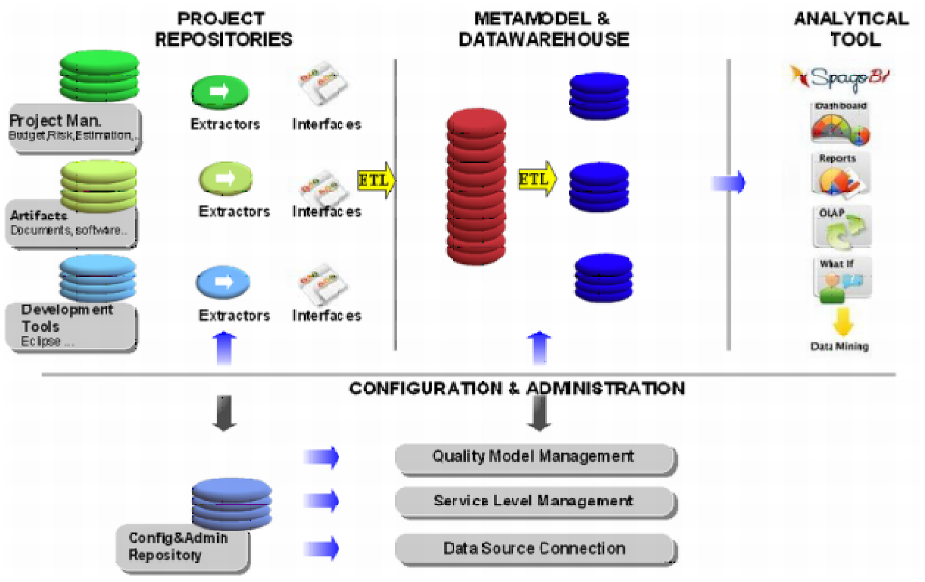


Fig. 2. Spago4Q framework

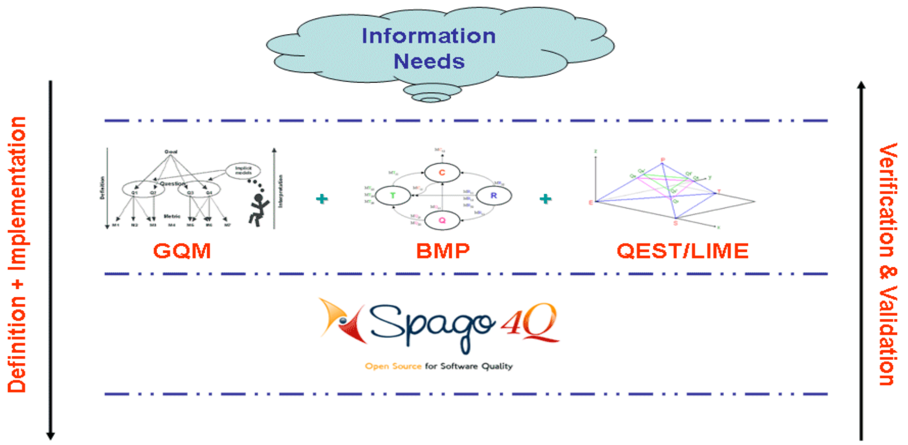

Information Needs
Definition + Implementation
Verification & Validation
GQM
BMP
QEST/LIME
Spago 4Q
Open Source for Software Quality

Fig. 3. The main flow followed for establishing a goal-oriented measurement framework

2.5 Putting All Pieces Together: The Suggested Measurement Framework

Techniques like GQM, BMP, QEST-LIME, and advanced tools like Spago4Q provide the pieces needed to solve the process monitoring puzzle described in Section 2. Each piece has its own value, but taking them singularly, they have only a limited impact. For instance, the GQM alone does not provide suggestions on how to build a measurement plan and on the proper number of measures to be adopted.

Furthermore, it covers neither the economics of monitoring & control process, nor their consolidated value for the decision-making. Therefore, our approach proposes to join such techniques and tools, keeping the best from each one and creating a simple procedure (Fig.3):

1. Define what we need to measure (using a GQM-based technique).
2. Filter and prioritize what we need to measure (using the BMP technique [4]).
3. Determine the project/activity performance values with a holistic view (using QEST-LIME), highlighting the improvement goals to manage.
4. Automate the last step with an OSS tool (e.g. Spago4Q), providing faster data gathering and – consequently – better data for the decision-making, as well as a better historical database for future estimates.

3 A Case Study

In order to validate our approach, we present a complete case study referring to monitoring two large proprietary products developed and maintained by Engineering Health Software Factory (HSF); the projects were analyzed, collecting data in the period January 2009 – October 2010.

Both are *on-going* projects deployed on customers' server, and new versions are released yearly. The first project (Product 1), deployed for 100 customers, produces a release per quarter and it is developed exploiting mainframe techniques, following the Unified Process paradigm. It is characterized by a dimension of 1100 EKLOC (Effective Thousands of Lines of Code) and a development team of 18 units. The second project (Product 2) is characterized by an average of ten releases per year, a dimension of 1600 EKLOC, and it has been deployed for 60 customers. The team who realized it is composed of 10 developers following an agile-based process.

In order to properly monitor each phase of the HSF software life cycle, we use BMP and LIME – together with the three classical QEST analysis perspectives (Economic, Social, and Technical) – as described in Section 2.

The main business goal for the *goal-oriented* analysis was to reduce the overall production cost.

Such cost is mainly driven by three factors:

1. *Lack of requirement stability*, as a source of overhead in design and implementation activities;
2. *Working group management overhead,* as a source of delays and variance in milestones;
3. *Corrective and adaptive maintenance activities.*

In particular, a relevant part of such cost was represented by the cost trend of corrective and adaptive maintenance activities, monitored by PR-MAN-E measures. These factors were identified after a thorough analysis on the process and on the analyzed projects [20] allowed to define the measurement goals shown in Table 1.

Table 2 presents the complete list of measures selected with respect to the goals. In order to reduce costs, different improvement actions were undertaken to monitor each

Table 1. Case study: Measurement goals with respect to cost factors

Cost Factors	SLC Phase	Dimension	Goal	Id.
Lack of requirement stability	Requi-rements (analysis)	E Economic	Reduce Delivery Variance	REQ-E- 1
			Reduce Requirements Variability	REQ-E-2
		S Social	Reduce criticality in working group management	REQ-S-1
			Involve users in requirements sharing and validation	REQ-S- 2
		T Technical	Compliance with pre-defined product quality levels	REQ-T- 1
Working group management overhead	Develop-ment & Design	E Economic	Reduce delivery variance	DEV-E- 1
		S Social	Reduce criticality in working group management	DEV-S- 1
		T Technical	Compliance with pre-defined product quality levels	DEV-T- 1
			Compliance with pre-defined product software quality levels	DEV-T- 2
			Process compliance	DEV-T- 3
	Test	E Economic	Reduce the # severe defect due to analysis and design phases, reducing rework costs	TES-E- 1
			Reduce Delivery Variance	TES-E- 2
		S Social	Reduce criticality in working group management	TES-S-1
		T Technical	Increase n.o. checks	TES-T- 1
			Process compliance	TES-T- 2
Corrective and adaptive maintenance activities	Running (mainte-nance) CMMI-SVC v1.3 Processes	E Economic	Reduce defects resolution costs	MAN-E- 1
		S Social	Reduce criticality in working group management	MAN-S- 1
			User satisfaction	MAN-S- 2
		T Technical	Increase # checks	MAN-T-1

Table 2. Case study: Measurement goals with respect to cost factors

Goal	Measure	CMMI-DEV v1.3 PAs	Measure Id.
REQ-E- 1	Incidence of delays on delivery milestones (deliverables) w.r.t. total # deliverables	PMC MA	PR-REQ-E-M1.1
REQ-E-2	Requirements Variability	RD REQM PMC	PR-REQ-E-M1.2
REQ-S-1	#. detected criticalities during the human resources management w.r.t. group size in the phase	PPQA PMC	PR-REQ-S-M1.1
REQ-S- 2	User satisfaction or % users involvement in the phase	all PA (GP2.7) - low correlation w.r.t. ISO 9001:2008 §5.2	PR-REQ-S-M1.2
REQ-T- 1	Document quality: respect of quality standard	PPQA	PR-REQ-T-M1.1
DEV-E- 1	Incidence of delays on delivery milestones (deliverables) w.r.t. total # deliverables	PMC MA	PR-DEV-E-M2.1
DEV-S- 1	# detected criticalities during the human resources management w.r.t. group size in the phase	PPQA PMC	PR-DEV-S-M2.1
DEV-T- 1	Document quality: respect of quality standard	PPQA	PR-DEV-T-M2.1
DEV-T- 2	Software quality: complexity, compliance, maintainability	PPQA VAL	PR-DEV-T-M2.2
DEV-T- 3	Compliance with end-phase checklist	PPQA	PR-DEV-T-M2.3
TES-E- 1	# Defects detected before System Test by PR or verifications w.r.t. code size	VER VA PMC	PR-TES-E-M3.1
TES-E- 2	Incidence of delays on delivery milestones (deliverables) w.r.t. total # deliverables	PMC MA	PR-TES-E-M3.2
TES-S-1	N.o. detected criticalities during the human resources management w.r.t. group size in the phase	PPQA PMC	PR-TES-S-M3.1
TES-T- 1	Incidence of the n.o. reviews (peer reviews or inspection reviews) w.r.t. total # deliverables	PPQA VER	PR-TES-T-M3.1
	Percentage defects distribution on phases that produced the defects (consider only analysis and design phase)	PP PMC MA	PR-TES-T-M3.2
TES-T- 2	Compliance with end-phase checklist	PPQA PMC	PR-TES-T-M3.3
MAN-E- 1	Incidence of defects tested in running and testing phase w.r.t. maintained code size (Lines of Code or Function Points)	PMC MA	PR-MAN-E-M4.1
	Mean defect resolution time w.r.t. severity during running phase	IRP	PR-MAN-E-M4.2

Table 2. (*continued*)

MAN-S- 1	N.o. detected criticalities during the human resources management w.r.t. group size in the phase	PPQA PMC	PR-MAN-S-M4.1
MAN-S- 2	User satisfaction	all PA (GP2.7) low correlation ref. ISO 9001:2008 §5.2	PR-MAN-S-M4.2
MAN-T-1	Percentage defects distribution on phases that produced the defects (consider only analysis and design phase)	PP PMC MA	PR-MAN-T-M4.1

phase. In particular, we performed the actions listed below in order to improve the overall economic results. All actions take as their input from the value of specific metrics.

Increase document quality produced in the requirements phase (monitored by the measure PR-REQ-T-M1.1).

Increase software quality to facilitate the its maintainability in the development phase (monitored by the measures PR-DEV-T-M2.1 and PR-DEV-T-M2.2).

Increase the number of reviewed deliverables (monitored by the measure PR-TES-T-M3.1). The reviews performed in the analysis and design phases had the goal to discover bugs early in the development cycle (as assessed by the measures PT-MAN-T-M4.1, PR-TES-T-M3.2).

Table 3 analyzes the results related to the goal "Reduce Defects Resolution Cost" in the Running phase. The goal has been achieved with an improvement of the 7.2% for Product 1 and 7.6% for Product 2; furthermore, the "Mean Defect Resolution Time" measure (PR-MAN-E-M4.2) improved in a significant manner. Also the Social (S) dimension, and in particular the "User Satisfaction" metric (PR-MAN-S-M4.2), received benefit from the improvements of the other phases and dimensions such as document quality improvement and the reduction of Mean Defect Resolution Time.

Table 3. Selected measures and results for the "Reduce defects resolution cost" goal

		Product 1		**Product 2**	
Goals	**Measure \ Year**	**2009**	**2010**	**2009**	**2010**
Reduce Defects resolution costs	Economical Dimension PR-MAN-E	0.7000	0.7550	0.7870	0.8530
	PR-MAN-E-M4.1	0.5244	0.5099	0.5793	0.7319
Reduce criticality in working group management	PR-MAN-E-M4.2	0.8173	0.9244	0.9262	0.9333
	Social Dimension PR-MAN-S	0.6810	0.7000	0.7100	0.6850
	PR-MAN-S-M4.1	0.6944	0.5833	0.5000	0.3000
User satisfaction	PR-MAN-S-M4.1	0.6750	0.7500	0.8000	0.8500
	Technical Dimension PR-MAN-T	0.6304	0.6552	0.7005	0.6842
Increased #. of checks/reviews	PR-MAN-T-M4.1	0.6304	0.6552	0.7005	0.6842

Table 4. Measures correlated to the "Reduce defects resolution cost" goal

		Product 1		Product 2	
Goals	**Measure \ Year**	**2009**	**2010**	**2009**	**2010**
Compliance with pre-defined documents q.l.	Requirement phase PR-REQ-T-M1.1	0.8333	0.8667	0.8667	0.9000
Compliance with pre-defined documents q.l.	Development phase PR-DEV-T-M2.1	0.8333	0.8667	0.8667	0.9000
Compliance with pre-defined software q.l.	Development phase PR-DEV-T-M2.2	0.5000	0.6000	0.7500	0.9000
Increase n.o. checks/reviews	Test phase PR-TES-T-M3.1	0.7500	0.8750	0.5250	0.6250

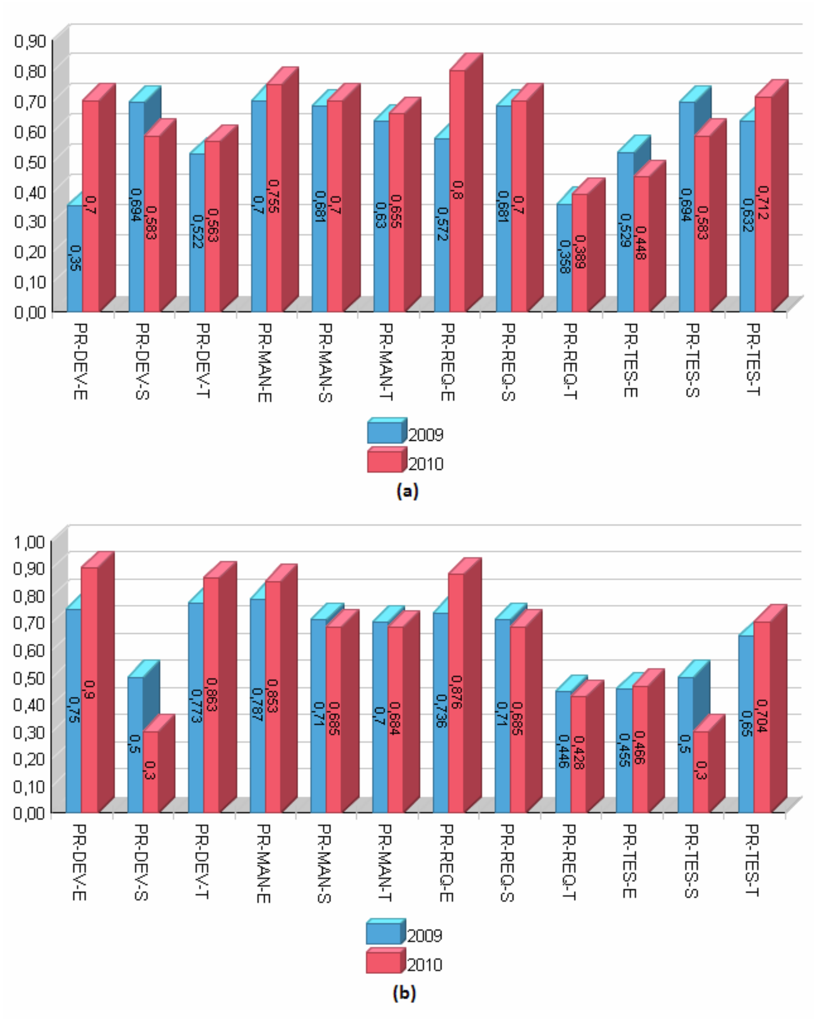
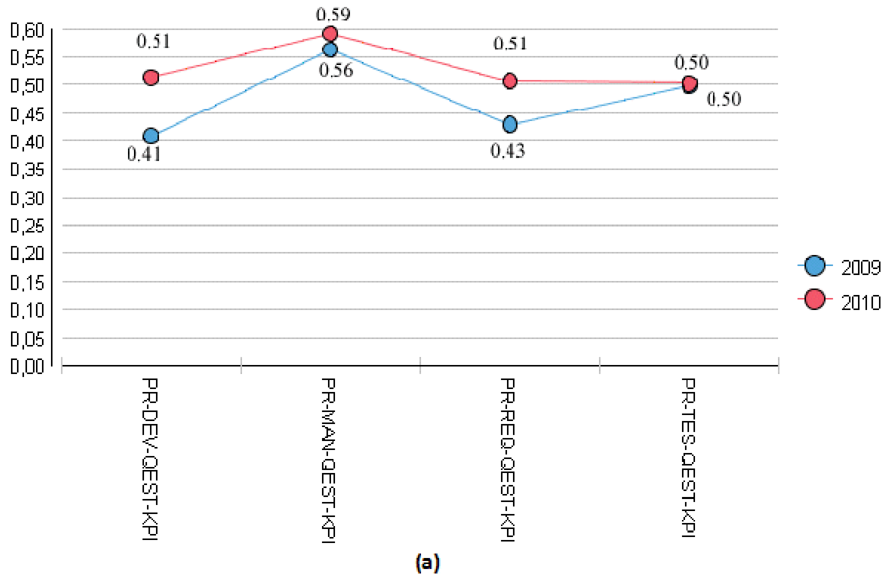


Fig. 4. Performance value by perspective for Product 1 (a) and Product 2 (b)

Table 4 shows the values of correlated measures: the increased number of reviews (PR-TES-T-M3.1) had a positive effect on the improvement of the overall quality of documents (PR-REQ-T-M1.1 and PR-DEV-T-M2.1), and software quality (PR-DEV-T-M2.2 with respect to the measure PR-MAN-T-M4.1), decreasing the percentage of defects due on initial development phases (analysis and design). Fig. 4 and 5 show charts produced by Spago4Q.



