Software Business: A Short History and Trends for the Future

KATARIINA YRJÖNKOSKI, Tampere University,
HANNU JAAKKOLA, University of Lapland,
KARI SYSTÄ, Tampere University,
TOMMI MIKKONEN, University of Helsinki,
JAAK HENNO, Tallinn University of Technology

During its 70 years of existence, the software business has been following an evolution curve that can be considered typical for several fields of industrial businesses. Technological breakthroughs and innovations are typically seen as enablers for business evolution in the domain of technology and innovation management. Software, data collection, and data analysis represent a greater and greater part of the value of products and services, and today, their role is also becoming essential in more traditional fields. This, however, requires business and technology competences that traditional industries do not have. The transformation also enables new ways of doing business and opens the field for new kinds of players. Together, all this leads to transformation and new possibilities for the software industry. In this paper we study the overall trajectory of the software business, and then offer some viewpoints on the change in different elements of business models.

1. INTRODUCTION

Software development and the software business have evolved over the past decades. There have been major changes on just about every level of operations. The most well-known change concerning development work is from linear development models like waterfall to agile and lean software development and further to Continuous Deployment [Royce 1970; Olsson et al. 2012]. This change is driven by a better understanding of the difficulties of setting detailed requirements for abstract artifacts like software in advance.

This is not, however, the only change driving the changes in software development and the software business. In this paper we explore technology-related breakthroughs that have disrupted software. Then we take a look at the concept of the business model in the domain of technology management. Finally, we describe how the value creation and selected components of business models have changed. The changes are discussed from different points of view – client, developer, and supplier. The business model is seen widely in the sense of "a way of doing business" – combining these views as its components. This paper proceeds as follows:

1. Defining the industry evolution. The evolution of the industry is defined and divided into three phases based on technology breakthroughs.
2. Reviewing business models in the software business. We take a brief look at general elements of business models in the context of the software business.
3. Reflecting the elements of business models in the industry evolution.

Authors' addresses: Katariina Yrjönkoski, Tampere University, Pori Campus, P.O.Box 300, FI-28101 Pori, Finland; email: katariina.yrjonkoski@tuni.fi; Hannu Jaakkola, University of Lapland, P.O.Box 22, FI-96101 Rovaniemi, Finland; email: hannu.jaakkola@iki.fi; Kari Systä, Tampere University, Korkeakoulunkatu 1, FI-33720 Tampere, Finland; email: kari.systa@tuni.fi. Tommi Mikkonen, University of Helsinki, P.O 68, FI-00014 University of Helsinki, Finland; email: tommi.mikkonen@helsinki.fi; Jaak Henno, Tallinn University of Technology, Ehitajate tee 5, 19086 Tallinn, Estonia; email: jaak.henno@ttu.ee

In: Z. Budimac (ed.): Proceedings of the SQAMIA 2018: 7th Workshop of Software Quality, Analysis, Monitoring, Improvement, and Applications, Novi Sad, Serbia, 27–30.8.2018. Also published online by CEUR Workshop Proceedings (http://ceur-ws.org, ISSN 1613-0073)

4. Finally, we discuss the changes of the industry evolution from different viewpoints.

We have divided the history of the software business into phases. The transitions between the phases are driven by the technology-related evolution:

1. Hardware-driven ecosystems. Software was developed for specific hardware. Even though some high-level programming languages were used and development of portable and manufacturer-specific hardware would have been possible, the ecosystems were formed around manufacturers. Applications were usually written by the manufactures or their partners.
2. Open eco-systems driven by open interfaces. Largely driven by PCs, open interfaces to hardware and software systems started to appear. This led to the appearance of independent software vendors that conducted their business independently from hardware vendors.
3. Diversification of software business models. The Internet together with the pervasiveness of computing capacity has diversified software business models. The consumers have become end users and sometimes also customers of the software business. This development has been enabled by three technical drivers: namely, affordable devices, the Internet as a delivery mechanism, and virtualization with cloud computing. These enablers are heavily interlinked and have an effect on each other.

The evolution of the software business has been enabled by the above technological development steps and affected by related trends. In this paper we review this evolution of the software industry, the kind of transformation there has been and analyze the key trends behind the transformation from one business model to another. The goal is to understand the new developments in the business domain in the light of the core competencies of software companies as well as their customers.

The rest of the paper is organized as follows: in Section 2, we review the trajectory of the software industry and describe the biggest transitions within the business. In Section 3 we take a look at the concept of business models within the software industry to answer the question "What has happened?" In Section 4 we focus on observing the change in business model elements from different viewpoints; this answers the question "Why have the changes happened?" Then, in Section 5, we predict some trends for the future – to answer the question "Where are we going?" - and in Section 6, we provide a summarizing discussion. Finally, towards the end of the paper, we draw some final conclusions in Section 7.

2. EVOLUTION OF THE SOFTWARE INDUSTRY - A HISTORICAL PERSPECTIVE

2.1 The Era of Hardware Driven Software

The history of modern computers has its roots in the middle of the 1940s. Computer equipment – hardware and peripheral devices – has always played the role of enabler for software engineers. During the first decades, the development of software was directed by low performance power and by the modest memory capacity of computers, as well as the poor technology of mass memories. At that time, software engineers had to adapt their work to the demands of the equipment; part of memory management (segmentation) had to be included in the code and code optimization had to be executed to guarantee its better performance. From the software development point of view, we can separate two eras. From the 1940s to the 1970s, software development was done in accordance with the conditions of the device – software was adapted in the equipment (hardware as an enabler). Since then, the situation has been accelerating in the opposite direction thanks to lowering hardware prices and new kinds of scalable infrastructures (cluster, cloud).

Likewise, the use culture of computers has changed significantly in the course of the decades. The era from the 1940s to the 1950s can be described as the era of computer laboratory use, in which computers were used mainly for technical calculations by a few research-intensive organizations. The era of closed shop mainframe computers for common use started in the 1950s; computers were separated from their users and also from software developers. Computers were rare and expensive; only large organizations had an opportunity to acquire them. The period of commonly available computing started in the 1960s;

first in big companies and spreading to smaller companies gradually in the form of minicomputers. Time-sharing operating systems in the 1970s opened the "closed shops" up to users and software developers (operating system as an enabler). Computers could be used via terminals for direct access to the applications for users and computer resources for application developers. This era could be called the era of centralized distribution, because the computers were still located in closed machine halls and the terminals were connected to them with fixed cable connections. The first commercial personal computers were adapted for business use at the end of the 1970s and early 1980s. This can be considered as the birth of the era of distributed computing resources, which is still going on. This progress first started the era of unmanaged distribution: computing power was available at a low price and allowed the satisfaction of individual needs. Networking (first Ethernet, then a variety of other network technologies) of computers returned the culture back to its roots – the era of managed distribution started in the early 1990s and still continues in its enriched form called the era of cloud technologies.

