SOFTWARE: FROM PRODUCT TO SERVICE THE EVOLUTION OF A MODEL

Dragoş Marian Mangiuc [1]

ABSTRACT: Among the consequences of the Enterprise 2.0 family technologies' growing popularity, we can observe the rise of a set of new business models for the use and employment of software applications, some of them also applicable for infrastructure components. The most popular of these models is by far "Software as a Service" (also called SaaS). SaaS is a software distribution model assuming the software applications are hosted and maintained by the vendor or the distributor, the user access being granted exclusively by means of the Internet. The paper at hand is a literature review and also an action research, meant to provide an objective and unbiased comparison of the two major software distribution models, and also to identify the strengths, the weaknesses and the favorite areas of applicability for each model. The paper is a part of a larger research performed by the author in the field of Enterprise 2.0 technologies.

Key words: Organizational knowledge, Enterprise 2.0, Software as a Service, Web 2.0, Semantic Web

JEL codes: M15

Introduction

Among the consequences of the ever-growing adoption of the Enterprise 2.0-related technologies, we can observe the advent of some new models for the purchase and use of business software applications and infrastructure components. The most popular of these new models is called "Software as a Service", usually abbreviated SaaS and nowadays represents a very common method of Enterprise 2.0-centered software distribution [Menken, 2008]. As the semantic technologies are adopted and integrated with the modern organizations' business processes, the standard "sale" of business software applications (usually called "Software as a Product") gives up in front of the new business model, mostly as the consumers realize the simplicity and efficiency of the new practice [Fan *et. al.*, 2009].

Research methodology

The paper at hand is a part of a larger research performed by the author in the field of organizational memory and Enterprise 2.0 technologies and also continues a previous doctoral research in the field of computer-assisted financial audit tools and techniques, whose final results were publicly defended in order to be validated by the scientific and academic community. The main goal of the aforementioned research was the identification of some new areas of applicability for the modern knowledge-based information technologies in the field of financial audit.

This paper is also a part of the IDEI 797/2007 research project, "*Development of Romanian Accounting Regulation between Heredity and Thanatogenesis*", funded on the basis of a national competition conducted by the National University Research Council (CNCSIS) within the Romanian Ministry of Education.

When possible, practitioners' expectations identification was attempted by means of direct interviews. In case some other author's opinion was enclosed, whether in exact quotation or synthetic form, a complete mention of the source identification information was made.

[1] Academy of Economic Studies in Bucharest, Str. Bozieni nr.8, Bl. 831, Ap.906, mangiuc@gmail.com

Validation of the research conclusions was performed by means of an informal discussion with some “real life practitioners”, members of some companies which performed or are in the process of performing the shift from using software as a product to using software as a service.

The author has over seven years of previous experience in the research area, and also a series of previous research results (published articles, conference attendances and doctoral research). By defending the research results at the proceedings of such a prominent scientific conference, attended by both scholars and practitioners bearing some interest in the research area, the author attempts to get further validation of his opinions, both confirmation and rejection of the aforementioned opinions’ scientific and practical importance being welcome.

The “Traditional” Model – Software as a Product

Traditionally, software applications are regarded as a product, or as an asset, both for the producer and the consumer. They are usually bought by the consumer, which may be considered the owner of a copy of the program [Cusumano, 2004]. The customer pays “on site” (when buying) a license fee which renders him the right to install and use the software application in a certain hardware configuration and for a certain number of users. In most of the cases, the software may be used for an unlimited time period, but on a single machine. The consumer might also pay a periodic fee, usually 5 to 25% of the initial price for update, maintenance and technical support services. From the accounting point of view, software applications are “capitalized”, which means they are to be presented as an asset in the buyer’s financial statements [Iod, 2002]. The buyer will also depreciate the initial cost for the whole estimated life span of the product (or the regulations-stated lifespan). In the author’s opinion, software is much less of an asset than hardware, but was always regarded and processed in the same way (at least by the accountants), both by the vendor and the consumer.