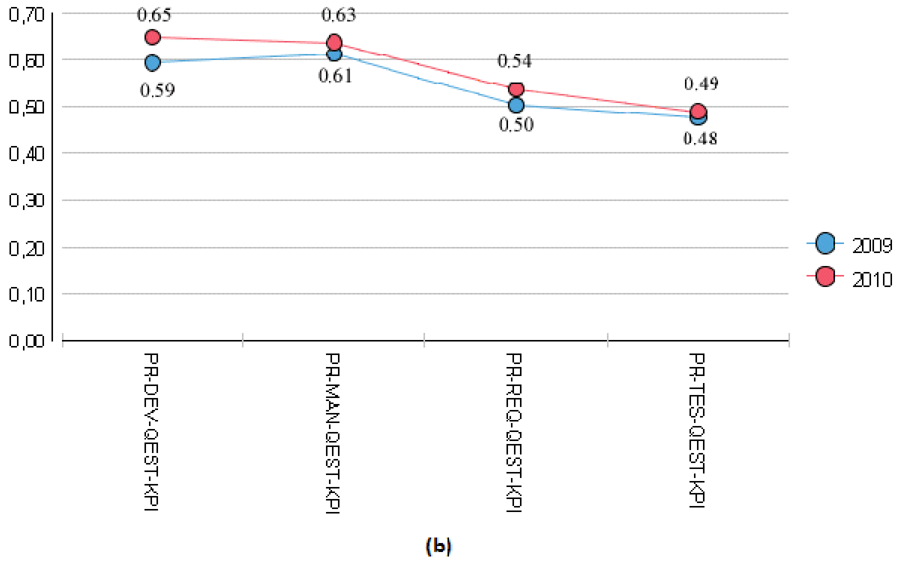


Fig. 5. Performance value for each phase, for Product 1 (a) and Product 2 (b)

These charts provide a synoptical representation of the results taken in the two periods exploiting the described monitoring framework. Performances for each dimension (Fig. 5 (a) and (b)) and phase (Fig. 6 (a) and (b)) have been, in most cases, improved. The calculation of the performance value (range value 0-1) for each dimension and phase is discussed in [15].

A further analysis on the case study results shows how the completeness and quality of the measured data are influenced by time constraints imposed by customers.

Tests showed that a strong commitment by the development team is needed to guarantee a good quality level on the released code. Moreover, the effectiveness of peer reviews in preventing errors in the development and design phase has been measured by means of the metrics PR-TES-E-M3.1, PR-TES-T-M3.1, and PR-TES-T-M3.2. To reduce peer reviews costs, a thorough analysis of criticalities in deliverables is necessary at each change request, in order to apply reviews only on such deliverables. This implies a definition of a risks map to point out the specific risk exposure of each component. Furthermore, an additional improvement to HSF development activities has been identified in the implementation of automatic functional tests to better define non-regression tests on new releases or patches, with probable benefits on maintenance costs. On the other hand, the high implementation costs of such a technique could be justified only in specific critical projects.

4 Conclusions and Next Steps

There is an increasing need to run real, valuable *governance* and not only *management* of ICT projects. Thus, a deeper attention on the measurement process is strongly needed. The importance of this factor such as a priority is also stressed in most SPI models and frameworks (e.g. CMMI, where “measurement & analysis” is a ML2 process and the quest for data is included in GP2.8).
Another major requirement is the usage of automatic tools for making faster and more accurate that allows to perform measures in a faster and more accurate way. The high costs of acquisition and maintenance of these tools can however discourage their adoption people from adopting them.

Now Open-Source Software has a higher level of affordability and reliability than in the previous years (see for instance the wide adoption of CVS tools
in Configuration Management processes). However, tools alone cannot support ICT project governance: advanced measurement techniques and models are also needed.

This paper presented an Italian experience running a measurement framework using advanced techniques together with a specific OSS toolkit, Spago4Q. The case study showed the results of the application of such a framework to two projects, presenting some snapshots of the achieved results. We claim that our approach represents a good starting point for a full implementation of a Balanced Scorecard (BSC) technique. Applying BSC to our scenario is simple but not trivial: it would require observing the number of companies failing (or having difficulties) when dealing with holistic and comprehensive governance from concurrent stakeholders’ viewpoints.

Our future works will focus on the improvement in Spago4Q reporting features (insertion of multi-dimensional representations as in the original QEST model, for making easier the interpretation of results; monitoring & control level by process from the desired process model) and the implementation of a GQM(R) matrix [21], for choosing new possible measures in order to cover a larger plateau of information needs at the same cost.

Acknowledgement

This work was partially funded by EC under the project ARISTOTELE (contract n. FP7-257886) and ASSERT4SOA (contract n. FP7-257351).

References

1. Wheeler, D.A.: Why Open Source Software / Free Software (OSS/FS)? Look at the Numbers!, `http://www.dwheeler.com/oss_fs_why.html`
2. Stefan, A.: How Software Copyright and Patent Laws Are Hurting Real Innovation. Technical Sciences and Applied Mathematics 2, 15–22 (2008)
3. Basili, V.R., Weiss, D.M.: A Methodology for Collecting Valid Software Engineering Data. IEEE Trans. on Software Engineering SE-10(6), 728–738 (1984)
4. CMMI Product Team, Capability Maturity Model for Development (CMMI-DEV) v1.3, Technical Report, CMU/SEI-2010-TR-033, Software Engineering Institute, `http://www.sei.edu.cmu/cmmi`
5. ISO/IEC, I.S.: 15504-x, - Information technology – Process assessment, Parts 1-7 (2004-2008), International Organization for Standardization (2010)
6. Park, R., Goethert, W.B., Florac, W.A.: Goal-Driven Software Measurement - A Guidebook. Software Engineering Institute, Handbook, CMU/SEI-96-HB-002 (1996)
7. Gray, A., MacDonell, S.G.: GQM++ A Full Life Cycle Framework for the Development and Implementation of Software Metrics Programs. University of Otago, New Zealand. Technical Report (1997), `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.47.9007`
8. Offen, R.J., Jeffery, R.: Establishing Software Measurement Programs. IEEE Software 14(2), 45–53 (1997)
9. Olson, T., Runeson, P.: V-GQM: a feedback Approach to Validation of a GQM study. In: 7th IEEE Symposium on Software Metrics (METRICS 2001), London, UK, pp. 236–254 (2001)
10. Gresse Von Wangenheim, C., Anacleto, A., Salviano, C.F.: MARES - A Methodology for Software Process Assessment in Small Software Companies, LQPS001.04E. Laboratório de Qualidade e Produtividade de Software, UNIVALI. Technical Report (2004)
11. Basili, V., Heidrich, J., Lindvall, M., Munch, J., Regardie, M., Trendowicz, A.: GQM+Strategies – Aligning Business Strategies with Software Measurement. In: 1st Int. Symposium on Empirical Software Engineering and Measurement (ESEM 2007), pp. 488–490 (2007)
12. ISO/IEC, IS 15939:2007 - Systems and software engineering - Measurement process, International Organization for Standardization (2007)
13. Abran, A.: Software Metrics and Software Metrology. IEEE-CS Press & John Wiley & Sons, Hoboken (2010)

14. Buglione, L., Abran, A.: Improving Measurement Plans from multiple dimensions: Exercising with Balancing Multiple Dimensions – BMP. In: 11th IEEE International Software Metrics Symposium (METRICS 2005). IEEE Press, New York (2005)
15. Buglione, L., Abran, A.: Performance calculation and estimation with QEST/LIME using ISBSG r10 data. In: 5th Software Measurement European Forum (SMEF 2008), pp. 175–192 (2008)
16. Buglione, L., Abran, A.: QEST *n*D: n-dimensional extension and generalisation of a Software Performance Measurement Model. Int. J. of Advances in Engineering Software 33(1), 1–7 (2002)
17. Colombo, A., Damiani, E., Frati, F., Oltolina, S., Reed, K., Ruffatti, G.: The use of a meta-model to support multi-project process measurement. In: 15th Asia-Pacific Software Engineering Conference (APSEC 2008), pp. 503–510 (2008)
18. Ardagna, C.A., Damiani, E., Frati, F., Oltolina, S., Regoli, M., Ruffatti, G.: Spago4Q and the QEST *n*D model: An open source solution for software performance measurement. In: Ågerfalk, P., Boldyreff, C., González-Barahona, J.M., Madey, G.R., Noll, J. (eds.) OSS 2010. IFIP Advances in Information and Communication Technology, vol. 319, pp. 1–14. Springer, Heidelberg (2010)
19. Holkar, V.: Experimental Implementation and Evaluation of Quality Management Process, Master Thesis. Department of Computer Science and Computer Engineering, La Trobe University, Australia (2007)
20. STSC Cost Analysis Group, Software Development Cost Estimating Guidebook, Software Technology Support Center. Handbook, `http://stsc.hill.af.mil/consulting/sw_estimation/softwareguidebook2010.pdf`
21. Buglione, L.: Misurare il Software 3/ed. Franco Angeli Editore, Milan (2008)

Transformations of a Solution Strategy: A Case Study

Marko Komssi[1,2], Marjo Kauppinen[1], Matti Ropponen[2], and Pirkka Palomäki[2]

[1] Aalto University School of Science and Technology, PO Box 9210, 02015 Aalto, Finland
[2] F-Secure Corporation, PO Box 24, 00181 Helsinki, Finland
{Marko.Komssi,Marjo.Kauppinen}@tkk.fi,
{Matti.Ropponen,Pirkka.Palomaki}@f-secure.com

Abstract. Fast-paced and turbulent business environments force companies to make repeated decisions concerning their solution strategy. This paper presents a retrospective case study that investigated changes in the strategy of a successful SaaS solution provided by a medium-sized Finnish software company. The study concentrated on the following research question: "How did the solution strategy evolve during the life-cycle of the software solution?" The main finding of the study is that the solution strategy has undergone four distinct stages over nine years. The stages differed in terms of whether the focus was on existing or new services offered to potential or new customers. Each of the four stages contributed to sustaining the solution's revenue growth in an increasingly competitive and maturing market. The study's findings suggest that customers' customers are crucial for growth opportunities, particularly once the original market has become mature.

Keywords: Solution Strategy, Product Life Cycle, New Service Development, Software as a Service (SaaS), Customer Relationships, Retrospective Case Study.

1 Introduction

Surviving in fast-paced and turbulent business environments is a crucial concern for high-tech companies [1]. Solution strategies are presented with critical challenges in high-tech companies, as the fast-paced environment forces repeated strategic decisions [2]. Solution strategies handle how core products and services are produced, designed, distributed, promoted, and innovated over time [3].

Product life cycles seem to be very short in software industries [4]. In a fast-paced business environment, even a successful software solution strategy soon becomes outmoded. A product (or solution) life cycle includes exploratory, growth, and mature stages, as stated by Klepper [5], who argued that each of the stages entails special characteristics. For instance, product innovation is high in the exploratory stage. The product stabilizes during the growth stage and the number of product innovations declines. Finally, in the mature stage, market shares stabilize, and management, marketing, and development techniques become more refined.

One example of emergent and dynamic concepts in software business is Software as a Service (SaaS). SaaS was introduced in the literature in the late '90s [6]. Thereafter, SaaS has been called, for instance, a delivery, business, pricing, revenue, or licensing model, as well as a demand-led paradigm [7]. The market share of SaaS solutions reached $9.6 billion in 2009 and SaaS is forecast to have a 17.7% compound annual growth rate through 2013 [8, 9]. From a technical point of view, SaaS architecture comprises valuable characteristics, such as *internationalization* and *extreme transaction processing* [10]. These characteristics allow even a relatively small software company to enter or create a mass market and serve a huge number of customers and customers' customers simultaneously. Once the software company has succeeded with its market entry, however, it must anticipate the changes in solution strategy and heavy competition in the growth and mature stages.

This paper aims to increase the understanding of the strategic changes that can make a SaaS solution successful over time in a fast-paced and unpredictable mass-consumer market. The paper presents a retrospective case study of a nine-year adoption of a SaaS model in a Finnish mid-sized software company. The study concentrated on the following research question: "How did the solution strategy evolve during the life-cycle of the software solution?" The solution investigated in this study represents one way of adopting the SaaS model. The adoption has been described previously [11]. The term 'solution' refers to a core software product and/or platform augmented by other components, such as training, manuals, support, and other services [12].

This paper is organized as follows. Section 2 presents the research background and Section 3 explains the case study. Section 4 introduces the market evolution findings and Section 5 presents the transformation of a solution strategy between the years 2001 and 2010. Section 6 discusses the findings and Section 7 concludes the paper.

2 Background

The theoretical framework of this study consists of two schemas called the *new service strategy matrix* [13] and *the three tiers of noncustomers* [14]. The first schema consists of four elements as illustrated in Table 1. Under a *new business* strategy, the company enters a new market with a new solution. A *market extension* strategy guides the company towards offering existing services to new market segments. By following a *share building* strategy, the company intends to sell more existing services to existing customers. Finally, the company endeavors to market new services to existing customers under a *line extension* strategy.

The four elements of the strategy matrix contain identifiable characteristics [13]. A new business strategy is the riskiest alternative for the company because it cannot rely on its existing competencies. The market extension strategy engages the company to pursue new market segments. The share building strategy may involve, for instance,

Table 1. A new service strategy matrix in new solution development [13]

Markets / Offering	**Existing customers**	**Potential customers**
Existing services	Share building	Market extension
New services	Line extension	New business

the aggressive pricing of solutions. The line extension strategy is common in mature industries and involves leveraging the current customer base.

The schema known as three tiers of noncustomers is presented in Figure 1. The first tier represents the potential customers who pay for an industry's offering but are not loyal to any existing solution. The second tier stands for potential customers who currently refuse to purchase the industry's offerings. The third tier represents potential customers who have never thought about the industry's offerings.

Connections can be found between the two schemas. Focusing on the first tier typically leads to account wars [14]. Therefore, the aggressive style of the share building strategy can be used in the tough competition in the first tier. In addition, a new business strategy may be required to develop a novel solution to be offered to a new market segment in the third tier.

In the literature, we found one previous work [15] that used the new service strategy matrix to illustrate how internationalization had impacted the service offered by a modern technopark operator. We also found one previous work [16] that utilized the three tiers of noncustomers to demonstrate the new market for the game consoles.

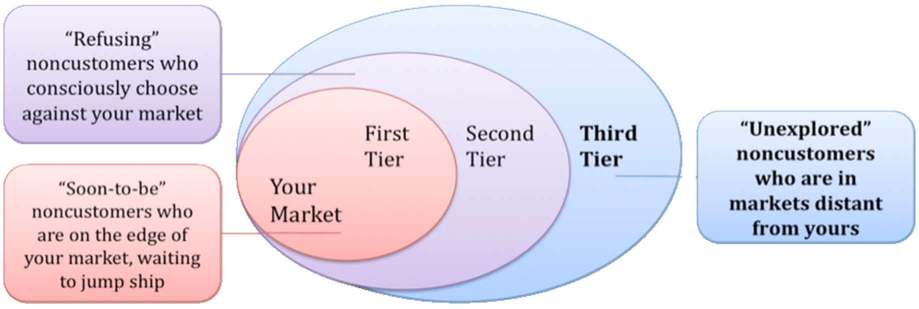

"Refusing" noncustomers who consciously choose against your market
"Soon-to-be" noncustomers who are on the edge of your market, waiting to jump ship
First Tier
Second Tier
Third Tier
Your Market
"Unexplored" noncustomers who are in markets distant from yours

Fig. 1. The schema known as *three tiers of noncustomers* [14]

3 Research Method

3.1 Research Approach

The research question "How did the solution strategy evolve during the life cycle of the software solution?" requires an historic perspective of the research topic. More precisely, studying the transformations of a solution strategy from the "novel idea" phase to the "mature industry" phase called for a retrospective case study, which is useful for addressing, for instance, a question regarding how a phenomenon behaves over time [17].

In order to gain rich understanding from the changes of important variables, such as market, technology, competition, and corporate strategy, the principles of two research methods were combined. As our primary research method, we used the case study method explained by Yin [18]. The method reveals richly detailed information that emphasizes the important contingencies that exist among the variables. In addition, the method promotes the researchers' own learning process with respect to the social phenomenon that is being observed. In particular, the method lets researchers study historical descriptions and events dealing with a full range of evidence sources, such as documentation, archival records, interviews, and observations. As a supporting research method, we used retrospective action research, in which practitioners afterwards reflect on what they have lived through [19]. The research method guides practitioners to act as researchers, which should provide unique access for and the collection of data and a pre-understanding of the research topic to be analyzed. Thus, practitioner-researchers who are closely involved in solution planning, from both corporate and solution strategy perspectives, are able to enrich data collection and analysis.

3.2 Unit of Analysis

In the study, the unit of analysis is the transformation of solution strategy. The selected solution is called *Security as a Service* (for consumers), provided by F-Secure. The company recently renamed the solution to *Protection Service for Consumers*. We selected this particular solution for three reasons. Firstly, the solution was novel in 2001 and its commercialization created a new market. Secondly, the new market grew very quickly and all the major competitors copied the business idea rapidly. Thirdly, the revenue from the solution continued to grow steadily, despite the tough competition in the mass consumer market. Thus, the solution was selected based on a purposeful sampling strategy [20]. Accordingly, the case was information-rich, and this provided the researchers with a great deal of data about issues of central importance to the purpose of the study. The rationale for selecting a single solution was to study the strategy of the same solution at different points in time [18]. A decision was taken to study the transformations of the solution strategy throughout the existence of the solution, from the year 2001 to the present.

The solution is illustrated in Figure 2. It consists of both software and service components that are provided for the two chains of customers. The direct customers are Internet Service Providers (ISPs). They receive, for instance, a hosted software platform and sales support for the management of the subscriptions and deployments of the security offered to their customers. Here, the customers of ISPs are called customers' customers. The customers' customers are consumers who have subscribed to the broadband services offered by the ISPs. To each of the customers' customers, for instance, F-Secure provides a client software product and daily protection service updates. The customers' customers can subscribe to both the broadband connection and security solution and, later, receive support from their local ISP. The solution is also co-branded to correspond to the ISP's brand.