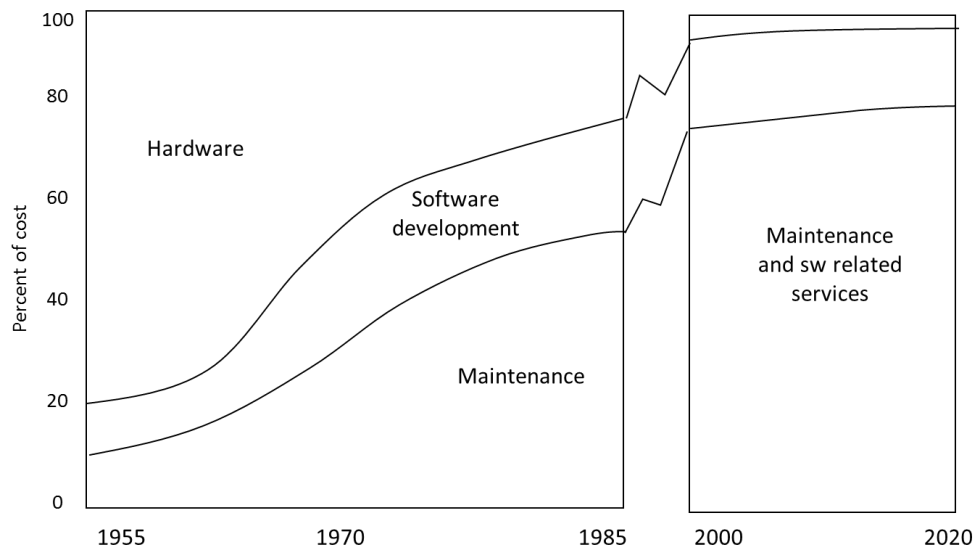

Fig. 1. Computer–based information system trends [Boehm 1976].

The old prediction made by Boehm in 1976 indicates the liberation of information systems from hardware dominance (Figure 1). The original figure (left side) is continued by the authors to cover the period from 2000 to 2020. The aim is to point out the constantly falling importance of device costs in information systems development and the transfer of costs to maintenance and especially to the related, use-time services. These aspects will be discussed later in this paper.

2.2 The Era of Open Ecosystems Driven by Open Interfaces

Liberation of Software Development. The first microprocessors had been launched in the late 1960s. Personal computers brought computing power available at a low price to satisfy the individual needs of computing. Since the late 1970s, the microcomputer evolution has dramatically increased the quantity of software produced and the number of companies involved in software development. These changes together kicked off the software revolution: software started to appear everywhere, and can nowadays be found not only in personal computers but in homes, domestic appliances, public services, offices, and cars. [Fayad et al. 2000]

Until then, the synchronous development of hardware and software gave few opportunities for software developers to iterate over the available features – as everything was basically controlled by hardware. Until then, even software written in high-level programming languages tended to depend on hardware

since the evolution of and interface to the new hardware features was controlled. The development platforms were also provided by the manufacturers – the ecosystem was more or less under the tight control of the computer manufacturers. Due to the need for hardware-specific optimizations, developers had to have a deep understanding of the underlying hardware and thus had to specialize in certain hardware-driven ecosystems. In addition, applications became optimized for specific hardware because of these optimizations and the lack of system libraries that would be the same for all types of hardware.

The key invention that liberated software developers from the dominance of hardware was standard interfaces, offered by an operating system that acted as an abstraction layer between a piece of hardware and the software it runs. Furthermore, while some operating systems are hardware-specific even today, operating systems such as Unix and Linux can be run on various hardware platforms.

The ability to write a program against a well-defined abstraction – given in the form of a stable interface – meant that the same approach could also be applied to contexts other than hardware only. Moreover, software systems could be partitioned so that some parts would come from one vendor, and others from someone else, based on open interfaces that would act as a contract between the subsystems. This progress was called by Barry Boehm [Boehm 2006; 2006a] in his ICSE 2006 keynote presentation the transfer towards "*complex systems of systems*". It points out the situation in which *interfaces and collaboration between software assets* become dominant factors in software development (*interface as an enabler*).

The emergence of open interfaces had many consequences. To begin with, the size of a software system was no longer limited by the size of the team that could be employed to build it, but libraries and other standard subsystems such as databases could be used off-the-shelf. Furthermore, the development cycle of new software features was no longer entangled with that of hardware, meaning that it was possible to start developing software at a pace that was better suited to meet the needs of the end user. This meant that hardware and its requirements no longer dictated software requirements, but rather the focus was placed on end-user requirements and needs.

While open interfaces have become an important enabler for business, they have also become a business tool for the companies that define the interfaces. With a public interface, the developer community can be divided into externals that can only use the public open interface and an inner circle that use functionalities beyond the public interface. Furthermore, the implementers of a public interface may be subject to specific licensing conditions.

From projects to product business When open software ecosystems appeared, the business started to be organized mainly according to two different business paradigms: the software project based business model and software product based business model. Consequently, software was mainly developed as unique systems for each organization, and very little standardization existed. In the ISO/IEC 12207 Systems and software engineering – Software life cycle processes standard, "project" is defined as "an endeavor with defined start and finish dates undertaken to create a product or service in accordance with specified resources and requirements" [ISO/IEC 2008]. *Project business* in general is the part of business that relates directly or indirectly to projects, with the purpose of achieving the objectives of a firm or several firms [Artto & Wikström 2005]. The end result for the customer is tailor-made software that is individually installed on a dedicated server. In the project model, pricing is typically based on total work, and that also limits selling this kind of software to Business-to-Consumer markets.

When the industry started to grow and the consumer market opened up, suppliers noticed that once software had been built, stabilizing and re-selling it could lead to significant economically benefits. Software is built both for increased selling volumes and for better profit margins. These observations led to the birth of the *software product business*: software is delivered in a similar form to every customer. Both the software and all the marketing, delivery, materials, support, and services are productized. The term "Commercial off-the-shelf" (COTS) is also used. According to Wikipedia, this means the purchase of packaged solutions that can be bought "as is" and then adapted to satisfy the needs of the purchasing organization, rather than the commissioning of custom-made solutions. When a software company

expands its platform outside organizational platforms, the model may be called the software ecosystem approach. [Bosch 2009]. While the two models, the project- and product-based business, are seemingly at the different ends of the spectrum of the software business, they surprisingly share many similarities in the actual software development itself. Today, both are done in an agile, customer-driven development, where either a direct customer or an imaginary customer played by, e.g., a product manager, feeds user requirements into the development process. Thus, there is a clear feedback loop from end users to the development, no matter what the business model is – the difference is often only in the monetization model.

2.3 The Era of Diversification in the Software Business

The technical drivers of the diversification discussed in Section 1 were affordable devices, the Internet as the delivery mechanism, and virtualization in the cloud. The consequences of these drivers will be discussed below.

Affordable devices Most office desks have PCs or laptops, most homes have PCs and game consoles. The majority of consumers in developed countries carry mobile devices like tablets and smart phones. These devices are used for both professional and private purposes. This means that the software is not only sold to professionals but also to consumers and business users who want to also use the application with mobile devices . This has caused changes to marketing and pricing but also imposes additional requirements for aspects like security and liquid multi-device experiences [Artto & Wikström 2005; Taivalsaari et al. 2014]. The appearance of mobile devices opened the market to low-cost mobile applications at a very low price – or the initial purchasing price may even be zero with the developer earning a living from alternative sources like embedded advertisement and in-application purchasing.

Internet as a delivery mechanism First, the Internet enabled the delivery of software without hardware media, but the applications were still installed on computers at customer premises. This made the delivery of software and updates faster and cheaper. When the speed, availability, and reliability of the software increased, many companies started to offer the applications primarily over the Internet. Thus, in addition to traditional customized or ”commercial off-the-shelf” software, the software business began to branch into ”Software-as-a-service (SaaS)”. This freed customer from hosting the application, but at the same time gave more control to the software vendor.