Building software as a product or as a commodity was a big and important “hit” for the companies offering revolutionary software products on the market. The fact is tightly related to the unique economic model of software development. The software production process is different from the usual production process as the marginal cost of each new unit produced is almost zero [Pohl *et. al.*, 2005]. Building a software application is a development project with a total cost ranging from a few thousands to a few millions dollars (or euros). Once the project is complete and the software application is finalized, the distribution costs are almost the same; no matter if a single copy or a million copies are sold. The first copy sold has a huge cost, but the marginal cost abruptly decreases as new copies are sold to the customers. As a consequence, the software companies whose products have “medium” success are able to cover production costs and also get some profit. The software producers whose products have tremendous success and usually sell millions of copies may report huge profits. The following figure (Fig. No. 1) represents the contrast between the “traditional” commodities profit margin and the software products profit margin. The area between revenues and expenses represents the profit margin. As is easily observed, for a certain growth of the sales value, in the software industry model the profit growth is much larger compared to the general model for commodities. According to the author this is mainly due to the almost inexistent variable costs.

The “standard” model of software as a product is mainly due to the tremendous success of some software producing companies like Microsoft, Oracle or SAP, which were proud to report the huge profits obtained. But aside from the “success stories”, the situation is very similar to the music industry, being almost exclusively based on “hits” or “breakthroughs”, which are extremely advertised software applications being of great interest for the large public [Haines, 2008].

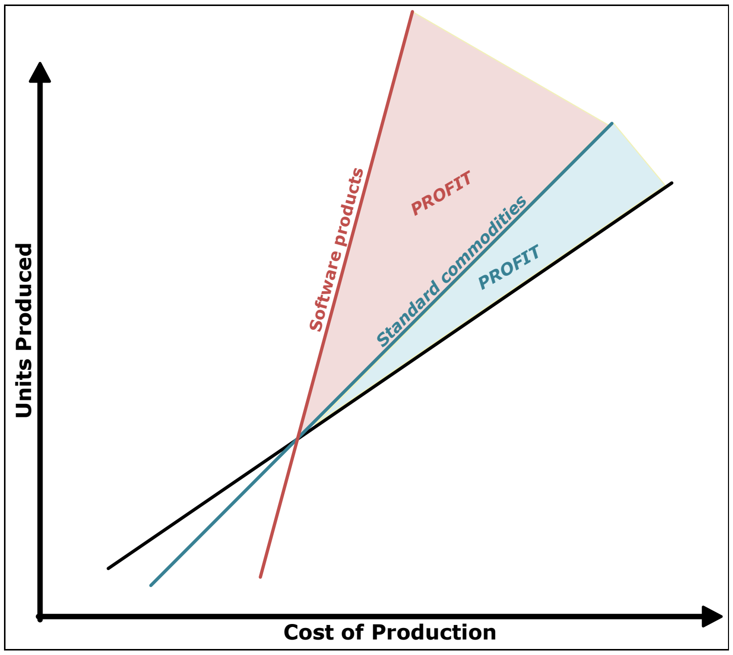

Fig. no. 1 - Standard commodities vs. software products profit model

However, the software products which are not regarded as "hits" by the market and the general public usually get much smaller profits, and their producers are almost always on a narrow line between profit and loss. Moreover, the top software producers almost never adopt the open standards which allow for free data transfer among applications. "Sealing" the applications, limiting the user's choice to a few proprietary formats and avoiding any possibility of converting documents to the formats of the direct competitors were always "features" of the top software producers, despite the major drawback they represent for the consumers and the final users. Once a company has become a customer, its possibilities to migrate to a cheaper or better product were drastically reduced [Gannod *et. al.*, 2005].

Even if the aforementioned analysis reveals a series of important benefits, using software as a product is also marked by a set of major issues. In most cases, the software application is downloaded from the vendor's website, and installation and setup are the exclusive task of the customer. As a consequence, the software application has to be prepared to run in heterogeneous, unstable or unforeseen environments [Pohl *et. al.*, 2005]. The software application is usually installed across the customer's network, on hardware configurations and operating systems installed and configured by the customer. At least in theory, the software application has to be able to face any challenge in terms of configuration and operate in any environment, with any set of parameters. According to the author's, reaching this goal is extremely expensive for the application's developer.

The second major drawback software developers have to face is the "cross-platform" support for their software, or the support for multiple operating systems. When a software developer intends to get a significant market share for its product, it has to develop a few separate versions of the software, one for each major operating system (Windows, Linux, MacOs, Unix). The more than significant differences among the aforementioned operating systems render just a small part of the application source code usable in all the versions, the development of four or five almost different applications (one for each operating system) being required in most of the cases. The negative impact on the software developer is obvious in this situation. A large quantity of time and human resources, which otherwise might be used for adding new features to the application, is used instead to test the software on different operating systems, on different operating systems' versions, or on different hardware configurations [Haines, 2008].