F-Secure has over 800 employees on three continents: Europe, North America, and Asia. The company's headquarters, which employs nearly 400 people, is located in Finland. Moreover, the company has 18 country offices and a presence in more than 100 countries. F-Secure develops commercial software-intensive solutions mainly for mass markets. The solution that is being studied has been offered to protect the irreplaceable content of its customers' customers and has had a key role in promoting F-Secure as the most profitable value-added services partner for customers.

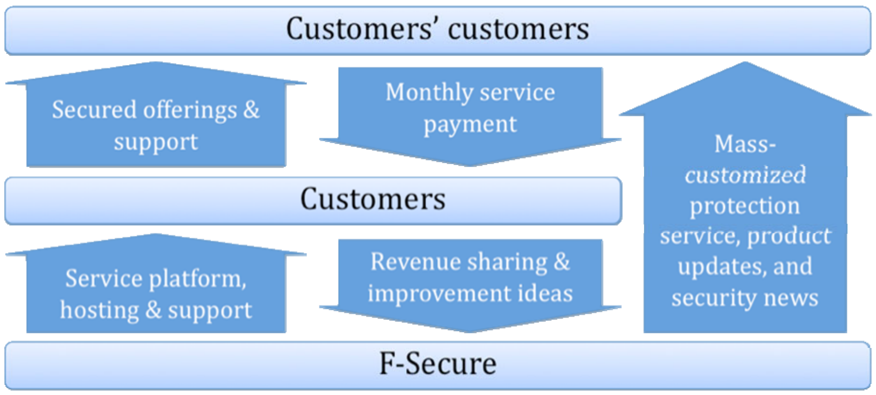

Customers' customers
Secured offerings & support
Monthly service payment
Mass-customized protection service, product updates, and security news
Customers
Service platform, hosting & support
Revenue sharing & improvement ideas
F-Secure

Fig. 2. A simplified illustration of the solution known as *Security as a Service for consumers*

3.3 Data Collection

Our data collection tactic was aimed at getting both external and internal points of view. From an external point of view, we focused on collecting data concerning the solution strategy and related influencing factors, such as market, competition, revenue, profit, customers, and partners, from a ten-year period. Archival records were our primary data source. We studied all the financial reports from the beginning of year 2000 to the present. We copied the relevant content from the 43 interim reports and 10 annual reports into separate text files for further analysis. We also read and listened to the multimedia package called "20 years of reliability" produced by the company. Four parts of the audio were transcribed for further analysis. In addition, we used an Internet archive called WayBackMachine [21] to collect the marketing messages for the solution from the company webpage from different points in time. Moreover, we studied the news archive on the F-Secure website from the beginning of year 2000 to the present. We concentrated on the material that contained information related to the solution and the deals made with partners and customers.

To get an internal point of view of the context and rationale of the solution strategy, we conducted nine interviews and two workshops. The interviews were conducted on an open-ended question basis. Nine key persons who were responsible for the development of the solution and/or who will have an important role in the development of future solutions were interviewed. Two of the interviewees were executive team members and the other seven were vice-presidents, directors, and managers of various functions, such as R&D, customer advocacy, and service development. The researchers conducted semi-structured interviews in pairs. All the interviews were recorded and transcribed from the recordings. The interview questions had three themes. Firstly, we clarified the most important customer group for the solution both at present and in the future. Secondly, the benefits of the solution for the customers were explored. Thirdly, the reasons why the customers were selecting the solution rather than one provided by the competitors were discovered. We also organized two workshops with the practitioners to clarify the preliminary findings from the interviews. The workshops were recorded and transcribed. In addition to the transcribed data, the researchers wrote notes during the workshops.

3.4 Data Analysis and Threats to Validity

The authors' pre-understanding of the research topic was utilized at first. The authors' had a wide-ranging prior knowledge of the topic, as three of the authors have worked at the case study company and the research group has been cooperating with the company on research for over six years. Throughout the study period, in particular, one author has been an executive member of the company and another author has participated in the business development of the solution. We utilized two research schemas to structure the pre-understanding. Firstly, we applied the three tiers of noncustomers [14] to analyze a case study market in 2001 and 2010. Potential customers of the intrusion prevention and content security solutions were categorized into three groups to describe the market and competition at these two times. Secondly, the strategy matrix [13] was used to identify the types of solution strategies followed by the company during 2001 and 2010.

The data collected from the archival records were carefully analyzed to enhance the preliminary findings. In particular, the data collected in relation to customers and solution development from the different periods of time were classified based on the four strategy matrix elements. After the classification, the rationale for each recognized solution strategy was identified. Next, the transcripts and notes from interviews and workshops were investigated to add details to the findings, and to format the key lessons learned from each of the four solution strategies. Finally, four of the company's key management personnel validated the findings.

The four strategy matrix elements [13] provided challenges for two reasons. Firstly, the strategy matrix elements consist of only one chain of customers. Therefore, we needed to decide that the potential and/or existing customer was a direct customer, an ISP, and to ignore the customers' customers, the consumers. On the other hand, the role of the customers' customers and information from them was used to identify the rationale behind why the case study changed the solution strategy over time. Secondly, the strategy matrix element known as the *new business* strategy was ambiguously defined as, "entering uncharted territory where the company cannot capitalize on any existing strength" which is hardly possible in practice. For instance, a company that is going for an uncharted territory might require, at least, its existing business management competence. We used market and technology uncertainties as classifiers to draw a clear distinction between the four strategy matrix elements [22]. As a result, the new business strategy meant developing and providing a new type of solution (including a high level of technology uncertainty) to a new type of customer segment (including a high level of market uncertainty). The same analogy was also used to clarify the definitions of three other strategy matrix elements. Moreover, it was also necessary to consider whether the new solution development affected the pricing of the solution in order to distinguish between share building and line extension strategies.

A threat to construct validity was the possibility of not being able to correctly collect and analyze the data related to the research question [18]. If the threat appears to be valid, our findings do not represent the reality of the changes to the solution strategy in the case study company. Therefore, we used triangulation of data sources and data collection techniques to reduce the threat. In addition, two key informants from the case study company and a researcher validated the findings several times to reduce the bias of a single researcher. A threat to external validity is evident in a single case study. However, we improved the possibility of our findings being useful to other software organizations by comparing them to related studies [3, 15] and providing contextual market and competition situation information from different periods of time.

4 Non-customers in the Market Evolvement

In this paper, the case study market is made up of existing or potential Internet users who have considered, or will consider, acquiring anti-virus and/or intrusion prevention solutions. In 2001, the case study market was growing very rapidly as households increasingly began to acquire their first broadband connections. The three tiers of non-customers are used to illustrate the case study market during that time (see Figure 3).

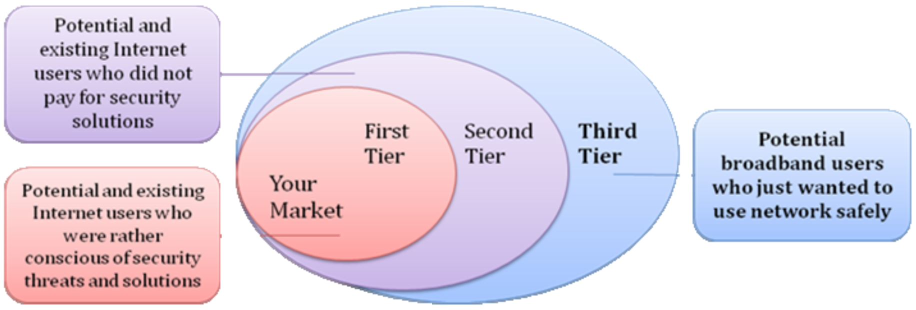

Potential and existing Internet users who did not pay for security solutions
Potential and existing Internet users who were rather conscious of security threats and solutions
First Tier
Your Market
Second Tier
Third Tier
Potential broadband users who just wanted to use network safely

Fig. 3. Consumer market in the anti-virus and intrusion prevention field in the year 2001

First Tier: Potential and existing Internet users who were rather conscious of security threats and solutions. The first tier describes the consumers who purchased security products as CD-boxes from the retail stores, or over the Internet from a security product vendor. The consumers valued the product's price and features as well as being influenced by magazine reviews and the recommendations from the salesperson at the retail store or a trusted friend. There was tough competition in the first tier. For instance, it was typical to see account wars between the software vendors to get deals with wholesale dealers, agents/suppliers, and large retail stores.

Second Tier: Potential and existing Internet users who did not pay for security solutions. The second tier portrays consumers who did not purchase any security solution. For instance, they did not care about security, they perceived the security of their operating system to be good enough, or they used illegal solutions. In addition, the second tier also includes advanced computer users, who used open source / freeware solutions. Open source solutions began to emerge at that time. However, vendors of existing security products had difficulties in finding a business model to profitably leverage the second tier. In fact, providing security solutions "for free" might have cannibalized their current business.

Third Tier: Potential broadband users who just wanted to use network safely. The third tier illustrates a very large set of households potentially purchasing their first broadband connections in 2001. These households typically did not include advanced users. These consumers valued convenience over technical details and product features. A large proportion of these consumers were not even aware of security threats. The optimal moment to increase their awareness of security threats and solutions seemed to be while they were negotiating broadband services with an Internet service provider (ISP).

In 2001, F-Secure launched a novel solution called *Security as a Service*, which aimed to meet the demands of consumers in the third tier. F-Secure hosted a customized security service that an ISP offered as a complement to its broadband services. The ISP was only involved in selling and billing the service and providing first-level support for consumers. In a broadband selling situation, a buyer (consumer) was able to gain an awareness of security needs and solutions as a complement to the broadband offering. The buyer perceived security as a natural and convenient part of a broadband offering. Mainly because of this SaaS solution, F-Secure was the fastest-growing publicly listed software security company for over four years in a row.

Less than two years after the launch, the third tier was the fastest-growing business area in the field of software security. Not surprisingly, the competitors had noticed the business area and had begun to address the same market segment. As a result, the characteristics of the original third tier began to resemble the original first tier. In other words, the original "unexplored non-customers" in the third tier were soon known to the every security vendor and the competition rapidly increased.

Figure 4 illustrates the situation in 2010. The original third tier has migrated to become part of the first tier. Consumers purchasing anti-virus and intrusion prevention solutions either as CD-boxes or as a service through ISPs both belong to the first tier. In particular, the business area of providing security as a service to consumers through ISPs has been subject to tough competition for years, in the same way as the traditional product business. In contrast, the second tier has not changed over time. While the number of users and providers of open source solutions has grown, the second tier has not been subject to tough competition. The major software security vendors do not yet have a strategic focus on open source solutions.

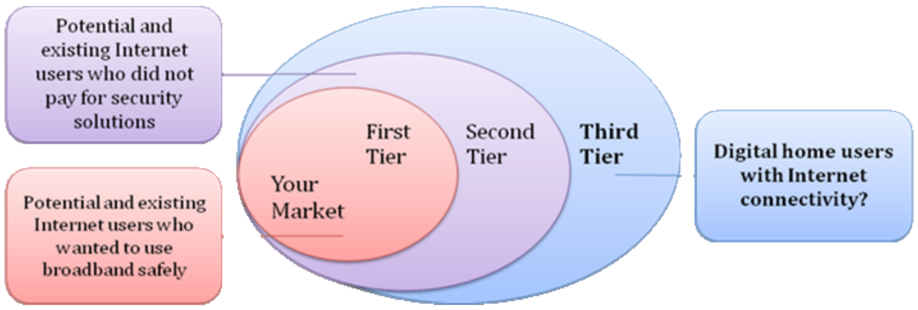

Potential and existing Internet users who did not pay for security solutions
Potential and existing Internet users who wanted to use broadband safely
First Tier
Second Tier
Third Tier
Your Market
Digital home users with Internet connectivity?

Fig. 4. The original third tier has migrated to form a part of the original first tier in year 2010

The future outlook for the second and third tiers is uncertain in 2010. For instance, "free" anti-virus solutions for consumers are of a good quality nowadays as the providers of these solutions aim to build a trusted brand at first, and then sell similar security solutions to companies. Moreover, Microsoft is providing security solutions for free. Therefore, the second tier may grow in the future, to the detriment of the first tier. The third tier may consist of digital home consumers who use, or will purchase, for instance, a variety of household appliances with broadband connections, such as Web TVs and game consoles. No major software vendor has launched a very profitable security solution to this market segment of digital home consumers. In addition, the number of new operating systems and platforms are increasing, due to mobile tablets, for instance.

5 Changes in the Strategy Matrix

The solution strategy of Security as a Service for consumers has transformed considerably between the years 2001 and 2010, as illustrated in Table 2. Using the terms of Scheuing and Johnson [13], the strategy variations have been: 1) new business (2001), 2) market extension (2003), 3) share building (2005), and 4) line extension (2008).

Table 2. Variations in the solution strategy between the years 2001 and 2010

Markets / **Offering**	**Existing customers**	**Potential customers**
Existing services	2005: Improving the existing solution and selling it to the existing customers (by leveraging the potential customers' customers)	2003: Improving the existing solution and selling it to ISPs worldwide
New services	2008: Developing new services and selling them to existing customers (by leveraging the existing and potential customers' customers)	2001: Developing and piloting new services and software technology to be consistent with a new business model

The new business strategy was required to commercialize a novel solution. In 2001, the company invested in developing a new solution to correspond to a novel business model. For instance, the new solution involved a different pricing model and chain of customers than traditional consumer products. Ari Hyppönen, the former CTO, recalled: "*It is not really a technological innovation but a business model innovation and this is where I would say that F-Secure's strategic advantage has been. We have been able to innovate in the way we provide the solution, not only in the way the solution works.*" In 2001, the company did not only develop a new software platform, but also new services such as billing support, as part of the offering to the ISPs. New solution development and commercialization included piloting with a domestic ISP partner.

The market extension strategy aimed at internationalizing the solution. After the successful piloting and first partnership deals with ISPs in nearby European countries, the strategy shifted from new business to market extension in 2003. The strategic focus was to build partnerships with new customers all over Europe, and later worldwide. ISPs formed the first chain of potential customers, and the second chain was their potential customers (consumers). The key target customers were major ISPs. In other words, the strategic focus was primarily to gain ISP partners globally, and then to help them sell a complementary solution to their new customers.

The share building strategy sharpened the implementation of value proposition. In 2005, the competition had become intense. On one hand, the company had a competitive advantage, as they had the longest experience in providing the services to ISPs. On the other hand, the competitors were bridging the gap, and this made it more difficult for the company to get partnership deals with new ISPs. Therefore, the strategy changed from market extension to share building. In practice, the company invested in actions that increased customer loyalty and the revenue shared with the customer. The key value proposition of the solution was to increase the loyalty of customers' customers while maximizing profits. The strategic focus pushed to develop and improve sales process and marketing support, with the aim of increasing sales by customers to their customers. The revenue growth was derived from the potential customers of ISPs, while the existing customers of ISPs were not excluded. The company began to pay more attention to understanding the reasons why some of the customers' customers were disloyal. All in all, the company focused on keeping the existing customers happy and on growing with them. These investments in the implementation of value proposition also promoted an increase in the number of ISP partners.

The line extension strategy aimed at leveraging the current customer base in new solution development. In 2008, the business domain of providing security as a service to consumers was already mature. The ISP market had become crowded, as almost all of the major ISPs had a partnership with a software security vendor. Because of the tough market situation, on one hand, the company kept focusing on their existing customers. On the other hand, the company's strategy and slogan were changed to preserve the company's future growth. The corporate strategy directed the transformation of the solution strategy from share building to line extension. The company invested in acquiring new technology and in the development of new solutions to provide *storage as a service*, such as online backup, to existing customers. The company has proclaimed to its investors that the current customer base (including customers' customers) is a valuable asset for the line extension strategy that provides growth opportunities. The company's financial report from the third quarter of 2010 states, "The company currently has more than 200 partners in over 40 countries with an addressable market of over 70 million broadband consumer customers. F-Secure has not lost any of its existing partnerships". According to the informants, it is crucial to understand the daily processes of both customers / partners and customers' customers in order to successfully leverage the customer base.

6 Discussion

This paper has sought to answer how the solution strategy evolved during the life-cycle of the software solution. The retrospective case study analyzed the solution strategy of Security as a Service for consumers against the strategy matrix [13]. The key finding of the study is that the solution strategy has gone through all four stages of the strategy matrix within nine years. The shifts in the solution strategy were critical in order to ensure sustainable growth in the revenue of the solution.

Our findings present an example of a sequential path for altering a software solution strategy in the mass-consumer market over time. Firstly, we illustrated the market situation in 2001 that provided a window of opportunity for the new business strategy of the case study company. Secondly, it was found to be beneficial to transform the solution strategy from new business to market extension, in order to internationalize the solution. Thirdly, it was found that the shift to the share building strategy was beneficial in closing relationships with existing customers and, in particular, further implementing the value proposition. Finally, the transition to line extension leveraged the current customer base in the development of a new solution.

The point at which the solution strategy should be transformed is not obvious, and neither is the direction it should take. F-Secure could have chosen to follow another path in the variations of solution strategy or to make the changes at different times. The initial extension from a small domestic market into a large international market was an obvious decision. However, as the next step, the company could have moved from market extension to line extension instead of sharing the building strategy or continuing with the market extension strategy. These alternative decisions might have resulted in the loss of some of its existing partnerships, while providing other competitive advantages, such as new (types of) partners or services. However, we believe that the actual transformation to sharing building strategy was correct and was performed early enough. The shift sharpened the implementation of value proposition and promoted the loyalty of the exiting customers/partners. Retention rate is certainly one of the most critical variables for management to focus on, as it is much more expensive to acquire new customers than to keep and serve existing ones [23].

Our findings indicate that the key drivers of the strategic changes were competition and growth potential. Eventually, the tough competition in the mass-consumer market led to a situation whereby existing customers and their customers provided better opportunities for the solution's revenue generation than did new customers. Similarly, a recent study by Claudio and Marchi [3] pointed out that mobile phone manufacturers have radically transformed their solution/product strategy over the industry life cycle. They reported that the key drivers for change have been the intense competition and rapid changes in technology and mass-consumer preferences. This implies that software companies, which operate in the mass markets, must carefully follow the evolution of the competition and the market and react accordingly. Actually, it is recommended to define a monitoring plan to detect any undertakings by competitors that could alter the solution strategy [2].