Virtualization and the cloud With these technologies, computing capacity can be sold as a utility and charged according to usage. This has improved the business of SaaS providers since the hardware capacity and costs became easier to manage. Virtualization is a key enabler of the cost-benefit of the SaaS model described above. It also amplified the challenges related to multi-tenancy, and solving these challenges has increased the engineering costs. The services offered to several customers are provided with the same computing resources and implemented with shared components. However, different customers need separate customizations. SaaS and multi-tenancy cause totally new, different requirements and affect every layer in the architecture. [Bezemer & Zaidman 2010]

The data of different customers need to be separated. Still, the use of SaaS with virtual clouds raises concerns and, among IT executives, security threats are the most dominating factor in risk perception.

Furthermore, new stakeholders – providers of computing capacity – have appeared. This is also called utility computing; such an approach means providing computational resources, with their provisioning based on actual use. This lowers the barriers to using a new system, as the initial costs are low or even non-existent if operational costs are ignored.

3. BUSINESS MODELS IN THE CONTEXT OF THE SOFTWARE BUSINESS

A look at previous literature shows that scholars overall do not have a common definition for business models. The challenge in finding a common definition is that business model related research is developed typically in silos, according to the researcher’s interest [Zott et al. 2011]. Although the concept of business

model has been described in many different words in the literature, all definitions usually cover the following two areas: ”what does the customer get?” (value creation process) and “how do we get the money?” (converting the market opportunities into revenue). Some researchers use the term ”business model” very simply:

- A business model describes how a company makes money and how it specifies its positioning in the value chain [Rappa 2001].
- Business model is an architecture for the product, service, and information flows, including a description of the various business actors and their roles, and a description of the potential benefits for the various business actors as well as a description of the sources of revenue. [Timmers 1998; Amit & Zott 2000]

Timmers [1998] considers a business model to be more of an industry level concept - or at least he does not limit it to the scope of one company. In the field of *software business research*, there is also a lack of rigorous previous definitions of a business model [Rajala et al. 2003]. They identified the need for a *software industry specific framework* for analyzing business models. The framework consists of four elements: product, revenue logic, distribution model, and services and their implementation. They consider that the business model refers to a single company, and that it describes only a single product at a time.

According to the literature review of Zott & Amitt [2011] the concept of business model has been employed mainly in trying to explain some of the following three phenomena: e-business, strategic issues, or technology and innovation management. The business model is applied and described in different ways in these three domains. Literature focusing on *e-business* has been interested only in cases where a company is engaged in Internet-based way of doing business; the concept of a business model consists of a value proposition, revenue model, and network and relationships together. In literature focusing on *strategies*, business models are seen as a strategy concept and the most interesting factor is the firm’s activities; these studies describe a business model as a notion of activities or activity systems. In *technology management* literature, business models are seen mainly as a mechanism to transfer technology and technological innovations into commercialized products; an important role of a business model is to release the potential embedded in a technology into market outcomes.

Technological development steps can trigger changes in a company’s business model, enable the new ones and force companies to seek them [Calia et al. 2007]. Existing revenue streams may decrease before the new ones begin to start bringing in cash, so a strong financial position is needed. A company needs new competences: new personnel have to come in and the existing personnel need to be re-trained/re-skilled. In software products, new architectures and modifications in existing software appear, which usually increases the complexity of the software. The whole field, the players, and their roles may change, which causes a need for a new kind of trust and rules. [Spinellis 2016; Cusumano 2010].

Figure 2 explains the elements of a business model, how they interrelate, and how the model is affected by the underlying business environment. Business models may not only emerge as a consequence of technological innovation, but can also be shaped by it. We apply the concept of business models in the way proposed by [Zotta et al. 2011] that is typical in the domain of innovation and technology management: *technology is seen as an enabler* of the business model, rather than as a part of the business model itself. We use the framework introduced by Rajala et al. [2003] as a guideline for our analysis.

The four elements of a business model – product, revenue logic, distribution, and services – are split into the components we consider to be in a close relationship with the industry evolution. Furthermore, in this study, the elements of the ”business model” represent the general way of doing business within the industry, and are not strictly limited to the business of one firm.

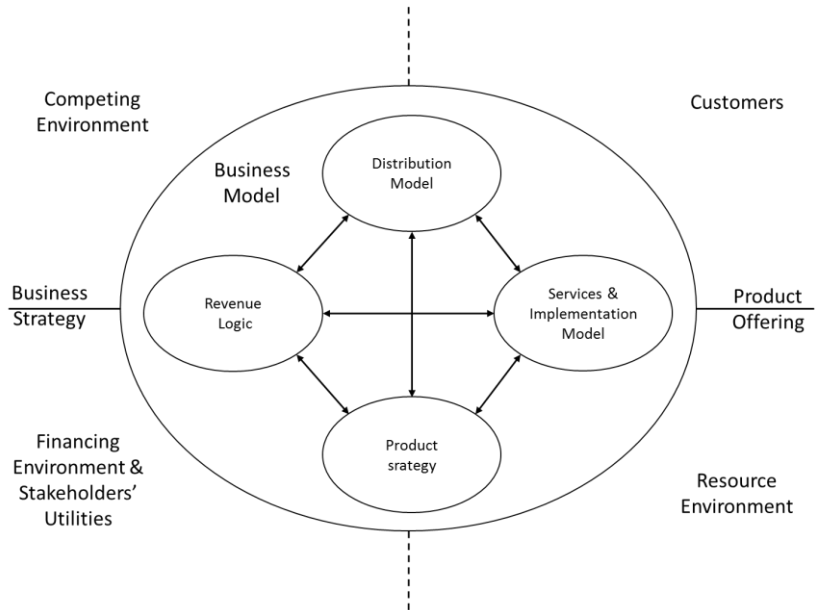

Fig. 2. Elements of a business model [Rajala et al. 2003].

4. VIEWPOINTS ON TRANSITION - EFFECTS OF THE BUSINESS MODEL ON THE SOFTWARE INDUSTRY

4.1 Conditions

The aspects of the software product are "what, for whom, and how" – what is the product, who are the end users, and what kind of technologies and tools are needed. In this chapter we take a look at the elements behind software products: technologies, tools, quality issues, the base characteristics of the commercial product, and the end users of the products.

4.1.1 *Innovations.* In Section 2 we listed a variety of enablers that indicated the beginning of a new era in the software business: hardware, open interfaces, the Internet, and virtualization. In innovation theory, changes are classified into four categories: incremental innovations, radical innovations, changes in technological systems, and changes in techno-economic paradigms [Dosi et al. 1988]. Incremental changes appear continuously in existing products and services (continuing the existing trend). Radical changes appear when new research findings are applied in products to transfer their properties or performance to a new step or cycle (movement to a new trend). Changes in technological systems are caused by combinations of several incremental and radical innovations in societal and organizational systems. Changes in paradigms are revolutionary and lead to pervasive effects throughout the target system under discussion. From the innovation theory point of view the enablers listed above have their primary source in technologies, but are also partially connected to societal changes, attitudes, and people's expectations. We can also find features of all four innovation categories in the background of changes in software business models. In the following we examine the innovations that have played an important role in the evolution of the software business over the decades.

Hardware related innovations The invention of microprocessors and VLSI in late 1960s started the rapid growth of computing power. In this context it is worth referring to Moore's law [Moore 1965], which forecast that the packing density of microprocessors and VLSI circuits would double at intervals of 18 months. This growth is still continuing and is accelerated by solutions based on parallel architectures and

multi-core processors. It was the beginning of the new era; it was cheaper to adapt the hardware for the demands of software needs than the opposite. In the 1980s the rapid spread of networking (technologies) accelerated this progress further and moved towards network-based (SOA-type) solutions.