The drawbacks often affect the consumer, too. In most cases the cost of installation, setup and configuration for the purchased software applications are significantly larger than the purchase cost *per se*. Each organization has its own network, having many features and idiosyncrasies and, by consequence, aspects as the network topology or hardware incompatibilities have to be foreseen, taken into account and dealt with. Even for the most popular applications, which usually are thoroughly tested and adequately documented, the system or the network administrators take major risks for each setup and update of the software.

The New Model – Software as a Service

As a result of the aforementioned drawbacks, both software application developers and their customers are eager to adopt a new model for the development and the distribution of such applications, usually known as "Software as a Service" and abbreviated SaaS. Even if the model is around for a few years, being far from a total novelty, the difference resides in its recent success registered as a consequence of the high compatibility with the Enterprise 2.0 family of technologies [Blokdijk, 2008]. The SaaS success during last few years is tightly interconnected with the advent and the rise of the Web 2.0 technologies. As network connection and Internet access are ubiquitous, the business model behind the new approach may be accessible for the vast majority of software consumers. Web applications have reached a maturity level allowing on-line users to get the same experience and facilities as from traditional, off-line applications [Heydarnoori *et. al.*, 2006]. In the author's opinion, a comparison of the Web-based e-mail management suites with traditional e-mail client applications as Microsoft Outlook, or a comparison of the Microsoft Office suite with the Google Docs on-line suite may be enlightening.

According to the author, the general support or interest for SaaS, which is clearly observed for the majority of the corporate software consumers resides in the rapid adoption of the SaaS business model by the small companies in the fields or industries requiring many complex (and often overwhelmingly complex) software applications. Using software as a service was mostly attractive because it allowed to small companies having a minimal IT department (or having no IT department at all) to use software application otherwise out of reach due to installation, configuration and maintenance issues not manageable in the absence of a well-staffed IT department. A large multinational company almost always affords to assemble a team of professionals in order to properly install, configure and manage networks or large-scale software applications, but a small company almost never can afford such costs.

In our opinion, the second major benefit of software as a service is that the customer is allowed to pay only for what he really needs. The vast majority of corporate business software applications come with a fixed minimal cost of the hardware, installation and configuration efforts involved, usually computed for a large-scale department. Even if the department dimensions are significantly smaller, the cost is much less elastic and does not fall back accordingly. As a result, small companies are often forced to support cost levels similar to the ones of the large companies. Using SaaS allows the customers to significantly reduce the aforementioned costs, as they usually are charged based on the amount of time, storage space or application resources used. The main reasons for the success of SaaS are briefly described by means of the following figure (Fig. No. 2):