Our study also explained how the solution life cycle went from the exploratory stage to the growth stage, and then to the mature stage. The tough competition made it increasingly difficult for F-Secure to maintain growth in the revenue of the solution. Therefore, the vital question for the management was how to sustain the growth, especially in the mature stage. The answer was to focus on the customers' customers. In fact, our findings indicate that the customers' customers can provide huge opportunities for growth. Our case study company has been able to address over 70 million customers' customers through just over 200 partners/customers. Grönroos [24] has also emphasized the role of customers' customers in extending the network of relationships. However, shifting the focus from the customers to their customers may bring new challenges. Our previous study found that the balance between the customer and the customers' customers requires special attention [7]. The study highlighted the fact that the company has to be aware of the needs of the whole customer chain. Our findings underline that the company needs to keep a careful eye on the daily business processes of both customers and the customers' customers.

The three tiers of non-customers [14] were used to illustrate the changes in the market and competition. The third tier of non-customers, which were potential broadband users who just wanted to use the network safely, was found to be a novel market. As the market soon faced heavy competition, it became part of the first tier. It seems that the shift from the third tier to the first tier characterizes the market evolvement and competition in mass-consumer markets. A similar example is the target market of Nintendo Wii, which was people who do not play video games; this also originally represented the third tier [16]. Four years after the successful release of Nintendo Wii, it is quite obvious that Nintendo's competitors are addressing the same market with their solutions; namely, with Playstation Move and Kinect. In other words, the third tier has become part of the first tier, as in our case. This paper suggests using the three tiers of non-customers to periodically describe potential customers in a mass-consumer market.

There may be some challenges when attempting to describe the changes of solution strategy with the strategy matrix [13] for a solution similar to our case. Firstly, without adaptations, the strategy matrix may provide a limited view of target customers. In mass-consumer market segments, the delivery of a solution to the end-customer typically involves a chain of customers/stakeholders. Therefore, just considering whether to focus on existing or new customers is not enough in the solution strategy. The managers need to determine the whole customer chain and prioritize the importance of each chain. The importance of each chain may also change during the life cycle of the solution, like it did in our study. Secondly, it is not clear whether the solution strategy should focus on only one or many options of strategy matrix at a time. For instance, a technopark company leveraged all four strategy matrix options concurrently for a service concept [15]. However, a single dominating option can provide a clear spotlight for the solution development teams. Moreover, pushing for a single denominator is likely to create better insights for the management team than letting themselves off with three or four denominators [25]. Therefore, the management should periodically decide and communicate the dominating option of the strategy matrix in order to crystallize the current solution strategy. Our findings provide insights into the advantages that each of the four strategy options can offer in the different stages of the solution's life cycle.

7 Conclusions

This paper presents the findings from the evolution of the software solution strategy between 2001 and 2010. Our retrospective study discovered that the solution first created a new mass-consumer market that grew and soon became mature. Consequently, the main finding of the study is that the solution strategy has gone through four different stages, each of which contributes to achieving a competitive advantage in the different phases of the solution's life cycle. The stages of the solution strategy differed in terms of whether the focus was on existing or new services offered to potential or new customers. Our findings indicate that customers' customers are crucial for growth opportunities, particularly when the original market has become mature.

This study presents the results from the analysis of the changes that occurred in the solution strategy over a period of nine years. In the future, it would be interesting to compare the transformations of solution strategies in different types of business domains. In addition, because the changes in the solution strategy must remain faithful to the company's strategy and resources, it is worthwhile studying how to continuously link the solution strategy to the corporate strategy and portfolio management. Also, the links between the solution strategy and requirements engineering practices need to be studied further. Finally, as the customers' customers were found to be a valuable asset for the development of new solutions, the challenge of improving ideas related to eliciting and analyzing the customers' customers' data seems to be a promising new research avenue.

References

1. Christensen, C.M., Suáres, F.F., Utterback, J.M.: Strategies for Survival in Fast-Changing Industries. Management Science 44(12), 207–220 (1998)
2. McGrath, M.E.: Product Strategy for High-Technology Companies, 2nd edn. McGraw-Hill, New York (2001)
3. Giachetti, C., Marchi, G.: Evolution of Firms' Product Strategy over the Life Cycle of Technology-Based Industries: A Case Study of the Global Mobile Phone Industry, 1980-2009. Business History 52(7), 1123–1150 (2010)
4. Karakaya, F., Kerin, R.A.: Impact of Product Life Cycle Stages on Barriers to Entry. Journal of Strategic Marketing 15(4), 269–280 (2007)
5. Klepper, S.: Industry Life Cycles. Industrial and Corporate Change 6(1), 145–181 (1997)
6. Brereton, P., Budgen, D., Bennett, K., Munro, M., Layzell, P., Macaulay, L., Griffiths, D., Stannett, C.: The Future of Software. Communications of the ACM 42(12), 78–84 (1999)
7. Komssi, M., Kauppinen, M., Heiskari, J., Ropponen, M.: Transforming a Software Product Company into a Service Business: Case Study at F-Secure. In: Proceedings of the 33rd Annual International Computer Software and Application Conference, COMPSAC, pp. 61–66 (2009)
8. Sharon, A.M., Eschinger, C., Eid, T., Swinehart, H.H., Pang, C., Pring, B.: Market Trends: Software as a Service, Worldwide, 2008–2013, Update. Gartner, November 4 (2009)
9. Katzmarzik, A.: Product Differentiation for Software-as-a-Service Providers. Business & Information Systems Engineering 3(1), 19–31 (2011)

10. Concha, D., Espadas, J., Romero, D., Molina, A.: The e-HUB evolution: From a Custom Software Architecture to a Software-as-a-Service implementation. Computers in Industry 61(2), 145–151 (2010)
11. Lassila, A.: Taking a Service-Oriented Perspective on Software Business: How to Move from Product Business to Online Service Business. IADIS International Journal on WWW/Internet 4(1), 70–82 (2006)
12. Lehtola, L., Kauppinen, M., Vähäniitty, J., Komssi, M.: Linking Business and Requirements Engineering: Is Solution Planning a Missing Activity in Software Product Companies? Requirements Engineering 14(2), 113–128 (2009)
13. Scheuing, E.E., Johnson, E.M.: A Proposed Model for New Service Development. Journal of Services Marketing 3(1), 25–34 (1989)
14. Kim, W.C., Mauborgne, R.: Blue Ocean Strategy. HBS Press (2005)
15. Kaartemo, V., Peltola, K.K.: New Service Development in an International Context: a Case Study of a Finnish Technopark Company in Russia. International Journal of Business Excellence 2(3-4), 338–401 (2009)
16. Ziesak, J.: Wii Innovate - How Nintendo Created a New Market through the Strategic Innovation. GRIN Verlag (2009)
17. Bozeman, B., Kingsley, G.: R&D Value Mapping: A New Approach to Case Study-Based Evaluation. Journal of Technology Transfer 22(2), 33–41 (1997)
18. Yin, R.K.: Case Study Research – Design and Methods, 3rd edn. Sage Publications Inc., Thousand Oaks (2003)
19. Gummesson, E.: Qualitative Methods in Management Research, 2nd edn. Sage Publications Inc., Thousand Oaks (2000)
20. Patton, M.Q.: Qualitative Research and Evaluation Methods, 3rd edn. Sage Publications Inc., Thousand Oaks (2002)
21. Internet Achieve WayBackMachine, `http://web.archive.org/web/*/http://www.f-secure.com`
22. Macmillan, I.C., McGrath, R.G.: Crafting R&D Project Portfolios. Research Technology Management 45(5), 48–60 (2002)
23. Gupta, S., Lehmann, D., Stuart, J.A.: Valuing Customers. Journal of Marketing Research 41(1), 7–18 (2004)
24. Grönroos, C.: The Relationship Marketing Process: Communication, Interaction, Dialogue, Value. Journal of Business & Industrial Marketing 19(2), 99–113 (2004)
25. Collins, J.: Good to Great: Why Some Companies Make the Leap ... and Others Don't. Collins (2001)

The Sun also Sets: Ending the Life of a Software Product

Slinger Jansen[1], Karl Michael Popp[2], and Peter Buxmann[3]

[1] Utrecht University, Utrecht, the Netherlands
s.jansen@cs.uu.nl
[2] SAP AG, Waldorf, Germany
karl.michael.popp@sap.com
[3] Darmstadt University of Technology, Darmstadt, Germany
buxmann@is.tu-darmstadt.de

Abstract. Sunsetting a software product is a painful and frustrating process, whether it happens in times of crisis or in an organized and planned manner. It is surprising that little information is available on how to perform sunsetting and it appears to be a blind spot in software product management literature. This paper describes the sunsetting method and provides practitioners with a well-defined process of how software products must be taken out of development, maintenance, and finally use. With the sunsetting method, product managers will have as little trouble as possible based on the experiences of others. The process description is elaborated using a method description. Furthermore, three retrospective case studies have been conducted to evaluate the method.

Keywords: sunsetting, product software, end-of-life, method engineering.

1 Introduction

We define sunsetting as the process of planning and executing the end-of-life of a software product that is currently in use by customers and maintained by a software producing organization. The end-of-life of a software product describes the point after which the product is no longer maintained or supported by the manufacturer of the software product. Phase-out is an alternative term for sunsetting. Sunsetting is actually part of the portfolio management process.

Portfolio management is defined as the strategic decision process, whereby a business portfolio is constantly updated and revised in order to meet business objectives [1]. In the context of software vendors, the aim of portfolio management is to get the most out of a companys investments and products [2]. Good portfolio management requires close oversight, constant review of historical and current performance, and the courage to rebalance and rationalize the portfolio when necessary while aligning your actions with the overall strategy of the organization [3]. Basically, product phase-out is part of the continuous assessment that a software product manager undertakes when evaluating the product portfolio.

In a time of double-digit growth in the software industry, it may seem illogical to address the death of a software product. Why would one wish to phase out an unsuccessful product when there is always the chance that a customer might order it, or orders extra licenses out of the blue? As more experienced software product and portfolio managers know, there are many reasons to do so. These reasons are found in three categories, being product strategy, platform changes, and portfolio decisions [4,5].

In regards to product strategy, there are simple reasons such as release deprecation (e.g. a software vendor only supports the last two minor releases) and a lack of demand for the product. Another reason may be that maintenance becomes too expensive, i.e., upkeep for multiple products is too expensive for one company, or when developers for a technical platform become too scarce. It must be noted that a products life expectancy and potential profitability is more relevant than current profitability. If a product is successful presently, but will be hard to monetize in a couple of years because the product is no longer needed or based on technology that will become outdated soon, it can become a candidate for discontinuation.

The platform on which the product is built can also change the course of a products lifecycle. If a new release is made of the underlying technology, for instance the operating system as a basis for applications, the product owner has to decide when the releases of the product based on the older platform are no longer required to maintain a healthy business. Another platform on which the product depends might be a database system. If a database system is phased out, the product needs to evolve as well, or be phased out as well.

Finally, there may be portfolio decisions that end the life of a product. A product may be an inferior duplicate to another product, or a product may be outdated. Another reason may be that the product is no longer profitable or even performing badly in such a manner that it is causing harm to the software vendors reputation. Finally, legal constraints may force a product owner to kill off a product. These legal constraints may be that the company has formed a monopoly and is forced to reduce specific activities, or that intellectual property laws are broken by the product.

There are several factors that influence the ideal moment, with the least damage to the company, to end the life of a software product. Environmental influences, such as the entrance of a new standard or the introduction of regulatory requirements, such as XBRL-based reporting (a universal standard that allows for automatic processing of business accounting data), may mark such an ideal moment. Also, please note that there is a difference between ending the life of a product all-together and ending its life as a customer of an application [6], even though these processes have many things in common, such as changing customer needs, technical change, regulatory change, and competitive change.

To further illustrate, we take the example of Microsoft when it ends the life of one of its own products, by taking a closer look at one of the Windows versions. Microsoft publishes three dates to customers for each version of the Windows operating system in regards to sales, being the date of general availability, retail end of sales, and the end of sales for PCs with Windows version pre-installed. Furthermore, three dates are published in regards to support, being the publication of the latest service pack, the end of mainstream support date, and the end of extended (paid) support date. Obviously, for some organizations a major operating system upgrade is a huge undertaking, in terms of system maintenance (imagine an organization with over 5000 workstations), in terms of system compatibility (organizations easily have over 10,000s of applications of which many are compatible with one version of Windows only), and in terms of investment (hardware may be outdated, the upgrade will involve acquiring new licenses). An interesting detail of Microsoft's terms of service is that pre-installed deployments of Windows sometimes have downgrade rights, enabling the customer to downgrade to a previous version of Windows that is compatible with the other Windows versions in the organization.

We continue this paper by describing the research method in section 3. The research method is followed by a decomposition of the interactions between software vendor and customer in section 2, to illustrate what type of agreements need to be dismantled when ending the life of a software product. In section 4 the product software discontinuation method is presented and described in detail. Section 5 continues with the description of three case studies that illustrate the method and show the intricacies of the sunsetting process. Finally, in section 6, the conclusions are derived and discussed.

2 Decomposing Sunsetting

We now provide a further explanation of the complex operation of sunsetting. This paper looks specifically at on-premise software, which is provided from a software vendor to the customer. For the sake of simplification, let us assume a simple, direct relationship between the software vendor and the customer. In this case, sunsetting is like rolling back a distributed transaction between the software vendor and a customer. This distributed transaction can be divided into three subtransactions: *Provide software*, *Provide maintenance* and *Provide support*. While rolling back this transaction and its three subtransactions seem simple, the next level of detail shows that there are subtransactions that cannot be rolled back. They need compensating transactions and thus introduce complexity and efforts into the process of sunsetting solutions. Another factor for adding complexity and efforts is customer lock-in.

Provide software - The transaction *Provide software* is divided into the subtransactions *Provide a copy of the software*, *Transfer usage rights* and *Provide license key*. The first transaction *Provide a copy* of the software can be easily rolled back if the customer has a time limited license. The customer just has to give back the copy of the software at the end of the license term. If the customer has a perpetual license, the customer can keep the copy of the software. The second transaction *Transfer usage rights* cannot be rolled back in a simple manner. Based on the contract terms, the customer can keep the usage rights, the usage rights can end. If the customer has perpetual usage rights, he needs to get usage rights on a software product that replaces the sunsetted product.

Replacing the sunsetted software product introduces more complexity and effort for the software vendor and the customer. Replacing a sunsetted software product with a new software product means that the results of activities that have gone into installing, implementing, maintaining and running the software cannot be rolled back and usually carry high sunk cost. A compensating transaction has to collect the results of these activities and migrate these results into a new software product that replaces the sunsetted product. Simple examples for these results are customer data or customer specific extensions of the software product. How this replacement is properly planned and executed will be covered later in this paper. The third transaction *Provide license key* follows the same logic of rolling back as *Transfer usage* rights.

Provide maintenance - *Provide maintenance* is divided into subtransactions *Provide new release*, *Provide new version* and *Provide bugfix*. Over time, the customer is served by multiple executions of these subtransactions. At a certain point in time, the customer arrives at a combination of release, version and bugfix. In the case of replacement of sunsetted product, the customer needs a replacement of exactly the combination of release, version and bugfix he is currently running. This shows additional complexity in replacing sunsetted software, since each of the customers might have a specific combination that might differ from the combination each other customers have. Numerous instances of these transactions have been executed and have lead to the current system landscape at the customer.

Provide support - The third high-level transaction is *Provide support*, which aims at providing resolutions or workarounds for customer issues with the software. Each of the transactions was executed several times. The result of the transactions is a customer specific set of resolutions or workarounds. The transactions do not have to be rolled back.

3 Research Method

The research question of this paper is:

How can a method be created for a product manager to sunset a product (line)?

The research question is answered by applying method engineering in a design research project. Method engineering is used for designing, constructing and adapting methods, techniques and tools to develop information systems [7]. Design science is an outcome based information technology research method, which offers specific guidelines for evaluation and iteration within research projects [8].

Research Execution - The research consists of three steps. A first version of the method was created to create a baseline method, based on literature and experience. The method is evaluated with several experts from the industry (with 15, 15, and 13 years of experience), who have long-standing experience with retiring and sunsetting software products and product lines. Thirdly, the method is evaluated by doing three exploratory case studies, to establish that the method is complete. The case studies are listed in table 1 and further discussed in section 5.

Method Engineering - van de Weerd et al. [9] describe a meta-modeling technique based on UML. This technique depicts a method in a Process-Deliverable Diagram (PDD). A PDD consists of two parts: a process model (UML activity model) on the left side and the deliverable model (UML class diagram) on the right side. An example PDD is depicted in Figure 1, which models a highly simplified requirements engineering process. On the left-hand side an activity called “Requirements elicitation” is modeled, which contains the sub-activity “Write requirements document”. The requirements engineer executes the activity. The activity results in a deliverable called “Requirements Document”, which has several properties. The main reason for using

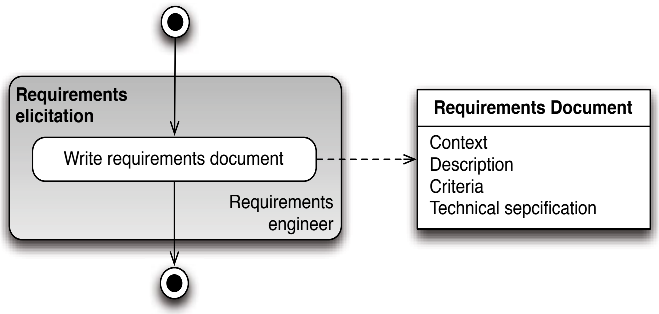


Fig. 1. Process-data diagram [9]

Table 1. Companies and Products (* age of the software business, the company was established in 1878)

Case Study Company			Phased out product			
Identifier	**# of employees/products**	**Age**	**Location**	**Reasons for phase-out**	**# of employees in product unit**	**Age**
PubComp	18000/40+	15*	Netherlands, Europe	Out of portfolio scope, not making targets	120	20
ERPComp	51000/100+	38	Germany, world-wide	Redundant, rebranded product branch, contract standardization, reduce maintenance efforts	20	10
ServicesComp	9100/100+	18	Netherlands, Europe	Duplicate functionality, aged technology	135	16

this type of method descriptions is that it enables us to present the sunsetting method in a structured manner, and enriches the main contribution of this paper from a simple checklist, to a rich method description that can be reused by practitioners and improved upon by academics.