Innovations in tools and paradigms for software development Until the 1960s, assembly (machine) languages were commonly used to reach effective use of processing capacity. The first commonly used "high level" programming languages were reasonably straight derivatives of computer machine language. FORTRAN - The IBM Mathematical Formula Translating System - was developed in the middle of 1950s for scientific and numeric computing [Wikipedia 2019]. In spite of having uniformity with the machine code of IBM 701 it started the transition towards high-level, machine-independent programming tools. Another language worth mentioning in this context is C-language. It was designed to support development of the Unix operating system for PDP-11 and indicates its architecture [Wikipedia 2019a].

The transfer of computers to the commercial and administrative sector caused pressure for new programming languages to take into account the needs of this new application area. Based on the sponsorship of the DoD (Department of Defense), an initiative to develop a COmmon Business-Oriented Language was originated [Wikipedia 2019b]. The initiative was based on a study according to which the costs of the software work exceeded the costs incurred by the equipment (late 1950s) and especially platform independence and portability of software were regarded as important properties. This can be considered as an observation of the first software crisis. The first specification of COBOL was published in 1960. The observations on the background of COBOL can also be considered a starting point to the quality driven approach of software work: portability, the dominance of software development costs instead of hardware, application oriented language structures, applicability in a variety of use contexts, and platform independence.

4.1.2 *Focus on Software Quality*. A transition started towards quality driven software development, first following the concept of structural programming and further towards an object oriented approach. The current approach of the main aspects related to software quality is documented by ISO/IEC in a series of standards [ISO/IEC 2011; 2001]. The first programming language supporting structural programming was Algol [Wikipedia 2019 c]. It was originally proposed in 1958 but it is best known as the Algol 60 specification from 1960. This language can be seen as an initiator of the "structural programming" paradigm. Algol never became a language used in the software industry, but its derivatives, C, C++, Pascal, Modula, Ada and many others, have been guides to the modern practice of software work. Algol can also be seen as a starting point of the object oriented programming paradigm. In the early 1960s, the Algol derivative SIMULA included the concept of classes and it has been the root of currently used languages such as Java and C++.

4.1.3 *The Evolution of Commercial software Products*. Software project business, which initially was the main model in developing software, appeared to be in many situations too slow and too expensive both to customer and to supplier. However, it is still valid in some cases, e.g., in highly specialized Business-to-Business systems. However, software prices have fallen so that you can buy "almost anything" for very little money or on a "pay as you go" basis. The model also has its challenges in the pricing of packaged SW: distributors and retailers get a big slice of the list price and eat away at the margins of the developing company. [Gewirtz 2015]

The main benefit of selling productized software is the effective reuse of code and other materials. This means generating new revenues without additional development cost. The software product business can be extended to a product line approach, which is a technology to support the derivation of a wide range of applications from a common core. Software product lines (SPL) are an effective approach for modular, large-scale reuse of software. In addition to software product lines, mass customization enables the building of products that are seemingly adapted to a particular use from the end-user perspective [Verdouw et al. 2014]. This enables customers to get features similar to customized software at an

affordable price. Similar results can be achieved by allowing the end user to fine-tune the product by enabling end-user programming in applications. Examples of such an approach include spreadsheets or accounting applications, shifting a part of the customization effort to the user.

While the two models, the project and product based business, are seemingly at the different ends of the spectrum of software business, they surprisingly share many similarities in the actual software development itself. Today, both are done in an agile, customer-driven development, where either a direct customer or an imaginary customer played by, e.g., a product manager, feeds user requirements into the development process. Thus, there is a clear feedback loop from end users to the development, no matter what the business model is – the difference is often only in the monetization model. Another common characteristic is that it is becoming next to impossible to draw a borderline between development and maintenance in terms of actual technical contributions [Mikkonen & Systa 2014]; again, the business needs to mark the difference in these approaches. Finally, as everything is becoming upgradable increasingly often online, the differences between the business models are becoming unclear since anything can be so easily modified that it does not really matter what the fundamental business rationale is. This new model is what is actually driving the next era of the software business we have entered, the time of diversification, enabled by the Internet (Internet as an enabler) and increasingly rapid development cycles.

4.2 Revenue Logic

Revenue logic can be split into two components: who receives the revenues and on what kind of basis. The transition from work amount based projects to a use-based cloud solution is described in this chapter. This transition has also opened the field up for new kinds of players.

4.2.1 *Revenue Models.* Revenue logic and pricing models have varied over time. New business models have given software companies new kinds of choices for pricing. However, the biggest winners have been the customers, who can buy software for almost any purpose at a very low price. As enterprise software has became available as SaaS versions, small companies in particular can now acquire software that they could not afford before.

Project pricing is based on the amount of work. It is typically expensive and contains a potential cost risk for the customer in the case of exceeding work estimates. These potential risks often lead to maximum cost and sanctions defined in the delivery agreement.

Due to the nature of information intensive products, producing software is expensive but reproducing it is very cheap [Sainio & Marjakoski 2009]. Standardizing and productizing software and pricing optimize the margins, which may be as high as 90 percent in the software product business. Also, the customer benefits of the product pricing, i.e., standard software and fixed price, are safe, because the cost is predictable.

New SaaS and cloud models are good from the customer's point of view: Customer expectations for services also pose a significant challenge for revenue logic. SaaS customers expect to pay according to time and usage and do not want to buy with a lump sum. At their best SaaS models enable the customer to "pay as you go" - pay according to usage. For the software vendor, this means differences in the revenue stream and needs to be considered in the pricing. The total revenue has to be gathered from small streams. It may also be a strength: if the customer commits to a certain cloud software and stays on the platform for a long time, he/she generates constant revenue every month. If the cloud strategy is based on mass customization, the pricing reflects it and collects certain features together in one pricing level. In any case, implementing and managing the desired pricing models is possible only if those needs are considered when designing the system architecture. Selling of licensing for a lump sum and typical service deals are priced differently. After developing a software product, the margins for selling it may be very high, even over 90 percent. As for services, they are typically very human-intensive, and may produce a

margin of only 30 percent or even less, which forces software companies to rethink their pricing and revenue models. [Cusumano 2008]

4.2.2 *Actors and Roles*. In the first phase of the software business, software development was done mainly inside the walls of hardware manufacturers. Companies themselves were both hardware and software suppliers. Hardware manufacturers could have the software done by subcontractors, but they were in any case tightly bound to a certain hardware ecosystem and certain technologies. When personal computers came on the market and private customers started to have software needs, open programming interfaces also started to appear. This enabled the rise of new business: software development could be done independently, without being restricted to a single hardware manufacturer. Later, this kind of outsourcing crossed borders between countries and was given new forms and names, for example offsourcing and nearsourcing. Global outsourcing was popular in countries of high labor costs (like Finland). Afterwards, callback decisions for global outsourcing were seen in those countries, due to raised prices and unpredictable add-on costs. In the early days of software engineering, software was a monolithic product, produced by an independent vendor.