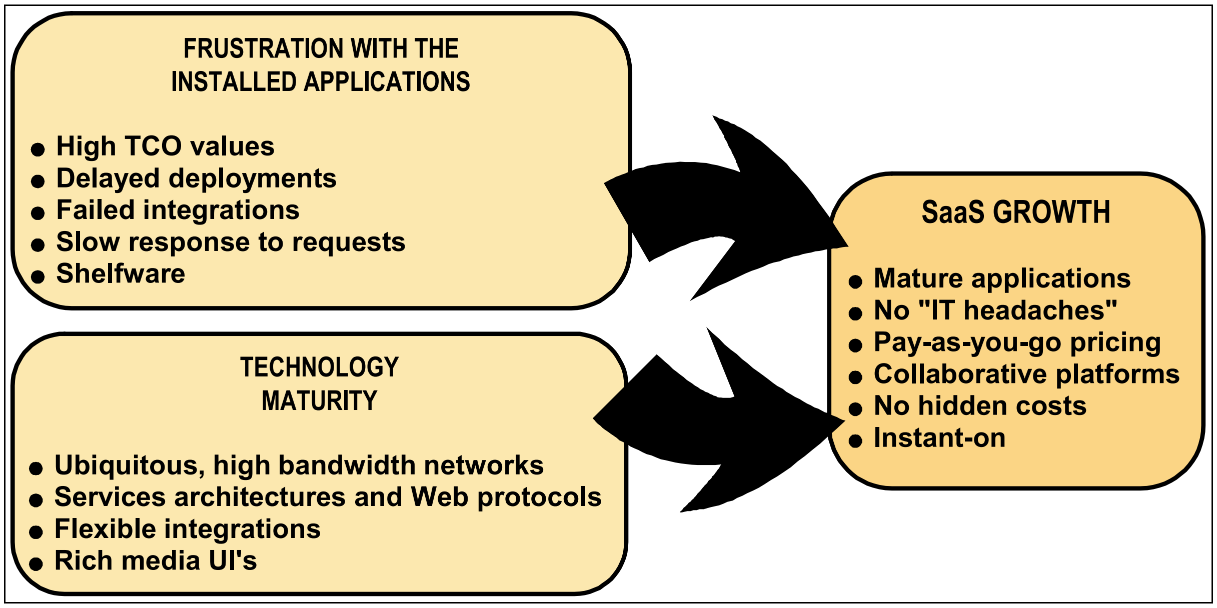

Fig. no. 2 - Reasons for the SaaS model success

Despite all the aforesaid advantages, we consider the SaaS model not to be a universal solution for all the corporate business software issues. According to some of the quoted authors [Heydarnoori et. al., 2006; Blokdijk, 2008], the new model will never replace the traditional model entirely, but will provide a increasingly better alternative to the software as a product. We also consider there are a set of major issues and a set of business areas where SaaS has less of a chance to succeed. According to the author, most of the issues originate from the fact that the customer is required to have extremely strong confidence in the software service provider. In some situations, granting such confidence to an outsider may be considered as a proof of irresponsibility and may even compromise business continuity. For example:

- A large multinational company producing candy and chocolate products should never let a software provider store and manage the secret manufacturing recipes for the products.
- A government agency processing secret or confidential information should never let an external service supplier manage its data sets.
- A healthcare institution taking the confidentiality of their patients' medical history very seriously, should not hand the management of the medical history data to an outside service provider.
- A bank should never let a service provider manage all the customers' financial information and even perform transactions on behalf of the customers.

Some other aspects here are open for interpretation. For example, the issue of the legal framework applicable in such contexts: to what extent has a Romanian company using applications hosted on Cayman Islands servers to comply with the local and the remote legal framework [Hall & Frey, 2007].

All the aforesaid issues diminish as the distributors of such applications provide solutions for the privacy, security and trust-related problems. Even though, the list of questionable practices remains open. For example, the SaaS model does not provide the means for the service consumer to locally store his own data. The customer's data are stored in the application provider's data center, placing the Internet between the customer and the provider. Consequently, any malfunction of the application provider's system or the customer's Internet Service Provider (ISP) renders the application unusable for the customer. Moreover, a malfunction of the application provider's system renders the application unusable for *all* its customers, which may be hundreds, thousands or millions of people or companies. The advantage of a centralized application management is the significant cost decrease, but the “dark side” of the matter is that any malfunction affects everybody. In the author’s opinion, a SaaS offer can easily become the victim of its own success if inadequately managed. A rapid increase in the number of customers not followed by the necessary increases in bandwidth, security systems, backup systems and staff may throw into chaos an otherwise successful project. Obviously, the “software as a product” model is also prone to disasters, but the malfunction only affects a customer or a small group of customers, not everybody in the same time. Even if the customer’s IT team is able to get involved swiftly in case of a malfunction, its expertise level in debugging the application is significantly lower than the application developer’s.

As a SaaS application provider, the merely existence of the company and business process depends on the provided application’s availability. The large providers of SaaS (like Google) are always bragging about the “24/7” availability of their applications and, by consequence, any malfunction, even a partial or limited one, is severely penalized by the media and also by the customers. The above is true even for the free applications, but in the case of commercial ones, a serious malfunction may irreversibly damage the image of the provider. On the other hand, investing in security and backup may be somehow appealing for a SaaS provider, as the benefits of the investment are simultaneously “delivered” to all its customers. The following figure (Fig. No. 3) performs a comparison of the “software as a product” and the “software as a service” models:

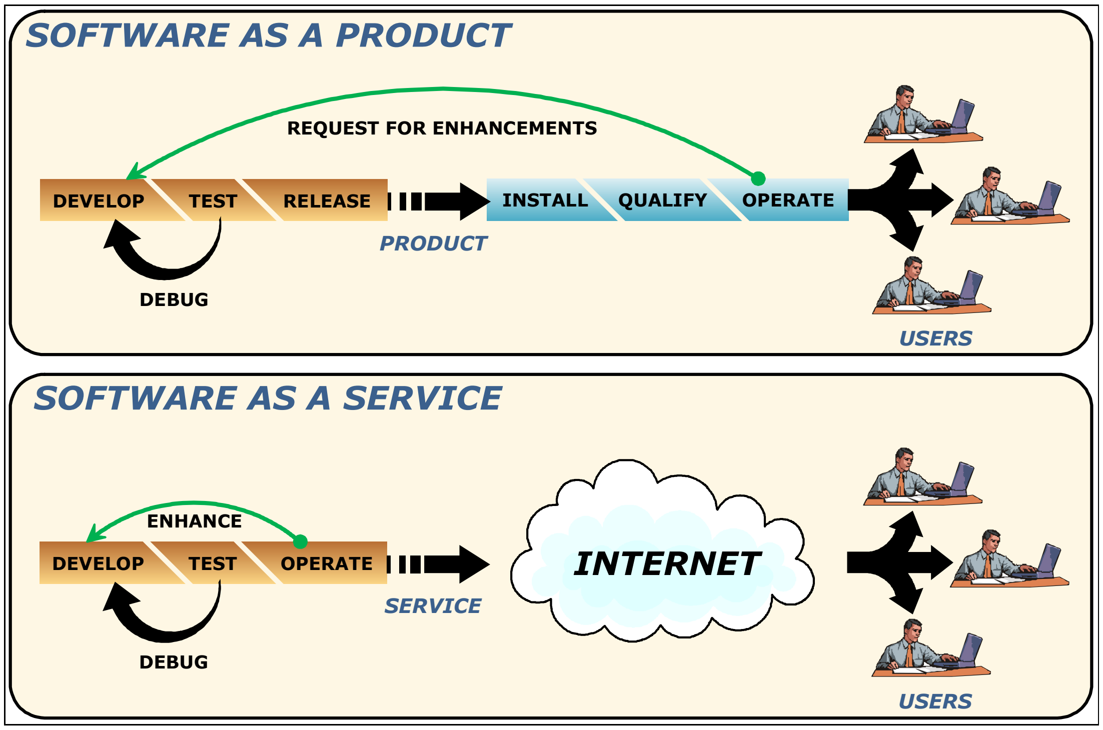

Fig. no. 3 - Software as a Product vs. Software as a Service

Further development of the SaaS Model – Infrastructure as a Service

The idea of providing technology “by request” is not limited to software applications, but is also extending to some other areas, as the infrastructure or the hardware system. The term “Infrastructure as a Service” (or IaaS) is used far less than SaaS, but is becoming a more and more important component of the Enterprise 2.0 family of technologies. For example, the Amazon company, worldwide renowned for its virtual store and generally considered the largest book seller in the world, started to provide IaaS services, the most successful one being S3 (short for Simple Storage Service). The service allows companies to rent data storage capacity, paying only for the occupied space. Even if the service, in its essence, is a very simple one, it allowed for the development of a whole suite of third-party applications aimed at access and data management, the very low prices asked forcing to a general decrease in the price of the Internet-based data storage services [AWS, 2009]. The S3 service may be used as a back-end for any software application (traditional or Web-based), its major strong points being scalability and extreme flexibility. A consumer of the S3 service in need of a significant and immediate increase of the S3 storage space does not have to do anything to get it, but use as many data storage space is needed, a subsequent payment being performed, depending on the subscription terms. There is no need a to add and configure new drives or storage units, and there is no need to contact Amazon in order to ask for a supplement, the whole process being implicit, when the new files of the customer are saved on the provider's storage servers. So, the customer gets instantaneous and smooth scalability for the infrastructure provided as a service.

A second member of the same family of services, is EC2 (short for Elastic Computing Cloud), consisting in a set of virtual machines which may be rented by the customers. The virtual machines are usually based on open-source software (Linux, Apache, MySQL, PHP) and are able to instantaneously scale up or down, depending on the customer's needs for the server. This service also had a price so low, that a general decrease in the price of the hosting services occurred. As the customers only pay for what they use, there is no minimum price [AWS, 2009].