The three companies were opportunistically selected, since we had access to board level management in each of the companies. Each of the companies, however, has a long experience dealing with large product portfolios and were selected with that reason in mind. Each of the interviewees was working at one of the case companies at the time. The interviews were undertaken in four steps. First, a general discussion was had about the organization. Secondly, we discussed the topic of phase-outs, and tried to establish the interviewees view on the topic, including any previous experiences the interviewee had with the topic. The interviewees were asked to develop a quick outline for a method themselves, to see whether they understood the topic and to see whether their experiences further confirmed the first version of the method. Documentation, if available, was handed over in regards to product end-of-life. In the third step, the first version of the method was shown to the interviewees and walked through with the exact same narration for all interviews. The interviewees were allowed to comment on the method and all three at some point grabbed a pen to make their own additions and changes. During step four, an example from the interviewee's past was taken to see whether the method was followed in any way.

Three interviews were undertaken, of which two in Dutch and one in English. The method was always described in English and each term was explained in a glossary, which the researchers brought to the interview. Interviews took between 100 and 150 minutes. Each of the interviews was recorded. The interviews were conducted by one researcher only. The results from one of the interviews were checked by a second researcher.

4 The Product Software Discontinuation Method

This section describes the method shown in figure 2. On the left side the method activities are displayed, on the right side the deliverables created during the execution of the method are shown. These deliverables are further explained in table 2. The first version of this method was created from literature, the second version was created based on three interviews and case studies.

The first version of the method was in part inspired by IEEE std 1074 [10], a process standard for the software lifecycle, which provides a concise description of the retirement process. The description consists of three steps, being "notify user", "conduct parallel operations", and "retire system". The "parallel operations" step consists of using two systems simultaneously, while one of the two is being phased out. The IEEE standard provides some insight into the retirement process, but does not specifically address the challenges a software vendor may experience during a product phase out. The method was also inspired by several product phase-out overviews from larger software vendors, such as the Microsoft Windows phase out web pages, the information pages from SAP about Business Object's (acquired by SAP) SRC, and the pages from Cisco about the Quality of Service (QoS) Device Manager Software, which was phased out over a long period in the previous decade.

Discontinuation Assessment - Any organization that maintains a software product must regularly assess the viability of its products and product lines, as part of the portfolio management process. This process consists of reviewing the portfolio plans for current products, assessing their success, profitability, market size, and growth. When a product becomes a potential candidate for discontinuation, a customer assessment needs to be done to establish how important the product is for these customers and how important these customers are, since discontinuation of the product could mean the termination of a long-lasting relationship. Finally, a list must be created of all the products that depend on the product that may be discontinued. In case of discontinuation, the teams behind all products on the list of dependent products must be informed.

Phase-out Planning - Whenever a phase-out is impending, software vendors need to evaluate possible alternatives before actually phasing out a software product. There are several alternatives to phasing out a software product that still ensure continuation of

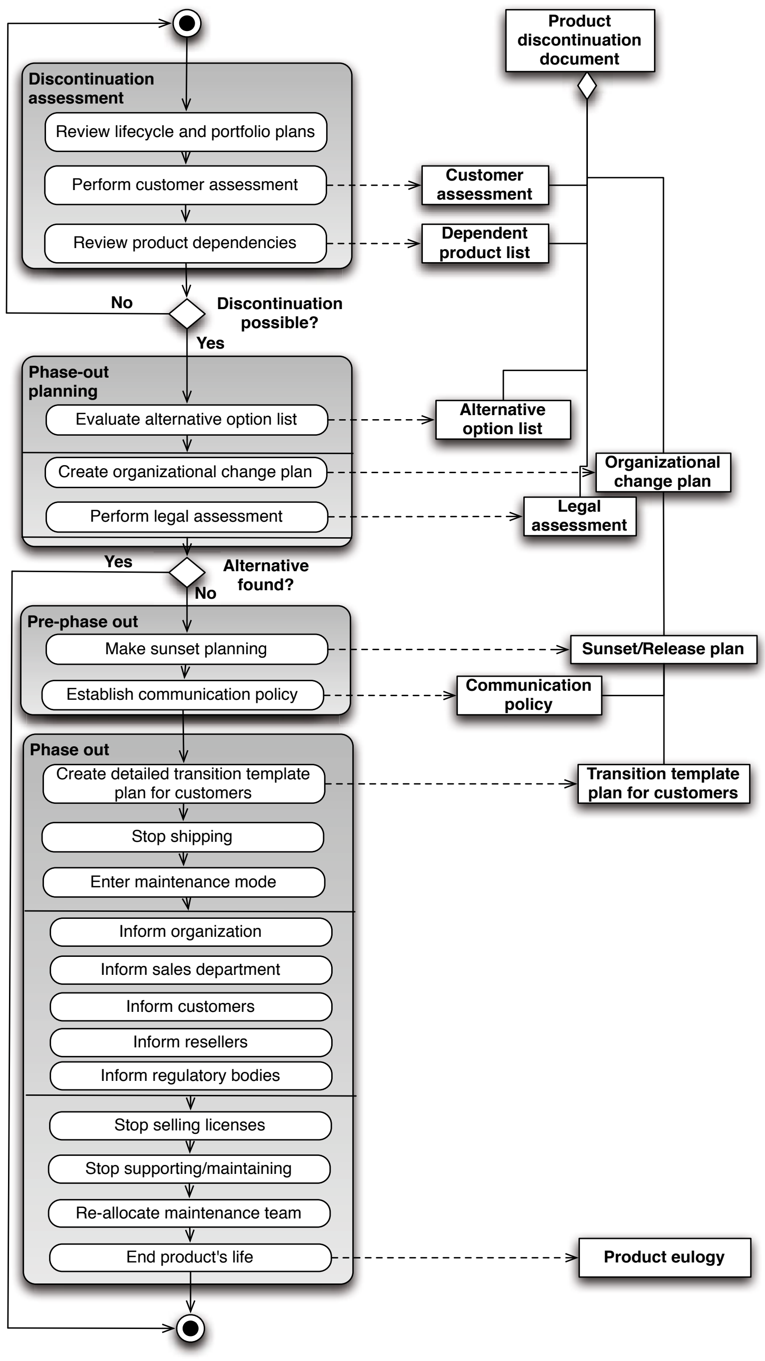

Product discontinuation document
Discontinuation assessment
Review lifecycle and portfolio plans
Perform customer assessment
Review product dependencies
Customer assessment
Dependent product list
No
Discontinuation possible?
Yes
Phase-out planning
Evaluate alternative option list
Create organizational change plan
Perform legal assessment
Alternative option list
Organizational change plan
Legal assessment
Yes
Alternative found?
No
Pre-phase out
Make sunset planning
Establish communication policy
Sunset/Release plan
Communication policy
Phase out
Create detailed transition template plan for customers
Stop shipping
Enter maintenance mode
Inform organization
Inform sales department
Inform customers
Inform resellers
Inform regulatory bodies
Stop selling licenses
Stop supporting/maintaining
Re-allocate maintenance team
End product's life
Transition template plan for customers
Product eulogy

Fig. 2. Product Software Discontinuation Method (all activities executed by product owner)

Table 2. Concept Table

Concept Name	**Concept Description**
PRODUCT DISCONTINUATION DOCUMENT	This document describes the complete plan on how to discontinue the product. The creation of the document only expresses the intent to explore the options for discontinuation and plans may be discarded after the LEGAL ASSESSMENT and CUSTOMER ASSESSMENT have been completed.
CUSTOMER ASSESSMENT	The CUSTOMER ASSESSMENT is performed to explore what the effect will be for customers and how relationships will be altered after discontinuation. The assessment includes a general assessment and a specific assessment per individual customer.
DEPENDENT PRODUCT LIST	The DEPENDENT PRODUCT LIST lists all products that are dependent on the discontinued product and therefore may also have to be discontinued.
ALTERNATIVE OPTION LIST	The list of alternatives provides an overview of alternatives to discontinuation, such as the sale of the product to a third-party.
LEGAL ASSESSMENT	This document describes the legal risks and consequences, based on the CUSTOMER ASSESSMENT and DEPENDENT PRODUCT LIST.
ORGANIZATIONAL CHANGE PLAN	This document describes how the organization will change in the following period as the product is discontinued.
SUNSET/RELEASE PLAN	The SUNSET/RELEASE PLAN describes how the product was planned to be phased out, if such a plan is available. If not a sunset plan is created. It is also called a lifecycle plan.
COMMUNICATION POLICY	The COMMUNICATION POLICY describes how and especially when the discontinuation plans are communicated. These documents are generally sensitive and must be treated so by the entire organization to avoid leaking.
TRANSITION TEMPLATE FOR CUSTOMERS	Based on the CUSTOMER ASSESSMENT a TRANSITION TEMPLATE is created for each customer group, that can be filled in by a consultant, as to advise a customer in the transition to (for instance) a new system.
PRODUCT EULOGY	The PRODUCT EULOGY describes what the product was, how well it performed, and why it was phased out.

the business, although perhaps with different levels of quality of support. The following list is not exhaustive, but presents some of the alternatives to completely phasing out the software product:

– **Open source -** it is possible to establish an open source community that further supports and maintains the product. This option, if the product is not already open source, should provide a real alternative, i.e., a sustainable community must be created that maintains and provides support, and there should not be a contractual conflict in regards to licenses.
– **Management Buy-out -** if a product is being maintained by an isolated group of people within the organization and the product could become more profitable if it

did not have to support the parent organization, one option to avoid product phase-out is to have a group of adequate managers or key people in the product group buy the product out of the portfolio, and continue as an independent organization.
– **Sale -** when a product no longer fits the portfolio of an organization, it can be sold to a third party, along with (parts of) its maintaining organization. Organizations will generally go to great lengths to sell off a product instead of killing it off. A common phenomenon is that an organization starts "putting lipstick on the pig", i.e., trying to maximize profit and growth numbers to make the product group more attractive to a potential seller.

Based on the alternatives, an organizational change plan is created and a legal assessment is performed. The organizational change plan establishes how the product discontinuation is implemented in the organization, together with a timeline of events. A legal assessment is performed as well, to establish the consequences of the different variations of the plan. The legal implications can be serious, in that contractual obligations towards customers and partners may need to be altered to accommodate a products discontinuation.

Pre-phase out - If no alternatives are found, the product will need to be phased out. To do so, a planning is made that details the steps that are taken in the process. Such steps include the cessation of maintenance and support of a product, and the last sale of licenses. One of the most precarious processes of the phase-out is the establishment of a communication policy, due to the nature of the phase-out process. Phasing out a software product entails in some cases the complete disbandment of a products maintaining unit. The impact the discontinuation of a software product can have requires that a communication policy is set-up that minimizes damage to the organization.

Phase-out - After a phase-out plan has been crafted, the plan must be executed, starting with the creation of a customer transition plan template, which lists several routes to a new solution for customers, such as migration to a replacement product. This is a template and must be adjusted for each individual customer, since each customer operates from a unique situation. If the product is still being sold that must be stopped. Furthermore, the product must fully go into maintenance mode, such that no new functionality is added to the product, with the exception of updating time-sensitive content. Once the product is in maintenance mode, the communication policy is executed according to the timeline created earlier. After communicating with all stakeholders that the product will be discontinued, a last round of license sales can be done, after which license sales are ceased as well. As soon as the product legally no longer requires support and maintenance, these processes can be stopped as well, after which the maintenance team needs to be re-allocated. Finally, the product is finished, and the product manager or entrepreneur who has been responsible for the phase-out process, can write a product eulogy.

Painful Process - The method description does not sufficiently show that the process of phasing out a software product actually has great consequences for the people working with it. Think, for instance, of the support engineer who knows every nook and cranny of the software product, or the user who has configured the product just to her specifications and is described as the wizard of that product by her colleagues. We advise practitioners to make compromises and be sensitive towards the emotions that surround legacy products, both in their internal and external communication. By taking it slow and on-boarding fanatic proponents of the legacy products, transitions may potentially go much easier.

5 Case Studies

The case studies were performed to provide different examples of how the lives of products are ended. The cases served as a measure to evaluate the method provided in figure 2.

5.1 Case Study: Health and Safety Product at PubComp

Context. The HSP (a Health and Safety Product) was a relatively successful product that no longer supported the business goals of PubComp, a software vendor in the Netherlands with a large portfolio. The HSP came up in regular evaluations at PubComp as being out-of-scope of the product portfolio, and was consistently underperforming.

Process. These evaluations were top-level management evaluations, and were treated as top-secret, since they deciding on the future of approximately 120 people. After considerable time, a team was identified within the HSP business unit that could potentially undertake a management buy-out. The HSP team and PubComp agreed to a buy-out price and strategy. Within 18 months the HSP business unit was functioning independently and the formal contract was signed.

Findings. When a management buy-out or the sale of a product to another company is in sight, some interesting processes start taking place. There are issues with personell, resources, and business unit value determination (although one could argue this is relevant for ‘regular’ phase-outs as well). Purchasers attempt to find all the things that could bring the value of the company down, whereas the seller will be tempted to make the business unit look more successful than it actually is. The sale of a business unit is a more organic and management-directed transition than a planned phase-out.

Impact on the method. The first version of the method assumed that the end of a life of a software product always entails the complete ending of sales, support, etc. This first case already showed that it is rarely the case that the product, whose customers always represent some kind of business value, is completely phased out without any viable alternatives for the product (management buy-out or purchase by another company) or for the customers (in the form of a product alternative, for instance).

5.2 Case Study: Enterprise Resource Planning Product at ERPComp

Context. A large platform provider in Germany has been growing autonomously, for almost 40 years. The company has gone through several product sunsets, but availability of knowledge on these processes within the organization remains scarce. In 2007, ERPComp acquired a company that specializes in identity management products and was aimed to be complementary to the ERP platforms supplied by ERPComp. Throughout the years, ERPComp integrated some of the technological feats that came with the identity management products into a new identity management platform, thereby making some of the original products redundant. In 2007 it was decided that the last products that were still being supported by ERPComp from the acquired company were to be sunsetted. In total, different versions of six products were to be phased out.

Process. First, an overview was created of the different products that were currently in use and of how many active customers each of these products had. Secondly, an overview was created of upgrade and migration routes that could be traveled by customers. Part of this overview was a plan that described what happened when a customer ordered extra support, licenses, or even a new deployment.

Findings. An extra complication was that some of the products had been resold and rebranded under another label and these also needed to be phased out. Phasing out the rebranded product proved to be a challenge, since the resellers that sold the rebranded product also had service contracts with their customers. As soon as ERPComp phased out the product, the reseller contracts were also ended. Because of this, the customers of those resellers no longer received support and maintenance. ERPComp solved this elegantly by timing exactly when these customers would be willingly transitioned from the reseller to ERPComp.

The phase-out timeline had to be extended twice with one year, because ERPComp wrongly estimated the availability of consultants that would be needed for each transition. Furthermore, for some customers it proved to be interesting to extend the maintenance period further, against much higher license costs. A positive observation was that out of approximately 200 customers, there was no attrition.

Another challenge was that the transition template plan for customers had to be updated, due to the fact that new releases of the replacing platform came out. This had not been taken into account, and introduced extra cost into the phase-out process. ERPComp required support from its ecosystem of partners in phasing out the product. To gain their support, the partners were informed early of the consequences, and communication protocols were established. Partners were also trained in providing support during transitions, such that customers could receive support from their preferred partner at the same quality level as ERPComp would provide.

Impact on the method. The role of the communication policy and legal assessment were further outlined during this case study. ERPComp has many large customers, which expect the absolute best from their ERP vendor, and are willing to pay for it. If ERPComp makes any serious changes, a network of customers exists that is powerful and can litigate if necessary.

5.3 Case Study: Three Municipalities Products at ServicesComp

Context. A large software and services company in the Netherlands has been growing mostly through acquisitions. Three software producing organizations creating products for municipalities were acquired in 1996. Each of these software producing organizations has become a separate business unit of ServicesComp. Two of the products are complementary, whereas the third product copies up to 70% of the functionality of the other two. All three products have been developed on older technological platforms and all three products are currently maintained and marketed separately. ServicesComp perceives that three business units building similar products is inefficient and an initiative has been started to build a new best-of-breed product that will replace the other three. The decision to reduce the inefficiency has been made a long time ago by the management team of the business division. No structural approach was implemented at that time at ServiceComp for portfolio management, but at the time of acquisition it was already clear that the three products had overlap and that a merge would happen over time.

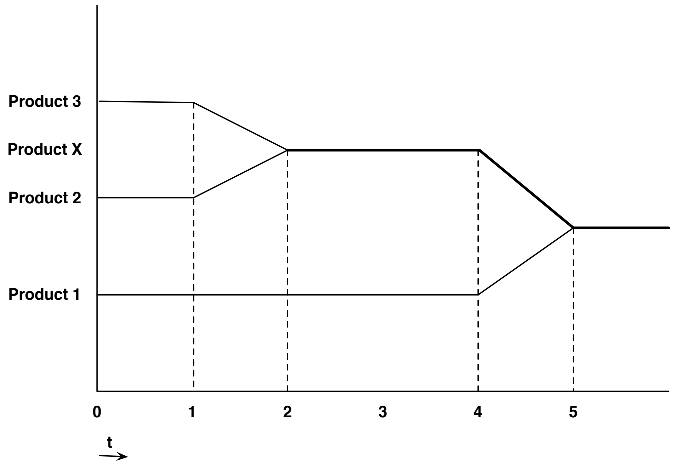


Fig. 3. Transitioning from Three Products to a New Product

Process. The actual merge has been visualized in Figure 3, which shows how the three products are phased out and replaced out by a fourth. The two products that are complimentary are phased out first, the third product will be phased out last. ServicesComp has steady five-year contracts, with yearly extensions after that. It is clear to ServicesComp when the last contract can potentially be ended, so planning for the merger of the products is more dependent on development speed than contractual obligations. ServicesComp looks positively toward the transition of customers from their old products to the new. To begin with, the transition is a business opportunity for ServicesComp, who provide a lot of services, and as their competition has made some mistakes in the past in regards to transition, ServicesComp is unafraid of customers transitioning to another vendor. ServicesComp has indicated that customer research is a major step in a product discontinuation document, since transitions will go much smoother and entail less risk if customers perceive the supplier of the original product positively.