When standardizing and productizing of software became common, component manufactures appeared on the market. Modern software strongly relies on infrastructure and components from third-party vendors: service operators, open source suppliers, etc. The relationships between software development companies and service firms turned software production into software ecosystems, where different companies collaborate to create value. Selling components is typically a mass business; they are cheap but easy to buy and their re-producing cost is almost zero. The increasing amount of different enterprise software caused a need for integrations between single systems. Depending on the field, integrations may be either very common and simple or very specialized and complex. Integration needs led some companies to focus on producing integrations. Overall, the opening of interfaces and ecosystems and the following division of business into different individual parts caused the birth of system integrators - companies who deliver solutions by selling a stack of hardware, software, and services as one product. Up until the 1980s, vertically integrated companies delivered complete system stacks: hardware, operating system, and applications. In the late 1980s and early 1990s, the horizontal layer structure of solution stacks changed into more modular clusters. In any case, software supply chains are transforming more and more into agile networks in response to the increasing volatility of business environments. [Cusumano 2008; Jensen & Cusumano 2013; Verdouw et al. 2014]

The opening and liberation of the software markets also caused the transfer of knowledge and competence from single, monolithic manufacturers to the network. The more complex the business becomes and the more the software also belongs to any other products, the more common it is for companies to subcontract the ICT competence from specialized companies. This transition has been an opportunity for ICT consultancy companies, whose business size, revenues, and competences have been growing continuously.

The invasion of SaaS and Cloud models has opened the field to a new kind of player. Customers expect cost-effective, efficient, and flexible delivery of IT services, with a maximum of monetary flexibility, which leads to the evolution from outsourcing to cloud. As a consequence, evolution from traditional IT outsourcing towards buying services from the Cloud is a certain significant trend that is changing the industry and actors within it. According to Leimester et al. [2010], the typical roles in the Cloud business are service providers, infrastructure providers, and service brokers [Riehle 2007]. *Service providers* - or content providers - develop applications that are offered on a cloud platform and use the hardware and infrastructure of an *infrastructure provider*. *Service brokers* - or aggregate service providers - offer new solutions by combining existing services into a new form of service. Also related to the Cloud environment, consultants serve as a support for the selection and implementation of relevant Cloud solutions. [Järvinen et al. 2014]

The term Open Source has almost as long a history as the history of computing. Since the majority of software was developed in the in 1950s and 1960s in academic and research organizations by developers working in collaboration, the results were commonly shared inside the collaboration network, and the IPRs (Intellectual Property Rights) were not tightly controlled by the developers. The software artifacts were distributed under the principles of openness and co-operation. Until the late 1990s, open software covered a wide variety of informal practices and artifacts, such as system software (operating systems, compilers, editors, development tools, etc.). In the early 1980s, Richard Stallman launched the GNU Project and some years later the Free Software Foundation was established to promote the concept of free software. Until the late 1990s, the free software concept mainly covered tools provided for common use for free – Linux, Netscape, Mozilla, Java, MySQL are examples of this era; a collaborative society of volunteer developers (crowdsourcing) is typical of (most of) these. Simultaneously with the spread of open software, licensing rules and practices were developed to guarantee the use of these products in a way that respects the original principles of openness (e.g. Creative Commons). Because of the long history of "openness" it is reasonably difficult to specify the moment when open source software had a real effect on the software business. It can be claimed that this began roughly in the early 2000s. Without going into details, discussed elsewhere in this paper, open source has accelerated the transition from license-based business towards business based on customization, services, and maintenance support of client products. In addition, the use of open data nowadays has a fast growing business value as well. Open innovation has also shown its power as a part of software engineering: fast transfer towards non-structured (non-SQL) data and the growth of applications based on "block chain" technology are examples of this. Openness has had important consequences in the software business field: old, large, traditional companies have lost their competitiveness and new companies based on lean and agile practices are the winners of the game.

4.3 Distribution

When the software business was liberated and open interfaces appeared, organized delivery chains also started to form. Thus the product itself was delivered as a concrete media; the distribution was also quite similar to distributing any other physical product: with retailers, sales partners, etc. The longer the delivery chain, the more challenges it caused to software pricing. Resellers and other intermediaries received a big share of the revenues, which decreased the supplier profits.

Affordable devices, like mobile devices, and the Internet as a delivery mechanism introduced new distribution channels. For example, "app stores," with related business consequences like different revenue-sharing models, are changing the software business. On the other hand, the browser has become the dominating platform for PC applications.

Another trend is extensive customerization due to affordable devices and the pervasiveness of the Internet. Access to applications is no longer controlled by a particular device, but by a user ID. Especially for consumer markets, the owning of a user ID means the collected user data has become a valuable asset since it can be used in targeted advertising. This has opened a totally new business line, in which the biggest business value lays on the data collected from users and their activities.

The app store concept introduced by the mobile industry and aimed at owning the user data together with the appearance of hosting services has added additional stakeholders to the business ecosystems. This is typically a third player that is not just a distributor in addition to the producer and consumer competing for the revenue.

4.4 Services

When interfaces became open and independent software development was formed into a separate business, the need for additional services also began to arise. On the one hand, the software suppliers added different installation, maintenance and implementation services to their offerings. Further, the offerings were extended to cover user training, consultancy (either system dependent or independent),

designing and delivering of extensive "whole solutions," and finally application provisioning, which relieved the customer from acquiring and maintaining the hardware environment.

The transition to SaaS especially affected the vendors of enterprise software, but has affected other software businesses, too. Game company sales are no longer based on products only; instead there are several online gaming services. In addition, platform companies, like Microsoft, had for a long time reported almost 100 percent revenue from products, but today their offering has also moved online. [Mikkonen & Systa 2014] Due to carry-on devices and SaaS-based offering, we are facing a fundamental systemic transformation towards a world where digital resources are constantly available online, and available for all to use [Jansen & Cusumano 2013]. SaaS started an era of licensing and delivering software on-demand based on a centralized hosting solution. Customers do not have to invest in their own hardware or pay extra for maintenance, which makes the cost very predictable. For instance, for SME (small and medium-sized enterprise) clients this gives access to software that many of them could not otherwise afford [Resceanu et al. 2014]. In fact, the potential cost advantages of SaaS are the strongest driver for SaaS adoption [Sultan 2014].

Before SaaS, software products were charged for either as a lump sum or "rented," i.e., charged per use or time. Customized projects were typically developed for a customer, and development was funded by charging a lump sum after delivery of the software. In both cases, a separate maintenance fee is often agreed. In the SaaS model, the maintenance is assumed to be included in the fee, and typically a separate maintenance business does not exist. On the other hand, the need for maintenance does not disappear and has to be included in the fees.

SaaS often reduces the possibilities for customization, unless the vendor can extend the versioning to hosting, too. However, versioning in hosting reduces the cost effectiveness of SaaS due to the multiple instances of HW and SW components. Thus, vendors typically try to satisfy the needs of multiple customers with a single product. The problem to be solved comprises multiple criteria and also the offering of SaaS products is wide, which is why different, systematic decision-making is needed to find the right product. [Godse & Mulik 2009]

The SaaS model also has organizational implications in companies. First, the role of IT departments is changing since development and IT need to collaborate more tightly. The term DevOps [Debois 2011] is often used to describe the required changes in the mindset. In addition, the SaaS mode outsources responsibility and work from customer companies. In SaaS, SW engineering and the customer interface also collaborate in new ways to constantly bring new value to customers. Instead of major releases managed by a separate business function, today's software development is about continuous maintenance" [Mikkonen & Systa 2014]. For example, software engineering at Facebook is about experimenting with what adds business value [Feitelson et al. 2013]. Also, feedback collection becomes systematic and automated. In an extreme form, this changes R&D to an experimentation system as in the Stairway to heaven model [Fayad et al. 2000; Olsson & Bosch 2012].

5. TRENDS FOR THE FUTURE

Despite today's divergence in the software business, we believe that we are just about to start seeing the wide opportunities of the future software business.