A third technology having a solid contribution for the success of the SaaS model is *virtualization*, which is the abstraction and the re-partitioning of the existing hardware resources [Battle & Benson, 2008]. The procedure provides an increased application independence of the hardware configuration, allowing processes or operating systems to execute in total isolation. A virtual machine is a "guest" operating system which executes over a "host" operating system, which release the guest operating system from dealing directly with the hardware components. According to the author, the main advantages of virtualization are:

- ***Server consolidation*** – more physical servers are "concentrated" in a much more powerful virtual server, with a significant decrease in the cost of the processing unit.
- ***Server partitioning with resource limitation*** – allows for a physical server to be "broken" in a set of virtual servers, and also for a very detailed limitation of the resources each virtual server (or *partition*) is allowed to use.
- ***Application sandboxing*** – provides a security and isolation mechanism for an application or operating system, allowing it to execute completely separate from the other applications and operating systems sharing the same physical resources.
- ***Management of the development and testing platforms*** – allows for the easy simulation of the different execution environments, a useful tool for software applications development and testing.
- ***Rollout, rollback and patching*** – allows simplifying the application update process, by means of the update rollout and updating rollback, both at the application and the operating system level.

The use of the virtual machines significantly increased the efficiency of server resources management, allowing for a significant decrease in the total amount of hardware that needs to be deployed, installed, configured and maintained. Some Internet Service Providers (ISP's) employed virtualization in order to simultaneously execute different operating system instances on the same physical machine. The instances are then offered to the customers as *Virtual Private Servers* (or VPS). Five years ago, the ISP had to buy, install and configure a physical server for each customer in need of a server hosting, rendering the server hosting process very expensive, even prohibitive for the small companies which did not need the whole power of a physical server.

SaaS Security

From the security point of view, the SaaS model has both strong and weak points. The main strong point may be the fact that security concerns are offloaded, usually to entities or people with good competence and large expertise, able to build security for a software application.

The SaaS model involves centralized security, which may be a benefit if the securing process is performed as it should be. To the extent that the application provider is concerned about security issues, all the application's users will benefit from that. The bad side is that any omission or lack of inspiration from the application provider about security will invariably expose all its customers [Zhu & Zulkernine, 2009].

In the author's opinion, most of the SaaS model's security issues are tightly linked to the Internet-based data transport. As the vast majority of the SaaS offers are Web applications, the data is usually sent and received by means of the HTTP and HTTPS protocols, which may lead to serious suspicion among the potential customers. Subsequently, we think that any serious SaaS offer has to include and to stress the existence of some additional security features, the open standards generally being regarded as unreliable.

The concentration of all the customer data in a single location may also raise some questions. Because of this concentration, a breach in a SaaS application may expose huge quantities of confidential data. If a hacker attack succeeds in passing through the application provider's security system, the attacker will have access to data belonging to all the users of the application. Even if a series of risk management practices are usually employed (such as sandboxing or multiple firewall layers among the servers), the risk of an intrusion will never be completely eliminated.

The author's previous experience in the field of network security, even limited, leads to the conclusion that most of the networks are built with very strong perimeter (border) defenses, but are usually quite vulnerable from the inside. By consequence, once an attacker has passed through the firewall or the external protection system of the network, the attack from the interior is significantly easier. From this point of view, the concentration of the whole application and data content in a single network may seem a bad idea. A successful attack of the central location will usually have much greater unwanted consequences than an attack of a location from a distributed environment. The distributed security models are usually regarded as a decrease in the overall security, but, from this perspective, the damage of a single attack may be significantly diminished.

In a typical SaaS distribution model, any attack passing through the exterior perimeter of the network may have catastrophic consequences and, as a result, we think it is the application consumer's duty to get informed about the security features *inside* the provider's network, and not only about the external network defense system. The potential user should also ask about the security segmentation of the network, and also whether the installed security systems are able to prevent a user to access other user's data, as data leaks among customers may be as harmful as data leaks to the exterior [Greer, 2009].

According to the author, another very important reason for the ever-growing number of companies adopting the SaaS model for their software applications is the general maturity level of the security systems and technologies. Even if no software system is 100% sure, the aforementioned maturity level may allow for a reasonable security level without forcing to excessive costs. As the security systems do not intend to completely eliminate the risks, but to efficiently manage them, we think the SaaS model is able to bring major benefits if the risks are identified and managed as supposed to.