Findings. When it was clear that the functionality of the three products would be merged into product four, the organization prepared a communication protocol. Parts of this protocol were reused to establish the rules of (potential) customer communication for sales personnel. ServiceComp used the following elements in this protocol:

– Roadmap of product X
– Rough timeline for phase-outs
– Generic features, such as connectivity
– News embargo on product 1, since the planning hadn't been finalized

The communication protocol was created to ensure customers that service for the products would be continued and discouraged them to look at competing products by describing the advantages of the new products.

Impact on the method. Again it was confirmed that providing alternative options to customers is a successful way of retaining them as customers. Furthermore, because ServicesComp has a mature view on the phase-out process, the case study functioned as a confirming case to show that the method does not miss any essential steps. None such steps were found and the second version of the method, as shown in this paper, could be published.

6 Discussion and Conclusion

The sunsetting process of a software product that has been deployed in the market is a complex and loaded process. It involves changes to the product portfolio of an organization, an impact analysis on the company, and structural change in an organization. The process should not be taken lightly, or mistakes will be made. In our case studies alone, we found examples of deadlines that were severely delayed, customers that experienced serious financial damage, and missed business opportunities (not offering a replacement product, for instance). The method provided in this paper hopefully helps organizations avoid future problems when ending the life of a software product.

During our research with experts and at companies, it became clear that this topic is considered sensitive. We have looked for method descriptions, process descriptions, and phase out plans, and found that these documents were always confidential, if available at all. We also found that because of its nature, the sunsetting process is generally performed ad hoc, even if a company is highly experienced in the field. It has become apparent that case studies and interviews are the best way to develop the method that has been created during this research.

It may appear that aspects of our approach (take it slow, communicate early, use a structural approach) are directly opposed to the profit goals of an organization. After all, if a business unit can be sold off, the sunsetting process is slow and mechanical, and the seller may feel it is missing out on a great deal. We hypothesize, however, that a structured approach will lead to less problems along the way and may even uncover that the life of the product (or business unit) is not yet over or cannot be over, due to customer lock-in.

In this paper we have focused on ending the life of a software product. It is surprising how little literature is available about such a meticulous and sensitive process, and this paper attempts to fill that void. We have defined the process of sunsetting, devised a method to support the process, and given some hints towards reasons to start sunsetting. We hope that practitioners will benefit from our research and that the scientific community further expands on how and when to sunset a software product.

References

1. Pohl, K., Bckle, G., van der Linden, F.: Software Product Line Engineering (2005)
2. Popp, K.M., Meyer, R.: Profit from Software Ecosystems. Books on Demand GmbH (2010)
3. Haines, S.: The Product Manager's Desk Reference. McGraw-Hill, New York (2008)
4. van de Weerd, I., Bekkers, W., Brinkkemper, S.: Developing a maturity matrix for software product management. In: Tyrväinen, P., Jansen, S., Cusumano, M.A. (eds.) ICSOB 2010. Lecture Notes in Business Information Processing, vol. 51, pp. 76–89. Springer, Heidelberg (2010)
5. Jansen, S., Brinkkemper, S., Finkelstein, A.: Business Network Management as a Survival Strategy: A Tale of Two Software Ecosystems. In: Proceedings of the First International Workshop on Software Ecosystems, vol. (2), pp. 34–48 (2009)
6. Furneaux, B., Wade, M.: The end of the information system life: a model of is discontinuance. SIGMIS Database 41, 45–69 (2010)
7. Brinkkemper, S.: Method engineering: engineering of information systems development, methods and tools. Information and Software Technology 38, 275–280 (1996)
8. Hevner, A.R., March, S.T., Park, J., Ram, S.: Design Science in Information Systems Research. MIS Quarterly 28(1), 75–105 (2004)
9. van de Weerd, I., Brinkkemper, S., Nieuwenhuis, R., Versendaal, J., Bijlsma, L.: On the creation of a reference framework for software product management: Validation and tool support. In: Proceedings of the 1st International Workshop on Product Management, Minneapolis/St. Paul, Minnesota, USA, pp. 3–12 (2006)
10. IEEE Standards Board: IEEE Standard for Developing Software Life Cycle Processes, IEEE Std 1074-1997 (1997)

Requirements Scoping Visualization for Project Management

Krzysztof Wnuk[1] and David Callele[2]

[1] Department of Computer Science, Lund University, Lund, Sweden
Krzysztof.Wnuk@cs.lth.se
http://www.cs.lth.se

[2] Department of Computer Science, University of Saskatchewan, Saskatchewan, Canada
callele@cs.usask.ca
http://www.cs.usask.ca/

Abstract. Determining requirements process efficiency, and measuring the corresponding monetary impacts, is a challenging but necessary aspect of project management. In this paper, we perform an independent analysis of scoping decisions from a large industrial project with the goal of providing visualizations that facilitate investigations of process efficiency, agility, and the effects of scoping decisions. The visualizations proposed in this paper can be used to analyze scoping dynamics and support process management decisions on a quantitative rather than a qualitative basis.

Keywords: Requirements visualization, process evaluation, requirements scope, project management.

1 Introduction

In Market-Driven Requirements Engineering (MDRE) [1], the time taken to deliver the product to market (and hence the overall release scheduling) is important and may strongly affect market success [2]. These time pressures place hard limits on all aspects of the development effort and force requirements efforts to be efficient and responsive [3]. Feature leapfrogging [4] between companies also imposes hard time constrains, which combined with resource constraints force requirements to be prioritized, with some requirements postponed for later implementation [5]. The process of selecting a subset of requirements for immediate implementation within a given project is called *scoping* and is considered a key activity for achieving economic benefits in product line development [6]. In MDRE, the scope of the project must adapt to competitive pressures and respond to changing market conditions in a timely manner – making appropriate scope decisions is a vital part of developing software systems that meet stakeholders' needs and expectations [5,7].

Project management in this context has the goal of delivering a quality product within the given resource constraints, with appropriate risk management and acceptable predictability. Making decisions in a timely manner is fundamental for productive software management – unnecessarily delaying decisions can lead to wasted resources and can place other aspects of the projects at risk due to resource constraints [8]. However, as reported by Boehm and Sullivan [9] there is a "disconnect" between the process of making technical software product decisions and the value creation criteria of the organizations that develop these products. Moreover, the growing importance of software in all aspects of successful business demands better understanding of the relationships between technical properties of the decisions and the criteria for business value creation. In other words, software development should be considered an "investment activity" that aims at maximizing value creation for the resources invested [9].

Our prior work [10,11,12] showed that in MDRE there is the potential for the project scope to be constantly changing, or at least be under pressure to do so, particularly from unanticipated market forces. For example, the last minute addition of cutting-edge features and technologies to a project can lead to significant investments in requirements definition and feasibility analysis efforts. The associated project management risks may include excessive resource allocations that starve other efforts, destabilizing technologies being added to the product before they are ready, and even outright project failure. This prior work proposed new visualizations to help to assess the dynamics of scope changes in this context using a post-mortem analysis perspective.

In this paper, we analyze scoping from a project management perspective and propose visualizations that support management decision-making. Using the action research strategy [13] the second author of this paper, who is an industrial Project Manager (PM) interested in Decision Support Systems (DSS), reviewed the previous work with the focus on the financial aspects of requirements engineering and requirements scoping and its implications on efficient project management. The project manager needs tools that can help him to identify opportunities for improvement *in the process itself* and to reduce risk by identifying wasted resources. As a result of this participatory effort, the following research questions are stated:

– What factors influence the process of removing features from the scope of the project? (RQ1)
– What factors influence the late addition of features to the scope of the project? (RQ2)

This work investigates the acts of adding and removing features from the scope of a product release; all features under consideration for a release are present before the scoping process begins. RQ1 identifies opportunities to optimize feature removal under the premise that, if the feature *is* to be removed from the release, then early removal wastes fewer resources. RQ2 investigates late feature addition to identify opportunities to reduce risks inherent in this decision.

In the remainder of this paper we review the related work and present a summary of the relevant results from our prior work (Section 2). We discuss the research approach and data used in this study in Section 3. Section 4 presents new visualizations that support financial analysis of the scoping process in projects.

We discuss threats to validity for this study in Section 5 and close with our conclusions and identification of future work in Section 6.

2 Related Work

Release planning is an integral part of software product management. Multiple releases are considered in a release plan [14] and the selection of the 'right' requirements for a particular release is normally preceded by requirements prioritization [15,5] and cost estimation [16]. Release planning is performed within the context of the product and corporate strategies, providing input to the feature selection process [17,18,19]. The feature selection process is not a trivial task; dependencies on other features [20] or frequent changes to the scope of the next release of a project [12] are typical complications. As a result, the selection process often becomes an uneasy compromise where the development efforts toward some features may need to be sacrificed (at the expense of wasted effort) in support of other features whose priorities have changed.

Managing this scope complexity is considered one of the core functions of software release planning and a key activity for achieving economic benefits in product line development [6]. While the importance of scoping has already been reported in several studies, most of the research is focused on the domain scoping aspect and the process of scope identification [6,21]. When looking at product portfolio scoping [6] most techniques focus on the financial benefits associated with reuse across the product line [22].

The quantity of information that must be managed for large software projects is often overwhelming and visualizations can be useful in this context, offering more dimensions to represent than text [23]. Appropriate visualizations can assist with, for example, the requirements comprehension problem of gaining a quick assessment of the state of a set of requirements; a task typically impeded by the need to browse through disjoint textual requirements documentation and accompanying models [24]. The visualizations presented in our previous work [10,11,12] give a support assessment of the scoping process for large projects and can also be used for more in-depth analysis of the details of the scoping process. In [10] the Feature Survival Chart was introduced and applied to one large industrial project. In [12] the Feature Survival Chart was complemented with a set of scope tracking measurements while in [11] the Feature Transition Chart technique, designed to cover scope changes across the projects, was proposed and initially validated. These visualizations can be extended via metrics such as the volatility of the feature set and temporal measurements such as the time taken to cancel a feature [12]. In this paper, we extend the previous work, presenting a new set of visualizations that aim at supporting project managers in understanding the scoping dynamics and assess the monetary impact of the visualized scope changes.

3 Research Approach and Data in Context

The analysis performed in this study is based on empirical data from a large company that develops embedded systems for a global market. The company uses

Table 1. The Project Timeline

Milestones Elapsed Days	Project Analyzed Launched May 2007
MS1	0
MS2	98
MS3	143
MS4	203

Table 2. Milestone Criteria

MS1	Potential features are drawn from the long-term roadmap documents. The initial scope (set of features) is defined and baselined. The scope is then documented and updated after the weekly meeting of the Change Control Board (CCB). The CCB is responsible for adding or removing features from the project plan.
MS2	Features are refined to requirements that are specified, reviewed and approved. Each feature typically contains ten or more requirements from various areas in the products. The feature requirements are forwarded to the design teams who return updated effort estimates.
MS3	Requirements are updated as necessary (based on design team feedback) and the effort estimates are refined.
MS4	The requirements work and design are finished. The final project scope has been negotiated with the development resources and the project is ready to start implementation.

the software product line approach [25] based upon a common code base (referred to here as the platform) for the product line. A platform project follows the stage-gate model [26] with several increments; Milestones (MSs) and Tollgates (TGs) are used to control the project progress. In particular, there are four milestones for requirements management and design before the implementation starts: MS1 through MS4. The scope of a given project is based on a unit called a *feature*, a group of requirements that constitute a functional enhancement to the platform. At the beginning of a project, the feature definition typically contains a functional description and estimates of market value and development effort. To cancel or descope a feature in this context means to permanently remove the feature from the project plan. The discussions in this paper focus on activity within a current project, features re-introduced in a later project are outside of the scope of this work.

The elapsed time information for the projects used in this work are presented in Table 1. The milestone criteria are presented in Table 2. We use the same metrics as in our prior work [12], summarized in Table 3 for convenience.

In this paper we have focused on the top four reasons for feature removal within the data set. The categories are defined as follows.

– *Stakeholder* A stakeholder made a business decision to to add or cancel the feature

Table 3. Metrics

M1	*Number of positive and negative scope changes per time stamp/baseline.* A positive scope change means a feature was added, and a negative scope change indicates a feature was removed. *M1 is not used in the current work and is only included here for completeness.*
M2	*Time to feature removal.* The time from feature introduction until permanent removal.
M3	*Number of state changes per feature.* Number of times that the state of the feature in the scope was changed. In this work, we do not consider the initial inclusion to the scope as a scope change.
M4	*Time to feature addition.* The time from the start of the project until the feature was added.
M5	*Reason for feature removal.* A categorical metric focusing on reasons for removal due to project constraints.

– *Resources* A feature has been removed due to lack of resources
– *Portfolio* A feature has been removed due to changes in the product line portfolio
– *Replaced* A feature has been replaced by an another feature.

The underlying data sets have been transformed, as necessary, for the purposes of this paper to ensure that they are logically correct and consistent across the projects. The analyzed project has 223 features considered during the analysis period. The project contained features that survived from project inception, features that were added during the project, and features that were canceled during the project.

For investigations involving M2, only features that were removed from the scope were included (120 data points). The values for M3 and M4 were calculated for all features in the analyzed project; features that survived the cancellation process were assigned a value of zero to remove them from consideration.

For investigations involving M5, 120 descoping decisions were analyzed and categorized. The entire set of 120 cancellation decisions was used during the M2-M5 and M4-M5 correlation analyses. However, some categories such as "dependent on supplier" or "inadequate feature description" contained such a small number of data points that we focused our analyses on the five categories with the most data points. Moreover, the normalization of the dataset (for example to MS4) further limited the datasets. The specific number of data points for each category is contained in the axis labels (*e.g.* (7dp) means 7 data points) for Figures 1, 2, 3 and 4.

4 Visualizations

The following sub-sections present visualizations designed to assist a project manager in analyzing the requirements scoping process. The visualizations portray the relationships between the (normalized) metric data.

The underlying business decisions that generate the data sets are assumed to be rational in context – the decisions may not be perfect but the decisions were acceptable in the situation. Under this assumption, if a feature is going to be canceled, it is less wasteful of resources to cancel the feature sooner than later. While there may be cases where efforts to promote early feature cancellation may impede innovation, in this product line process the innovation decisions are made prior to the studied scenarios and are based on an ROI analysis rather than technical feasibility. This is not greenfield product development and innovation is incremental and often small in scope (*e.g.* a new software feature as compared to a new hardware platform).

4.1 Interpreting the Visualizations

Box-plots are a non-parametric means for visualizing datasets; they provide greater insight into the data than a simple average value without making any assumptions about the distribution of the underlying data – a factor that makes them particularly useful for smaller data sets.

The box plots used in this work have horizontal bars (whiskers) that identify the most extreme data points for the population (approximately +/-2.7 σ and 99.3% coverage if the data are normally distributed) that are not considered outliers; outliers are individually plotted. The box captures the 25% to 75% range (second and third quartiles) of the data set and the median value is represented by the horizontal line within the body of the box.

In Figure 1, comparing the plot for the entire dataset to the other plots for subsets of the dataset, we note that the plot for the *Resources* category of M5 is similar in size and shape to that for the entire dataset. This shape correlation identifies resource management as the principle contributor to the long-tail of the entire dataset. Five possible causes for the tail are identified in Section 4.2, that may or may not be the cause – the critical factor is that the visualization helps the Project Manager to immediately understand that they should devote their attention to resource management as a source of issues related to *time to cancel* decisions for this project. In comparison, the *time to add* issues in Figure 3 are attributable to the stakeholders rather than resources – again directing the Project Manager's attention in an appropriate manner.

4.2 The Relationship between the Time to Feature Removal (M2) and the Reason for Feature Removal (M5) - Addressing Research Question RQ1

The data set was partitioned based on the Reason for Descope of the feature and the resulting temporal distributions (normalized to MS4) of the top four categories for M5 are presented as box-whisker plots in Figure 1. We see that approximately 40% of the features (29/72) were removed as a result of a stakeholder business decision with the mean time to feature removal (as a result of a stakeholder business decision) approximately 10% of the way into the final milestone of the requirements management process. These characteristics indicate that the feature list was pruned relatively aggressively and relatively quickly.

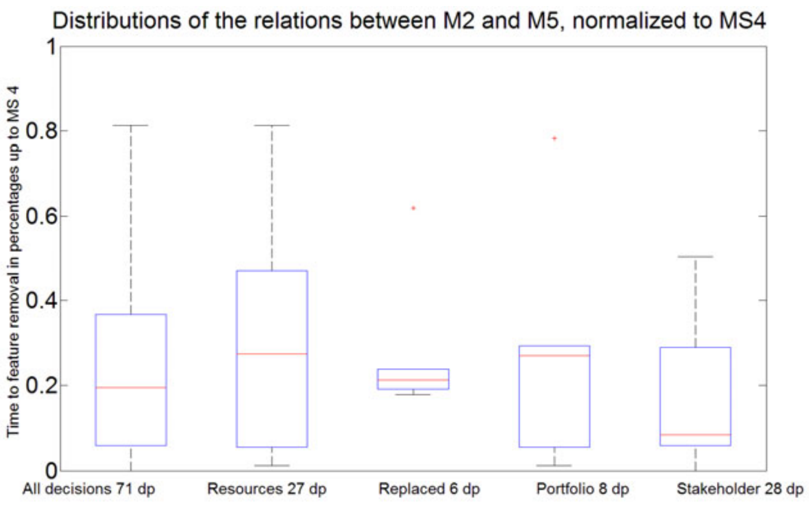


Fig. 1. The Relationship between the time to feature removal (M2) and the reason for feature removal (M5), normalized to MS4

In contrast, approximately 50% of the features (median line of the Resources box in the plot) were removed from scope due to lack of resources by approximately 30% of the way to MS4. However, some features were not removed until almost the end of the project. This is a matter for concern to a project manager for it may be evidence that one or more of the following statements are true (this list is only exemplary and not exhaustive):

– There are difficulties estimating the scope (effort) required for a feature.
– There are difficulties determining what resources are available.
– There are difficulties estimating the contributions of the available resources.
– There are difficulties matching the available resources to the features (resource mismatch).
– The business process (algorithm) used to make the decisions may need improvement.