The future is a continuum of the past In this context we refer to the words of Larry Page (CEO of Google): "The main reason why companies fail is that they missed the future." This is true, but we have to remember that the future is built of the components of today, and today is the future of yesterday. As a result, we have to look backwards and recognize how we have come to this point, and evaluate the opportunities it gives for the future. The use culture of computers reflects strongly the opportunities enabled by the computer technology of each era. It has also had a significant effect on software work, which has always been done with the best tools available and following the practices that are best for their use. The changes in the tools and practices have been naturally carried out as a result of the needs of

different interest groups; progress in technology has been the enabler of the step-by-step changes that have taken place over the decades. These changes have been dealt with from different aspects of software and systems engineering in the book [Endres & Rombach 2003]. The phenomena reported in the book cover Moore's Law (discussed earlier), Hoagland's law (the capacity of magnetic devices increases by a factor of ten every decade) as a source of current masses of stored information, Cooper's law (the doubling of wireless bandwidth every 2.5 years) as an enabler of the fast growth of wireless solutions, and Fred's law (indicating the unknown source: the transmission capacity of wired networks for a fixed price doubles annually). These are examples of the technology-related trends we can expect to continue. The book is structured around the software development lifecycle, covering a variety of laws that it is relevant to understand in each life cycle. [Endres & Rombach 2003]

A comprehensive review of the general trends that can be discovered today is given in the paper [Jaakkola et al. 2014]. The analysis is based on a wide variety of sources and trends are classified into the innovation categories discussed earlier in this paper. The most important segment is naturally paradigms. This covers the transition towards openness (from several points of view) and the growing role of (big) data analytics as part of applications. Application intelligence, including machine learning based technologies, is an important aspect of modern information systems. Autonomous - independent, (more or less) intelligent devices play an important role as applications themselves but also as a source of data and as a collaborative partner of applications. The radical changes category covers for instance the growing importance of the context sensitivity and built-in intelligence of applications, new technical solutions like block-chain, and the changing characteristics of data (enriched data formats). The incremental change factors category covers aspects like transfer towards mobility, consumerization as a phenomenon, and gamification - creating pressure to improve user interfaces and the push towards gamified real life processes. Renewal of distribution channels also belongs to this category; combined with consumerization progress, the role of software product has changed radically, as discussed elsewhere in this paper.

Every business is a software business As indicated by the rapid rise of service providers such as Uber and AirBnB, software is enabling new ways to deliver services we so far have expected only an established service provider to provide. In addition to transportation and lodging, the same idea can be applied to virtually any line of business, covering also fields such as electricity networking and banking, where barriers to entry have been almost impenetrable and few attempts have been made. Partly this is due to legal regulation, but as such an approach can often provide a lower-cost alternative to fully supported infrastructure based services, to at least some extent it will only be a matter of time before the majority of them will be opened up to new competition.

Ultimately, almost all organizations need the ability to maintain their own IT operations, either directly or with a dedicated partner, which has been consciously selected. Managing such a software business is tough because software's ethereal nature offers infinite lucrative or catastrophic choices – the things to manage include the business model, market/customers, the execution strategy, the product or service, and the development process [Spinellis 2016].

Every product is a software product As an example, let us consider the challenges of the automobile market. Teslas are first and foremost computers that run on four wheels and are able to transport people from one point to another. Google and Apple are pushing the envelope even further with their plans to transport people automatically, without a driver other than a computer. In fact, even with a traditional car, millions of lines of code are needed to run it [Zax 2012]. The same trend is commonplace in numerous settings that have been considered unrelated to computing, such as ports or manufacturing plants. Much of this software is unlike the software that has been built before — it can never be shut down, as the systems that are controlled and, more generally, powered by software never sleep. The emerging Internet of Things (IoT) is paving the road to a society as well as humanity that is more dependent on software than any individual technology before — software is taking over the world [Andreessen 2011].

Clearly, systems of the scale mentioned above cannot be created just in time and for just one product or service. Instead, they are systems of systems created out of components that either happen to be readily

available [Hartman et al. 2018] or are engineered to the perfection that a group of developers deem critical enough. We expect that this will lead to the introduction of a collection of domain-specific "designs of dominance", where one vendor becomes irreplaceable in a certain field, much like Google is today in the field of maps and web searches. These islands will then be control points in the future software business, where smaller companies, with less important positions and control points, perform fractal-like development to create services by combining and reusing existing systems.

This development model will introduce the best and the worst parts of today's software business. On one hand, services are based on reusing existing systems that provide a robust starting point for development. On the other hand, services are constantly being refined towards a form where we are expected to consume them the most, disregarding at least some of the quality issues.

Variance in business models Even now, IT is fueling new business models, such as offering hardware and software for free, but making the customers pay as they go with zero initial cost. So far predominantly used only in the telecom industry, this business model requires the additional ability to fund underlying software development via IPO, partnering with funding organizations, or simply the company's own capital. Opportunities such as crowdfunding via services such as Kickstarter (http://www.kickstarter.com) are making this a viable option even for smaller companies, not only global giants.

6. DISCUSSION

The modern business expects ability to execute sophisticated routines and to perform analytics that require business and technology competences that traditional industries do not have (e.g. complex data analytics). So far, the typical solution for companies that lack ICT skills has been to subcontract ICT work from specialized companies, while keeping the leadership regarding the services within the companies themselves.

As part of this process, ICT consultancy companies are learning more and more. Moreover, the latest trends, like DevOps, require even more skills for successful ICT operations, and therefore give even more responsibility to ICT-related operations. In the long run, this will lead to slow but major change – size, revenues, and control of the ICT companies will increase, and the traditional companies have only the underlying hardware – if they still want to host it; in any case hardware can be replaced – and the brand – which can be diluted by the ICT companies who have an opportunity to introduce their own brand through continuous maintenance of the service [Mikkonen & Systa 2014].

Due to the possibilities provided by ICT, ways of doing business and related business needs are changing faster and faster. Approaches such as the Lean Startup provide an iterative, almost scientifically justified approach to creating new enterprises. In general, applying these approaches in the domain of software development, software and associated business – or business and associated software – must co-evolve. One of the caveats is an accidental, perpetual vendor lock-in, or a situation where there is no way to switch the vendor, as everything takes place in real time within business operations. In an extreme case, this can lead to hindering someone else's operations with hostile intent.

How the business related changes are seen in software quality, or actually, what has happened to the concept of software quality during the technological progress of decades. The paper [Jaakkola et al. 2017] handles reincarnation cycles in the certain areas of ICT. According to the paper, the main trigger of the changes is the progress in VLSI technology. It reflects directly to the processing capacity and memory size (both RAM and mass memory) of computers. Further, because data transmission is also based on complex calculations (in addition to the advanced transmission channels), even this part of ICT is tightly connected to the progress in VLSI technology. In all these fields changes have been exponential – already over decades. The progress creates platform for new applications, new kinds to use ICT infrastructure, also for new software development practices; this progress is handled in [Jaakkola et al. 2019].

Simultaneously to the technological progress the concept of "software quality" has lived its own life. The paper [Jaakkola et al. 2017] describes this progress in the following way: "In the 1950s, the period of a shortage of resources, quality software was based on minimal usage of the main memory (small size) and effective processing – *the first wave*. In the 1980s the focus was on the logical structure and maintainability of software – *the second wave*. The next *wave (third)* focused on quality (process) management. It was based on the idea that *quality software is a product that is produced by a high capability and maturity processes*." At this moment, in the *fourth wave*, we have new components in software quality: interoperability both in software and in development process level, openly available components and data, ability for collaborative development of software, ability to monitor the development process and further to use the monitoring data in improving the software process, etc. New aspect in software quality is also the growing role of data quality, which indicates that focus must also be placed on the quality of the data that is handled; this would be also an essential part of the *fourth wave*.