Conclusions

Employing SaaS implies a series of major changes in the way software applications are licensed and used. Many challenges arise, both for the software services providers and for the software consumers, but SaaS is able to provide both sides the benefits of a new and efficient software distribution model. The main benefits for the consumers usually reside in the decrease of the infrastructure expenses and immediate access to the latest version of the software applications they use. As for the software developers, they are able to get improved feed-back from the users of their applications, leading to a general decrease in the development costs and, as a result, an increase in the profit margin of their product. Moreover, the SaaS model is not the only successful initiative of this kind. Due to the almost unlimited possibilities offered by the virtualization process, infrastructure also becomes a service, significantly decreasing installation and maintenance costs for the hardware systems and the network (infrastructure management costs).

The adoption of the new software distribution model will not happen overnight, but will become a gradual process, having a variable growth rate, but, according to the author, the first companies to discover the benefits of the new model, and the companies willing to adapt in order to get the benefits, will gather significant competitive advantages from the adoption of the model.

References

1. AWS, 2009. *Amazon Web Services documentation*, available on-line at http://aws.amazon.com/documentation/
2. Battle, Robert, Benson, Edward, 2008. *Bridging the semantic Web and Web 2.0 with Representational State Transfer*, in: Web Semantics: Science, Services and Agents on the World Wide Web, Volume 6, Issue 1, February 2008, Pages 61-69.
3. Blokdijk, Gerard, 2008. *SaaS 100 Success Secrets - How companies successfully buy, manage, host and deliver software as a service (SaaS)*, Emereo Pty. Ltd. Publishing, ISBN 978-0980471649.
4. Cusumano, Michael A., 2004. *The Business of Software: What Every Manager, Programmer, and Entrepreneur Must Know to Thrive and Survive in Good Times and Bad*, Free Press Publishing, ISBN 978-0743215800.
5. Fan, Ming, Kumar, Subodha, Whinston, Andrew B., 2009. *Short-term and long-term competition between providers of shrink-wrap software and software as a service*, in: European Journal of Operational Research, Volume 196, Issue 2, 16 July 2009, Pages 661-671.
6. Gannod, Gerald C., Mudiam, Sudhakiran V., Lindquist Timothy E., 2005. *Automated support for service-based software development and integration*, in: Journal of Systems and Software, Volume 74, Issue 1, 1 January 2005, Pages 65-71.
7. Greer, Melvin B. Jr, 2009. *Software as a Service Inflection Point: Using Cloud Computing to Achieve Business Agility*, iUniverse Publishing, ISBN 978-1440141966.
8. Haines, Steven, 2008. *The Product Manager's Desk Reference*, McGraw-Hill Publishing 1st edition, ISBN 978-0071591348.
9. Hall, Thomas J., Frey, Kelly L. Sr., 2007. *Application Service Provider and Software as a Service Agreement Line by Line: A Detailed Look at ASP and Saas Agreements and How to Change Them to Meet Your Needs*, Aspatore Books, ISBN 978-1596228535.
10. Heydarnoori, Abbas, Mavaddat, Farhad, Arbab, Farhad, 2006. *Towards an Automated Deployment Planner for Composition of Web Services as Software Components*, in: Electronic Notes in Theoretical Computer Science, Volume 160, 8 August 2006, Pages 239-253.
11. Iod Group, 2002. *Software as a Service*, Director Publications Ltd., ISBN 978-1901580778.

12. Menken, Ivanka, 2008. *SaaS - The Complete Cornerstone Guide to Software as a Service Best Practices Concepts, Terms, and Techniques for Successfully Planning, Implementing and Managing SaaS Solutions*, Emereo Pty. Ltd. Publishing, ISBN 978-1921573132.
13. Pohl, Klaus, Böckle, Günter, Linden, Frank J. van der, 2005. *Software Product Line Engineering: Foundations, Principles and Techniques*, Springer Publishing 1st edition, ISBN-13: 978-3540243724.
14. Zhu, Zhi Jian, Zulkernine, Mohammad, 2009. *A model-based aspect-oriented framework for building intrusion-aware software systems*, in: Information and Software Technology, Volume 51, Issue 5, May 2009, Pages 865-875.