Simplistically, from the project manager's perspective, all features that are canceled waste scarce resources. It is imperative to make the keep/cancel decision for a feature as soon as possible to minimize this waste. In this regard, the keep/cancel decision could be considered a form of potential defect detection in the sense that a canceled feature is a feature that is no longer deemed appropriate for this feature release. As such, the economic benefits are those identified by Boehm and Basili [27].

Next, the earliest stage of the project was investigated, the time between MS1 and MS2 to see if we could gain any further insight. The data set was partitioned again and only features that have been removed in the time between MS1 and MS2 were kept. The resulting data was normalized to MS2 and the results are shown in Figure 2.

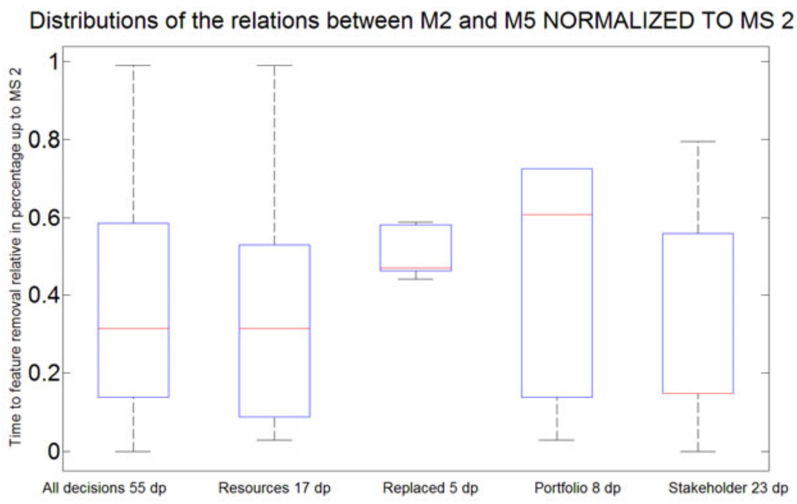


Fig. 2. The Relationship between the time to feature removal (M2) and the reason for feature removal (M5), normalized to MS2

The resulting data represents approximately 76% of the entire data set. We note that all portfolio changes (8/8), almost all replaced or renamed decisions (5/6) and most of the stakeholder business decisions (23/29) were made in this interval. This indicates that the project scope is being reduced in an effective manner, although portfolio changes and reduced or renamed decisions still lag the start of the interval by an average value of approximately 50%.

Only 17 of 27 resource decisions were made in this interval, again with an average delay in excess of 30% of the interval and a tail that extends to the end of the interval. While the situation is not as challenging as it appeared in Figure 1, there does appear to be evidence that improvements could be made.

4.3 The Relationship between the Time to Feature Addition (M4) and the Reason for Feature Removal (M5) - Addressing Rresearch Question RQ2

Figure 3 illustrates the relationship between M4 and M5 for the analyzed project, normalized to MS4 for this project. We note that the medians for each partition show significantly greater diversity than those of Figure 1. The average time to feature introduction was approximately 50% of the timeline. We note that features introduced in the first half of the projects were predominantly cancelled due to lack of resources, replaced or renamed, or portfolio changes. Features introduced after the midpoint were predominantly cancelled as a result of stakeholder business decisons, although there is a much lesser contribution from a lack of resources.

Figure 4 illustrates the relationship between M4 and M5 normalized to MS3 for this project. Again, we note that there is significant diversity in the median

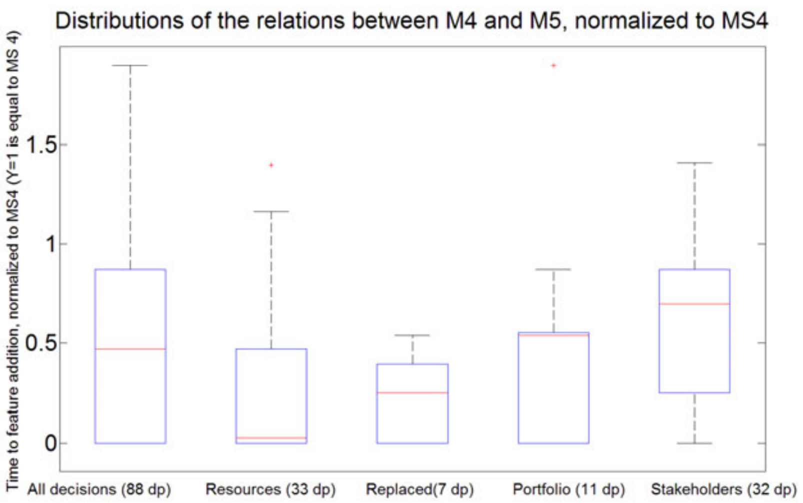


Fig. 3. The Correlation between the Time to feature addition and The Reason for Cancellation, normalized to MS4 of the project

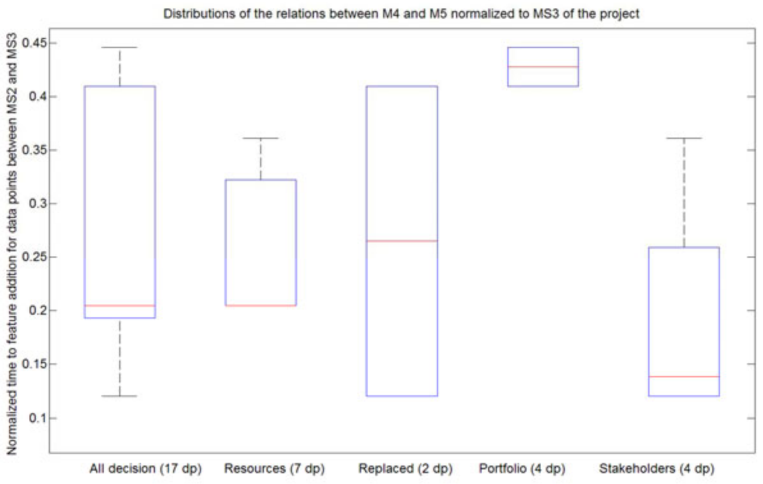


Fig. 4. The Correlation between the Time to feature addition and The Reason for Cancellation, normalized to MS3 of the project

values. However, we are cautious of drawing strong conclusions due to the relatively small data set. There is some indication of a relatively binary decision making process: cancel due to lack of resources or defer to a later date (portfolio changes + stakeholder business decision). We present this visualization to draw attention to the challenges faced with analyzing the tails of distributions in support of management decisions.

5 Threats to Validity

In this section, threats to validity are outlined and discussed based on the classification by Wohlin *et al* [28]. The causal conclusions drawn from our analyses were validated in a number of meetings with practitioners who confirmed that frequent (and sometimes late) scope changes are principally caused by specific market forces. However, more research is needed to validate the causal influence of the different reasons for adding and removing features from the scope of the project. The causal influence for the timing of when the features were included or excluded from the scope of the analyzed projects also requires further investigation.

The proposed visualization do not promote particular reasons for including or excluding features from the scope of the project. Therefore, researcher bias toward (or *fishing* for) a specific justification is minimized. Finally, although the example visualizations show four main reasons for removing features from the scope of the project, there is no theoretical limit to the number of attributes that can be visualized using this technique. As a result, the mono-operation threat for under-representation of the construct is minimized.

With respect to external validity threats, we would like to emphasize that while the investigated scenario has been observed in a specific company, using a specific development paradigm and releasing software products to a specific market, the identified issues of estimation challenges and changing the scope of the project are known and reported in the software engineering literature. We are aware that the approach should be validated in one or more independent contexts, yet we believe that the presented visualizations have sufficient support in the current context that they can be applied, with appropriate caution, in other than studied contexts.

6 Conclusions and Future Work

Software engineering, as an engineering discipline, should be guided by the goal of value creation, measured in terms that count for the enterprise that is investing the resources [9]. In order for software systems to best catalyze their potential for novel value generation, management must maintain a broad perspective to ensure that investments is software artifacts deliver an acceptable return. In this context, deciding which requirements to include into the scope of an upcoming project is crucial for the process of value creation. In a rapidly changing situation, such as MDRE, the failure to quickly cancel features or projects that new information shows are unlikely to succeed is a common example of failing to make a value-optimizing decision [9]. Improving the understanding of the connections between technical decisions and enterprise-level value maximization will enable software engineers and managers to make better choices.

In this paper, we present visualizations that emphasize the relationship between the technical and financial aspects of scoping decisions. Utilizing a particular industrial example, with rapidly changing context and a high degree of uncertainty, we demonstrate methods for analyzing the impact of scoping decision on the financial aspects of the project and the company (RQ1). The visualizations help to understand the drivers of late scope exclusions and inclusions and can assist management efforts to control them (RQ2). The results from this paper support both diagnostic and predictive aspects of decision making [29] and have been expressed in a manner that supports strategic decision management for the project.

In future work, we hope to obtain other data sets that will allow us to further generalize this work. We are particularly interested in investigating whether new reasons for removing features from the scope of the project, other than those identified in this work, can be identified as influences on the scoping process. A larger dataset may enable us to propose a method for minimizing waste incurred by late scope removal; we are particularly interested in temporal optimizations based on cost-benefit analysis or return on investment. Formal statistical analyses of the current dataset are in progress; additional datasets are expected to strengthen the results of these analyses. Finally, the investigation of possible similarities between the defect detection and correction process and the associated costs are considered within the future work agenda.

In the visualization domain, we plan to investigate cost feedback using geometry (such as line thickness proportional to cost) and luminance (proportional to cost). Cost feedback can also be provided using cost as a function of assessed (predicted) risk. Finally, further empirical studies are planned that investigate the utility of the visualizations for other project managers and how the visualizations are interpreted by them.

Acknowledgments

This work is supported by VINNOVA (Swedish Agency for Innovation Systems) within the UPITER project.

References

1. Regnell, B., Brinkkemper, S.: Market–Driven Requirements Engineering for Software Products. In: Engineering and Managing Software Requirements, pp. 287–308. Springer, Heidelberg (2005)
2. Chen, J., Reilly, R.R., Lynn, G.S.: The impacts of speed-to-market on new product success: the moderating effects of uncertainty. IEEE Transactions on Engineering Management 52(2), 199–212 (2005)
3. McPhee, C., Eberlein, A.: Requirements engineering for time-to-market projects. In: Proc. Ninth Annual IEEE Int. Conf. and Workshop on the Eng. of Computer-Based Systems, pp. 17–24 (2002)
4. Schumpeter, J.: Capitalism, Socialism and Democracy. Harper (1942)
5. Karlsson, J., Ryan, K.: A cost-value approach for prioritizing requirements. IEEE Software 14(5), 67–74 (1997)
6. Schmid, K.: A comprehensive product line scoping approach and its validation. In: 24th Int. Conf. on Soft. Eng (ICSE 2002), pp. 593–603 (2002)

7. Regnell, B., Beremark, P., Eklundh, O.: A market–driven requirements engineering process – results from an industrial process improvement programme. Requirements Engineering Journal 3(2), 121–129 (1998)
8. Institute, P.M.: A Guide To The Project Management Body of Knowledge, 4th edn. Project Management Institute (2009)
9. Boehm, B., Sullivan, K.: Software economics: a roadmap. In: Proc. of the Conf. on The Future of Soft. Eng., ICSE 2000, Limerick, Ireland, pp. 319–343. ACM, New York (2000), http://doi.acm.org/10.1145/336512.336584, doi:10.1145/336512.336584
10. Wnuk, K., Regnell, B., Karlsson, L.: Visualization of feature survival in platform-based embedded systems development for improved understanding of scope dynamics. In: Third Int. Workshop on Req. Eng. Visualization (REV 2008), pp. 41–50 (2008)
11. Wnuk, K., Regnell, B., Karlsson, L.: Feature transition charts for visualization of cross-project scope evolution in large-scale requirements engineering for product lines. In: Forth Int. Workshop on Req. Eng. Visualization (REV 2009), pp. 89–98 (2009)
12. Wnuk, K., Regnell, B., Karlsson, L.: What happened to our features? visualization and understanding of scope change dynamics in a large-scale industrial setting. In: Proc. of the 17th IEEE Int. Req. Eng. Conference (RE 2009), pp. 89–98 (2009)
13. Robson, C.: Real World Research. Blackwell Publishing, Malden (2002)
14. van de Weerd, I., Brinkkemper, S., Nieuwenhui, R., Versendaal, J.A.: A reference framework for software product management. Technical Report UU-CS, vol. 2006(014), Utrecht (2006)
15. Karlsson, J.: A Systematic Approach for Prioritizing Software Requirements. Doctorial Dissertation, PhD thesis, Linköping University, Sweden (1998)
16. Jorgensen, M., Shepperd, M.: A systematic review of software development cost estimation studies. IEEE Transactions on Software Engineering 33(1), 33–53 (1992)
17. Svahnberg, M., Gorschek, T., Feldt, R., Torkar, R., Saleem, S.B., Shafique, M.U.: A systematic review on strategic release planning models. Inf. Softw. Technol. 52, 237–248 (2010), http://dx.doi.org/10.1016/j.infsof.2009.11.006, doi:10.1016/j.infsof.2009.11.006
18. Gorschek, T., Gomes, A., Pettersson, A., Torkar, R.: Market-driven requirements engineering process maturity model. Journal of Software Maintenance tba (tba 2010) (2010)
19. Khurum, M., Gorschek, T.: A systematic review of domain analysis solutions for product lines. J. Syst. Softw. 82, 1982–2003 (2009), doi:10.1016/j.jss.2009.06.048
20. Carlshamre, P., Sandahl, K., Lindvall, M., Regnell, B., Nattoch Dag, J.: An industrial survey of requirements interdependencies in software product release planning. In: Proceedings of the Fifth IEEE Int. Symp. on Req. Eng., RE 2001, p. 84. IEEE Computer Society, Washington, DC, USA (2001)
21. Wiess, D.M., Lai, C.T.R.: Software Product Line Engineering. Addison-Wesley, Reading (1999)
22. Green, P.E., Krieger, A.M.: Models and heuristics for product line selection. Marketing Science 4(1), 1–19 (1985), http://www.jstor.org/stable/183706
23. Tufte, E.: Envisioning Information. Graphics Press LLC (1990)
24. Gotel, O.C.Z., Marchese, F.T., Morris, S.J.: On requirements visualization. In: Proc. of the Second Int. Workshop on Req. Eng. Visualization (REV 2007), pp. 80–89 (2007)

25. Pohl, K., Bockle, G., van der Linden, F.J.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer, Heidelberg (2005)
26. Cooper, R.G.: Stage-gate systems: A new tool for managing new products. Business Horizons 33(3), 44–54 (1990)
27. Boehm, B., Basili, V.: Software defect reduction top 10 list. IEEE Computer 34(1), 135–137 (2001)
28. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslen, A.: Experimentation in Software Engineering An Introduction. Kluwer Academic Publishers, Dordrecht (2000)
29. Pomerol, J.C.: Scenario Development and Practical Decision Making under Uncertainty: Application to Requirements Engineering. Requirements Engineering 3, 3–4 (1998)

Variability-Based Release Planning

Samuel Fricker[1] and Susanne Schumacher[2]

[1] Blekinge Institute of Technology, School of Computing
Campus Gräsvik, 371 79 Karlskrona, Sweden
samuel.fricker@bth.se
[2] Zurich University of the Arts
Ausstellungsstrasse 60, 8005 Zurich, Switzerland
susanne.schumacher@zhdk.ch

Abstract. A release plan defines the short-term evolution of a software product in terms of development project scope. In practice, release planning is often based on just fragmentarily defined requirements. Current release planning approaches, however, assume that a requirements catalogue is available in the form of a complete flat list of requirements. This very early commitment to detail reduces the flexibility of a product manager when planning product development. This paper explores how variability modeling, a software product line technique, can be used to plan, communicate, and track the evolution of a single software. Variability modeling can reduce the number of decisions required for release planning and reduce the information needed for communicating with stakeholders. An industrial case motivates and exemplifies the approach.

Keywords: release planning; variability; features.

1 Introduction

Release planning is used to plan the development of software products by allocating requirements to development projects [1]. Thereby, the evolving product is aligned with market and stakeholder needs, company objectives, and constraints such as time, resources, and legacy. Release planning is also a central concern in iterative development, where the scope of iterations, rather than projects, is defined [2].

Release planning involves the following steps [3]. Requirements are elicited and specified based on an understanding of stakeholders, organizational environment, and culture [4]. That same understanding is a basis for defining criteria [5] to evaluate and prioritize requirements [6]. The priorities are then used to scope releases by allocating requirements to development projects. The resulting release plans are implemented, delivered, and analyzed with post-release reflections [7].

Release planning is challenging in practice because of the typically large number of requirements [8]. These requirements often are not properly defined and detailed due to the limited effort that can be invested before a development project is funded. These challenges contradict with the assumptions taken by current release planning approaches. For example, requirements catalogues cannot be considered complete or correct until their effect on solution architecture and project planning has been understood and agreed [9, 10].

This paper proposes to ease the release planning problem not by improving prioritization algorithms, but by structuring the requirements catalogue. The approach is based on analyzing variability [11] by utilizing AND, OR, and REQUIRES relationships between requirements [12]. Such variability is a particular form of decision options [13] utilized to define the scope of product releases.