7. CONCLUSIONS

Our paper covers a variety of aspects relevant to changes in the software business. Figure 3 summarizes our findings and discussion in this paper.

We will not reiterate the detailed discussion related to the topics summarized in the figure. The line title indicates the factor under discussion and the columns in each line list the changes over time. The time scale between lines is not comparable: the listed items are in chronological order, but not comparable between lines. The changes indicated in the table are overlapping (no clear borders between the eras) and incremental (features of earlier eras remain and can also be seen in the latter periods).

The software business is currently in the final phases of a transition from shrink-wrapped software products (commercial, off-the-shelf software sold in retail) to cloud-based services, delivered either in the form of mobile apps - which often in essence are just an access method to a cloud service - or as full-blown web applications [Gewirtx 2015; Mikkonen & Taivalsaari 2013]. Indeed, software, together with associated data collection and data analysis, accounts for a greater and greater part of the value of products and services. In fact, companies such as Facebook have reported that it does not have a direction in its engineering - instead, it is up to the users to act so that a business-satisfying direction emerges based on usage data [Feitelson et al. 2013].

The development of the software industry has been guided by technological breakthroughs. This trend is still continuing and has also led to changes in paradigms. For example, accessibility and the low price of network capacity has had the effect on moving away from server solutions towards the cloud, or the heterogeneity of devices, and diversification of platforms has led to digital convergence.

Many of the strong trends of the industry have been simultaneously both opportunities and threats. Consumerization has extended the market and created opportunities for new businesses. In recent years, consumerization, including for example decreasing prices of mobile devices, has also led to a crash in software prices. It is possible to buy almost any software functionality for both private and enterprise purposes. Borders between these two segments have also almost disappeared; nowadays the biggest difference is in service capacity.

The Internet as a delivery channel has supported the general globalization trend. In contrast to many other industries, the Internet solves many logistics problems and eases entry into international markets. Naturally, some other new skills and competences are needed for successful internationalization. These skills cover, in addition to the widening variety of technical competencies, also a variety of "soft" skills, like understanding the differences between cultures in business, leadership and management, ability to benefit on the technical infrastructure in communication and interaction, etc.

A service culture and market have been established in the software industry. Customers want to have additional services and total solutions. However, this is both an enabler and a challenge; resourcing, work organizing, offering etc. have to be rethought and observed from new viewpoints. Margins in the service

business tend to be much lower than in the software product business, which requires new kinds of productizing and servicizing skills.

Factor	1945 --->	---> Time (Not comparable scale between rows) --->			---> 2020
Main era	Hardware driven Manufacturer owned ecosystem		Open ecosystem		Diversification
Enabler / Trigger	Hardware	Operating system	Interface	Internet	
Availability	Laboratory use	Common availability	Distributed	Everywhere	
The use mode	Closed shop	Open shop	PC Personal	Networked	
Computer unit	Mainframe	Minicomputer	PC /workstation	Affordable devices	Scalable Cloud
Computer resources management	Centralized	Centralized distributed use	Unmanaged distribution	Managed distribution	
SW development	Hardware drivern		Open Ecosystem		Continuous delivery Collaborative
Business focus	Programming	Project	Product	COTS Modifiability	Services / Affordable assets
Delivery	Manufacturer installation	On-site installation		Over-Internet installation	Customer upload / Virtualization Cloud
HW Innovations	Semiconductor		Microprocessor	Ethernet & TCP/IP	
Tool Innovations	Machine related languages / High level languages	Structured programming	Quality driven development	Object oriented approach	End-user programming
Revenue models	Development work		Product sales Licenses	Reproducing Use-based	SaaS In-application purchases
Actors, roles	In-house	Software companies		Outsourced Distributed	Collaborative

Figure 3. Summarizing the software industry evolution

As the industry becomes more mature and diversified, it also requires wider competences and successful utilization of networks to create a value creation that satisfies customers. Optimal competence profiles change fast, which causes challenges in the educational sector. How to educate an expert with good basic skills, the ability to refresh and learn fast, who masters both technical issues and architectural, integrational, marketing issues?

A big trend among customership is that the customer is becoming closer to products and product development. Customers are joining in the value creation at an earlier and earlier phase. Customer actions and feedback define and guide the development work in quite a short time span. The value of the business is being formed more and more by data gathered from customers. All this requires companies to be truly open to customer expectations and feedback. In addition, fast reactions from suppliers are needed.

REFERENCES

Raphael Amit, Christoph Zott . 2000. Value Drivers of E-Commerce Business Models. INSEAD working paper. 10-11. Retrieved May12th, 2019 from http://sites.insead.edu/facultyresearch/research/doc.cfm?did=2183.

Marc Andreessen. 2011. Why software is eating the world. Wall Street Journal August 20th, 2011. Retrieved May 12th, 2019 from https://uptakedigital.zendesk.com/hc/en-us/articles/115001167933-Why-Software-Is-Eating-the-World.

Karlos A. Artto, Kim Wikström. 2005. What is project business? International Journal of Project Management 23, 5 (2005). 343–353. DOI: 10.1016/j.ijproman.2005.03.005.

Cor-Paul Bezemer, Andy Zaidman. 2010. Multi-tenant SaaS applications: Maintenance dream or nightmare? Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE), ACM, 2010; 88–92. DOI: 10.1145/1862372.1862393.

Barry W. Boehm. 1976. Software Engineering. IEEE Transactions on Computers C-25, 12 (December 1976), 35-50.

Barry W. Boehm B. 2006. Some Future Trends and Implications for Systems and Software Engineering Processes. Systems Engineering 9, 1. 1-19. DOI: 10.1002/sys.20044.

Barry W. Boehm. 2006a. A view of 20th and 21st Century Software Engineering. Proceedings of the 28th International Conference on Software Engineering (ICSE), Shanghai, China. ACM. 12-29. DOI: 10.1145/1134285.1134288.

Jan Bosch. 2009. From software product lines to software ecosystems. Proceedings of the 13th International Software Product Line Conference 2009. 111–119. DOI: 10.1145/1753235.1753251.

Rogeri C. Calia, Fabio M. Guerrini, Gilnei L. Moura. 2007. Innovation networks: From technological development to business model reconfiguration. Technovation, 27. 426-432.

Michael A. Cusumano. 2008. The changing software business: From products to services and other new business models. Computer 41, 1 (January 2008). 20-27. DOI: 10.1109/MC.2008.29.

Michael A. Cusumano. 2010. Cloud computing and SAAS as new computing platforms. Communications of ACM 53,4 (April 2010). 27–29. DOI: 10.1145/1721654.1721667.

Patrick Debois. 2011. Devops: A software revolution in the making. Journal of Information Technology Management 24,8 (2011). 3–39.

Giovanni Dosi, Christopher Freeman, Richard Nelson, Gerald Silverberg, Luc Soete. 1988. Technical change and economic theory. Pinter London.

Albert Endres, Dieter A. Rombach. 2003. Handbook of Software and Systems Engineering - Empirical Obrervations, Laws and Theories, Pearson and Addison Wesley.

Dror G. Feitelson, Eitan Frachtenberg, Kent L. Beck. 2013. Development and deployment at facebook. IEEE Internet Computing17,4 (2013). 8-17. DOI: 10.1109/MIC.2013.25.

David Gewirtz. 2015. Selling software: Where have all the business models gone? ZDNet. Retrieved May 12th, 2019 from http://www.zdnet.com/article/selling-software-where-have-all-the-business-models-gone/.

Manish Godse, Shrikant Mulik 2009. An approach for selecting software-as-a-service (SaaS) product. IEEE International Conference on Cloud Computing, IEEE. 155-158. DOI: 10.1109/CLOUD.2009.74.

Björn Hartmann, Scott Doorley, Scott R. Klemmer. 2008. Hacking, mashing, gluing: Understanding opportunistic design. IEEE Pervasive Computing 7,3 (2008). 46-54. DOI: 10.1109/MPRV.2008.54.

ISO/IEC. 2008. Systems and software engineering – software life cycle processes. ISO/IEC 12207:2008.

ISO/IEC. 2011. Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models. ISO/IEC 25010:2011.

ISO/IEC. 2001. Software engineering — Product quality — Part 1: Quality model. ISO/IEC 9126-1:2001.

Hannu Jaakkola, Timo Mäkinen, Jaak Henno, Jukka Mäkelä. 2014. Openn. Proceedings of the 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO 2014). IEEE. 608–615.

Hannu Jaakkola, Jaak Henno, Jukka Mäkelä. 2017. Technology and the Reincarnation Cycles of Software. In Zoran Budimac (Ed.), SQAMIA 2017 - Proceedings of the Sixth Workshop on Software Quality Analysis, Monitoring, Improvement, and Applications. Belgrade, Serbia, September 11-13, 2017. (Vol. Vol-1938, pp. 5:1-10). Belgrade, Serbia: CEUR Workshop Proceedings.

Hannu Jaakkola, Jaak Henno, Jukka Mäkelä, Bernhard Thalheim. 2019. In Karolj Skala (Ed.), Proceedings of The 42nd International ICT Convention – MIPRO 2019 (pp. 985-992). Opatia: MIPRO and IEEE. Available in http://docs.mipro-proceedings.com/proceedings/mipro_2019_proceedings.pdf.

Slinger Jansen, Michael A. Cusuman. 2013. Defining software ecosystems: a survey of software platforms and business network governance. Software Ecosystems: Analyzing and Managing Business Networks in the Software Industry, Edward Elgar Publishing. 13-28.

Janne Järvinen, Tua Huomo, Tommi Mikkonen, Pasi Tyrväinen. 2014. From agile software development to mercury business. Casper, Lassenius and Kari Smolander (Eds.), Software Business. Towards Continuous Value Delivery: 5th International Conference ICSOB 2014. 58-71. DOI: 10.1007/978-3-319-08738-2_5.

Stefanie Leimester, Markus Böhm, Christoph Riedl, Helmut Krcmar. 2010. The business perspective of cloud computing: Actors, roles and value networks. ECIS 2010 Proceedings, Paper 56. Available in https://aisel.aisnet.org/ecis2010/56.

Tommi Mikkonen, Kari Systä. 2014. Maximizing product value: Continuous maintenance. Product-Focused Software Process Improvement. PROFES 2014. Lecture Notes in Computer Science, vol. 8892. Springer Cham. 298-301. DOI: 10.1007/978-3-319-13835-0_26.

Tommi Mikkonen, Antero Taivalsaari. 2013. Cloud computing and its impact on mobile software development: Two roads diverged. Journal of Systems and Software 86, 9 (2013). 2318–2320. DOI: 10.1016/j.jss.2013.01.063.

Gordon Moore. 1965. Cramming more components onto integrated circuits. Electronics38, 8 (1965). Proceedings of the IEEE 86,1 (1998). 82-85. Available in https://ieeexplore.ieee.org/xpl/tocresult.jsp?isnumber=14340 (Reprint).

Helena Holmström Olsson, Hiva Alahyari, Jan Bosch. 2012. Climbing the "stairway to heaven" – a multiple-case study exploring barriers in the transition from agile development towards continuous deployment of software. 38th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA 2012). 392– 399. DOI: 10.1109/SEAA.2012.54.

Helena Holmström Olsson, Jan Bosch. 2012. From opinions to data-driven software R&D: A multicase study on how to close the 'open loop' problem. 38th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA 2012). IEEE. 392–399. DOI 10.1109/SEAA.2014.75.

Risto Rajala, Matti Rossi, Virpi K. Tuunainen.2003. A framework for analyzing software business models. Proceedings of 11th European Conference on Information Systems (ECIS 2003). 1614–1627.

Michael Rappa. 2001. Business models on the web. Managing the digital enterprice. Retrieved May 12th, 2019 from http://home.ku.edu.tr/~daksen/mgis410/materials/Business_Models_on_the_Web.pdf.

Ionuţ C. Resceanu, Cristina F. Resceanu, Sabin M. Simionescu. 2014. Saas solutions for small and medium businesses. 18th International Conference on System Theory, Control and Computing (ICSTCC 2014). 140 - 144. DOI: 10.1109/ICSTCC.2014.6982405.

Dirk Riehle. 2007. The economic motivation of open source software: Stakeholder perspectives. Computer 40,4 (2007). 25–32. DOI:

10.1109/MC.2007.147.
Winston W. Royce. 1970. Managing the development of large software systems. Proceedings of IEEE WESCON, 26. 328–338.
Liisa-Maija Sainio, Emma Marjakoski. 2009. The logic of revenue logic? Strategic and operational levels of pricing in the context of software business. Technovation 29,5 (2009). 369–378. DOI: 10.1016/j.technovation.2008.10.009.
Diomidis Spinellis. 2016. Managing a software business. IEEE Software 33,5 (2016). 4-7. DOI: 10.1109/MS.2016.111.
Nabil Sultan. 2014. Making use of cloud computing for healthcare provision: Opportunities and challenges. International Journal of Information Management 34,2 (April 2014). 177– 184. DOI: 10.1016/j.ijinfomgt.2013. 12.011.
Antero Taivalsaari, Tommi Mikkonen, Kari Systä. 2014. Liquid software manifesto: The era of multiple device ownership and its implications for software architecture. IEEE 38th Annual International Computers, Software and Applications Conference 2014. 338-343. DOI 10.1109/COMPSAC.2014.56.
Paul Timmers.1998. Business models for electronic markets. Electronic Markets 8,2 (1998). 3-8.
Cor Verdouw, Ardie Beulens., Sjaak Wolfert. 2014. Towards software mass customization for business collaboration. 2014 Annual SRII Global Conference. 106-115. DOI: 10.1109/SRII.2014.24.
Wikipedia. Fortran.2019. Retrieved May 12th, 2019 from https://en.wikipedia.org/wiki/Fortran.
Wikipedia. 2019a. C (programming language). Retrieved May 12th, 2019 from https://en.wikipedia.org/wiki/C_%28 programming_language%29.
Wikipedia. 2019b. COBOL. Retrieved May 12th, 2019 from https://en.wikipedia.org/wiki/COBOL, 2019b.
Wikipedia. 2019c. Algol. Retrieved May 12th, 2019 from https://en.wikipedia.org/wiki/ALGOL, 2019c.
David Zax. 2012. Many cars have a hundred million lines of code. MIT Technology Review. Retreived May 14th, 2019 from . https://www.technologyreview.com/s/508231/many-cars-have-a-hundred-million-lines-of-code/.
Christoph Zott, Raphael Amitt, Lorenzo Massa. 2011. Business model: Recent developments and future research. Journal of Management 37,4 (2011). 1019–1042. DOI: 10.1177/0149206311406265.