The approach works with a body of only fragmentarily and vaguely defined requirements and can be used in continuous agile product management processes [14]. Simple abstractions allow incremental aggregation and extension of requirement catalogues, hence address the scalability problem [15]. The abstractions further allow expressing the essence of release decisions to support the dialogue of the product manager with stakeholders, hence addressing the problem of limited trust that results from mathematical, black box-oriented prioritization [15].

This paper motivates and describes the variability-based release planning approach. An industrial case of an organization that transited from flat requirements list-based release planning to variability-based release planning is used to illustrate the approach. The paper is structured as follows. Section 2 describes background and motivation. Section 3 introduces variability-based release planning. Section 4 provides a short discussion and concludes.

2 Background and Motivation

Current release planning approaches require a flat and complete catalogue of requirements that are evaluated, prioritized, and selected for implementation [16]. Known approaches include manual techniques such as top ten, numerical assignment, raking, and 100$-test [6], and computer-based techniques such as Integer Linear Programming [17] and the Analytical Hierarchy Process [18].

We investigated release planning in a company that offered software as a service for managing media such as text, sound, pictures, and movies. The product manager and important stakeholders, henceforth called "release planners", regularly planned software releases that corresponded to small and large version increments.

The requirements catalogue was managed in a word processor document and used as a basis for release planning. It contained 108 requirements. Some of them were specified with a few words in a declarative manner. Others were specified in detail with descriptions of up to 245 words. The requirements were grouped into 12 sections and 19 subsections or themes. The grouping, however, did not show a relationship with requirements allocation to development releases. In average, a group contained 3.6 requirements and was allocated to 1.93 releases.

Current release planning approaches would have expected the release planners to evaluate every detailed requirement. For example the planners would have compared each requirement individually with other requirements by posing questions such as "are *thumbnails of variable sizes* (requirement 11.3) more important than *storage of search results* (requirement 8.1)?" Such evaluation would have led to detailed evaluation results. However, it also would have increased the risk of losing the understanding of the big picture and sub-optimizing details irrelevant in the given product evolution stage.

Considering requirements out of their larger context also would have increased the risk of misunderstandings: *When would thumbnails be shown? For what purpose? Which sizes? What (photos, videos, documents, etc.) would be depicted by these thumbnails?* It is evident, independent of the applied prioritization technique, that the lack of a common understanding leads to considerably different interpretations of criteria such as importance, urgency, dependencies, implementation cost, and risk.

These problems motivated us to indentify alternative release planning approaches based on the following criteria. *Minimalism*: release planning should involve as few decisions as possible to reduce effort and likelihood of errors. *Traceability*: a release plan should be traceable to roadmaps to align long-term with short-term planning. *Saliency*: a release plan should abstract detail and contain just salient information to support negotiations and communication. *Evolution*: a release plan should change to reflect evolving knowledge and progress.

A solution that respects these criteria moves the release planning problem away from mathematical optimization towards supporting the dialogue with stakeholders. A minimal number of release planning decisions enables stakeholders to reflect on alternatives, for example by building a mental model of the decision options and by exploring what-if scenarios. Traceability to roadmaps allows stakeholders to understand the impact of decisions on their stakes and commitments regarding the software product. For example they need to know whether they need and can hold promises given during long-term planning. Saliency allows focusing on the big picture, e.g. by discussing just the most important decision-making topics and options during steering committee meetings and by communicating just the important information for marketing purposes. A release plan finally, is in continuous evolution. It is expanded when new requirements are elicited, refined when solutions are explored, and its status changed when development progresses.

3 Variability Modeling for Release Planning

Feature trees are a widespread approach to document and analyze variability of software products [11]. They are used to specify how features vary for the products of a product line (variability in space). Applied for release planning, variability models can be used for defining the evolution of software (variability in time) [19]. How feature trees are utilized for release planning, has not been researched yet though [1].

The here proposed approach structures requirements with such a feature tree. The tree's root refers to the central parts of the solution: the architecture and infrastructure assets to be developed before value-adding features can be added. The branches capture variability by referring to features that are enhanced incrementally with extending sub-features. Each feature groups requirements with an AND relationship [12]. These requirements are intended to be developed together in the same development increment. An enhancing feature stands in a REQUIRES relationship with the enhanced feature. Alternative enhancements stand in an OR relationship.

A product manager constructs a feature tree by first grouping requirements into coarse features and then building feature vectors [20] that connect the root of the feature tree with leafs. Feature vectors are built iteratively by extracting requirements from given features into extending sub-features [21]. The feature vector-building process stops when no requirements can be extracted without making the concerned super-feature useless.

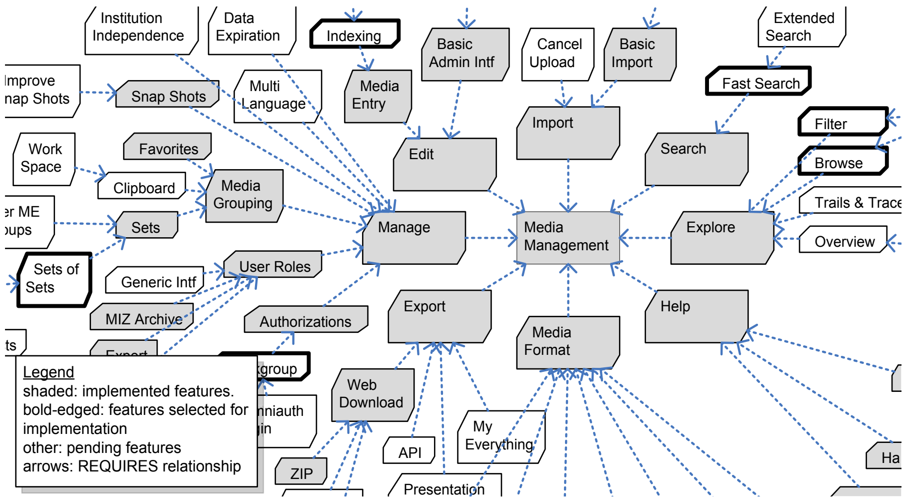


Fig. 1. Extract of the media management solution's feature tree (notation [21]). Allocation of requirements to features (AND relationship) not shown

The product manager uses the feature tree for release planning, communication, and controlling. Initial development starts with the root of the tree. Release planning involves selecting those features that are not implemented yet from those that are connected with already implemented or planned features (connectivity rule). Implementation progress is documented by tagging features as being implemented. The tree is used in project status and in steering committee meetings as an instrument to illustrate plans and progress. Emerging requirements, for example discovered during elicitation activities or late in the development process, are added to existing non-implemented features or as new leaf features to the tree.

Figure 1 shows the feature tree that the product manager of the media management solution created and continuously used for planning, communicating and controlling implementation progress. *Media Management* was the root and referred to the basic software infrastructure. Each node selected for implementation was marked with a bold edge and the contained requirements were entered into the backlog of the concerned development increment. The feature *Indexing*, for example, contained 12 requirements (AND relationship). The feature *Indexing* was only selected for implementation after *Media Entry* was implemented (REQUIRES relationship). Concluded feature implementation was marked by shading the corresponding nodes. In an earlier development stage, *Media Entry* stood in competition with *Basic Admin Intf* (OR relationship): none, one of them, or both features could be implemented as an enhancement of *Edit*. The chain *Edit* ← *Media Entry* ← *Indexing* represented one of the feature vectors of the media management solution.

A major challenge concerned the implementation of the approach. No tools were available that were integrated into the development environment. Ad-hoc tools were used instead. The feature tree was specified with diagramming software and the requirements–feature allocation (AND grouping of requirements) with a word processor document. The development project tracked requirements and implementation progress with a task management solution. The product manager ensured consistency in meetings with the development team.

4 Discussion and Conclusions

This paper has introduced an approach, variability-based release planning, that simplifies release planning by structuring the underlying requirements catalogue. It utilizes feature trees and feature vectors to structure requirements with AND, OR, and REQUIRES relationships. It uses feature extraction as the basic technique to construct the tree and a simple lifecycle model to track and plan software development. An industrial case was described to motivate and exemplify the approach.

Variability-based release planning reduces the complexity of release planning compared with the traditional approaches that are based on a flat list of requirements. The feature tree allows abstracting from detailed requirements to groups of requirements. It provides a graphical representation that allows stakeholders to develop a mental model of the decision options. A planned release can now be described by referring to a few features instead of a potentially large and incomplete set of requirements. The described tree construction, planning, and controlling approach is used to evolve the information base for release planning. Containers for adding requirements that are discovered late in the development process allow dealing with requirements incompleteness. The connectivity rule provides support to rapidly identify those features that are candidates for release planning. Release planning scenarios can be analyzed by exploring the OR relationships between sub-features. Roadmaps can be traced to by referring with important features to agreed activities of the company, for example by utilizing appropriate naming.

Variability-based release planning has not been integrated with other product management and development activities yet. Future research should investigate how the approach relates to upstream techniques for requirements triage such as the requirements abstraction model [4] and downstream techniques for requirements communication such as handshaking with implementation proposals [9]. Empirical research is needed to better understand benefits, and limitations of variability-based release planning in practice, for example compared to other release planning approaches.

References

1. Svahnberg, M., Gorschek, T., Feldt, R., Torkar, R., Bin Saleem, S., Usman Shafique, M.: A Systematic Review on Strategic Release Planning Models. Information and Software Technology 52, 237–248 (2009)
2. Cohn, M.: Agile Estimating and Planning. Prentice-Hall, Englewood Cliffs (2006)
3. Amandeep, N.F.N.G., Ruhe, G., Stanford, M.: Intelligent Support for Software Release Planning. In: Bomarius, F., Iida, H. (eds.) PROFES 2004. LNCS, vol. 3009, pp. 248–262. Springer, Heidelberg (2004)
4. Gorschek, T., Wohlin, C.: Requirements Abstraction Model. Requirements Engineering 11(1), 79–101 (2006)
5. Wohlin, C., Aurum, A.: What is Important when Deciding to Include a Sotware Requirement into a Project or Release. In: International Symposium on Empiricial Software Engineering (2005)

6. Berander, P., Andrews, A.: Requirements Prioritization. In: Aurum, A., Wohlin, C. (eds.) Engineering and Managing Software Requirements. Springer, Heidelberg (2005)
7. Karlsson, L., Regnell, B., Karlsson, J., Olsson, S.: Post-Release Analysis of Requirements Selection Quality - An Industrial Case Study. In: 9th International Workshop on Requirements Engineering: Foundation for Software Quality, RefsQ 2003 (2003)
8. Regnell, B., Svensson, R.B., Wnuk, K.: Can we beat the complexity of very large-scale requirements engineering? In: Rolland, C. (ed.) REFSQ 2008. LNCS, vol. 5025, pp. 123–128. Springer, Heidelberg (2008)
9. Fricker, S., Gorschek, T., Byman, C., Schmidle, A.: Handshaking with Implementation Proposals: Negotiating Requirements Understanding. IEEE Software 27(2), 72–80 (2010)
10. Fricker, S., Glinz, M.: Comparison of Requirements Hand-Off, Analysis, and Negotiation: Case Study. In: 18th IEEE International Requirements Engineering Conference, Sydney, Australia (2010)
11. Schobbens, P.-Y., Heymans, P., Trigaux, J.-C., Bontemps, Y.: Generic Semantics of Feature Diagrams. Computer Networks 51, 456–479 (2007)
12. Carlshamre, P., Sandahl, K., Lindvall, M., Regnell, B.: Nattoch Dag, J.: An Industrial Survey of Requirements Interdependencies in Software Product Release Planning. In: 5th IEEE International Symposium on Requirements Engineering (2001)
13. Haberfellner, R., Nagel, P., Becker, M., Büchel, A., von Massow, H.: Systems Engineering: Methodik und Praxis, 11th edn. Verlag Industrielle Organisation (2002)
14. Vlaanderen, K., Jansen, S., Brinkkemper, S., Jaspers, E.: The Agile Requirements Refinery: Applying SCRUM Principles to Software Product Management. In: 3rd International Workshop on Software Product Management (2009)
15. Lehtola, L., Kauppinen, M.: Suitability of Requirements Prioritization Methods for Market-driven Software Product Development. Software Process Improvement and Practice 11, 7–19 (2006)
16. Carlshamre, P.: Release Planning in Market-Driven Software Product Development: Provoking an Understanding. Requirements Engineering 7, 139–151 (2002)
17. Ruhe, G., Saliu, M.O.: The Art and Science of Software Release Planning. IEEE Software 22(6), 47–53 (2005)
18. Karlsson, J., Ryan, K.: A Cost-Value Approach for Prioritizing Requirements. IEEE Software 14(5), 67–74 (1997)
19. Pohl, K., Böckle, G., van der Linden, F.: Software Product Line Engineering: Foundations, Principles and Techniques, 1st edn. Springer, Heidelberg (2005)
20. Nejmeh, B., Thomas, I.: Business-Driven Product Planning Using Feature Vectors and Increments. IEEE Software 19(6), 34–42 (2002)
21. Stoiber, R., Glinz, M.: Feature Unweaving: Efficient Variability Extraction and Specification for Emerging Software Product Lines. In: 4th International Workshop on Software Product Management (IWSPM 2010), Sydney, Australia (2010)

EPIC 2011: Third Workshop on Leveraging Empirical Research Results for Software Business Success

Maya Daneva[1] and Andrea Herrmann[2]

[1] University of Twente, The Netherlands
[2] Axivion, Germany

For many companies, software development is their core business process. For this process to be economically viable, it is not enough that software companies deliver software products that satisfy customers´ written specification. Software businesses also deem other requirements important as to deliver in time and on budget, to increase developers´ satisfaction and to optimize their delivery processes and reduce waste.

Collective efforts by software engineering practitioners, consultants and researchers have yielded a huge variety of solutions for improving software processes, products and services. While it is generally known that the suitability and effectiveness of most of these solutions depend on the context where they are applied, only few empirical studies were done to uncover how the current process/product/service-focused approaches used in software businesses yield outcomes that are aligned to the business goals of these organizations. With few exceptions, little is known about the empirical evidence that can possibly confirm or disconfirm the claims of effectiveness of different commercially viable approaches that solve particular process, product or service related problems.

The primary goal of the first EPIC workshop was twofold: to initiate (1) the conversation of leveraging empirical research for software business success and (2) the process of creating a forum and a community to debate the need and value of using empirical/evidence-based approaches to researching aspects of software processes, products and services that contribute to software business success. An outcome of the workshop is the LinkedIn group, EPIC FORUM.

The second EPIC event was a panel session that defined some roadblocks to the collaboration of software business practitioners and researchers, and some solutions that worked.

The third workshop builds upon these first results and will extend the discussion on state-of-the art good practices for empirical research that adds value to both small and large software companies.

IWSECO 2011: Third International Workshop on Software Ecosystems

Slinger Jansen[1], Jan Bosch[2], Faheem Ahmed[3], and Piers Campbell[3]

[1] Utrecht University, The Netherlands
[2] Intuit Inc, USA
[3] United Arab Emirates University

Software vendors no longer function as independent units, where all customers are end-users, where there are no suppliers, and where all software is built in-house. Instead, software vendors have become networked, i.e., software vendors are depending on (communities of) service and software component suppliers, value-added-resellers, and pro-active customers who build and share customizations. Software vendors now have to consider their strategic role in the software ecosystem to survive. With their role in the software ecosystem in mind, software vendors can become more successful by opening up their business, devising new business models, forging long-lasting relationships with partnership networks, and overcoming technical and social challenges that are part of these innovations. The focus of the first workshop was the definition of the research field. The second workshop's focus was the 'ideal' architecture of a software platform. The third workshop on software ecosystems focuses on the management of software ecosystems, i.e., how a software vendor can manage its network of partners, developers, service deliverers, and other third parties that play a role in the software ecosystem.

Typically, software vendors have several instruments available to them for managing their ecosystem, such as the creation of partnership models or the introduction of component and service certification. The effects of these decisions on the software ecosystem have not yet been made measurable, which can be considered one of the main challenges of the field of software ecosystems.

A software ecosystem is a set of actors functioning as a unit and interacting with a shared market for software and services, together with the relationships among them. These relationships are frequently underpinned by a common technological platform or market and operate through the exchange of information, resources and artifacts. Several challenges lie in the research area of software ecosystems. To begin with, insightful and scalable modeling techniques for software ecosystems currently do not exist. Furthermore, methods are required that enable software vendors to transform their legacy architectures to accommodate reusability of internal common artifacts and external components and services. Finally, methods are required that support software vendors in choosing survival strategies in software ecosystems.

The workshop aims to further increase the body of knowledge in this specific area of software reuse and software engineering by providing a forum to exchange ideas and discuss state-of-the-art results. It will build and shape the community of leading practitioners and research experts. Given the relevance of software ecosystems, and the rather unexplored scientific and industry contribution in this field, the workshop will deliver a state-of-the-practice overview of the available knowledge on software ecosystems, as well as an overview of challenges for further research.

Author Index

Ahmed, Faheem 188

Bosch, Jan 188
Buglione, Luigi 125
Buxmann, Peter 154

Callele, David 168
Campbell, Piers 188
Cocco, Luisanna 56
Concas, Giulio 56

Damiani, Ernesto 125
Daneva, Maya 187

Finkelstein, Anthony 1
Frank, Lauri 70, 110
Frati, Fulvio 125
Fricker, Samuel 181
Frühwirth, Christian 85

Helander, Nina 70
Herrmann, Andrea 187
Hu, Shuangzeng 98

Jansen, Slinger 17, 44, 154, 188

Kabbedijk, Jaap 44
Kauppinen, Marjo 32, 140
Komssi, Marko 140

Luoma, Eetu 70, 110

Mannaro, Katiuscia 56
Marchesi, Michele 56
McNaughton, Rod B. 98

Oltolina, Sergio 125

Palomäki, Pirkka 140
Peltonen, Juhana 85
Popp, Karl Michael 154

Riepula, Mikko 110
Rönkkö, Mikko 85
Ropponen, Matti 140
Ruffatti, Gabriele 125

Schumacher, Susanne 181
Selin, Joona 2

Tyrväinen, Pasi 2

van de Zande, Tommy 17
Viljainen, Martti 32

Wnuk, Krzysztof 168