International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 04 Issue: 11 | Nov -2017 www.irjet.net p-ISSN: 2395-0072

CHANGES IN NECESSITIES TRADE AFTER MIGRATING TO THE SaaS MODEL

S. P. SANTHOSHKUMAR[1], G. RAMYA[2]

[1]Assistant Professor, Dept. of CSE, Rathinam Technical Campus, Coimbatore, India
[2]Assistant Professor, Dept. of IT, Rathinam Technical Campus, Coimbatore, India

---

***Abstract*:** *Service-oriented architectures are widely considered to be the determining trend in software engineering. Vendors of software products want to benefit by migrating to cloud environments. However, when transforming an existing software system from the Software as a Product model to the Software as a Service model the software engineering process changes. While the process in general has been researched sufficiently, very low effort has been put into understanding the impact on requirements elicitation. This paper investigates the necessary changes in the requirements engineering process and provides a systematic approach for a successful transformation. Furthermore, it discusses the new benefits in requirements elicitation that are inherent in a cloud environment. The paper then discusses the identified problems and developed solutions with regards to deduced guidelines and best practices. We conclude that the requirements engineering process profits from a systematic transformation when migrating a traditional software product to the Software as a Service model.*

Keywords: Software Engineering, Requirements Engineering, Software as a Service (SaaS), Cloud Environment, Reengineering.

1. INTRODUCTION

Studies show that 20% of the IT companies consider using Software as a Service (SaaS) as important or very important. For the majority of the IT specialists the topic is of average importance or lower. Nevertheless, this is due to reservations regarding security (76%), performance and availability (64%) and integration with existing systems (62%), as these companies describe. Another study points out that hiring a software instead of purchasing yields in a saving of 45% of the customer's expenses in a three year time span.

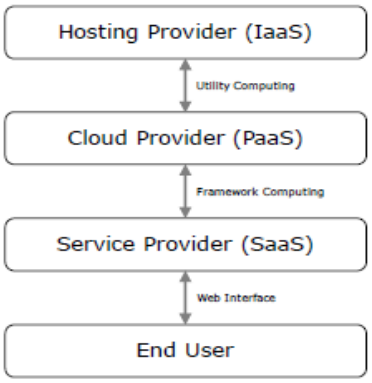

Figure 1: The cloud computing model

Figure 1: The cloud computing model [1] [10] SaaS is an element of the Internet-based computing model Cloud Computing. A cloud computing environment is essentially characterized by on-demand self-service, broad network access, resource pooling (using multi-tenancy), rapid elasticity and measured service according to the National Institute of Standards and Technology (NIST) [10]. To complete the cloud infrastructure two further elements – besides SaaS – have been identified by the NIST. The provision of runtime environments, libraries, other services and software tools by a certain provider is called Platform as a Service (PaaS). The customer of a PaaS has hardly any management control over the underlying platform components, but full control over the deployed applications [10]. Another step away from the end user is the Infrastructure as a Service (IaaS), which completes the cloud computing model. The IaaS provider is accountable for storage and network facilities and other hardware components, while the customer can install and run arbitrary software, including even operating systems [10]. Following the SaaS model, both the software system itself and the user data are hosted and stored centrally. Instead of purchasing a product, the user rents a software system, IT infrastructure and annexed services from the vendor and is typically charged on a pay-per-use principle [10]. A SaaS based software system, however, differs from one that is developed under the Software as a Product (SaaP) model in many ways. Its architecture is mainly database-oriented, middleware-oriented, PaaS-based and service-oriented

Figure 1. This result in differing non-functional requirements compared to classical software products. When transforming a software product into a software service, the vendor has to consider these changes in architecture and requirements [4] and in the whole software development process [7]. In this paper, we collect differences in software requirements between the two models. For this purpose, we have conducted a review of pertinent literature. We also studied research on existing software migration processes and developed conclusions on how to consider variations in requirements in such process changes. The aim of the present work is to provide a generic approach that helps those software developers who want to migrate their software product to the SaaS model. Section 2 covers the background information auxiliary for a comprehension of the SaaS model and the concomitant changes in software requirements. Section 3 outlines the work related to the requirements engineering process in a SaaS environment. In Section 4 the necessary changes in a requirements engineering process are presented and such a transformation is systemized. Section 5 discusses the developed process by means of deduced guidelines and best practices. In Section 6 limitations of the proceeding are discussed and Section 7 draws conclusions and provides future work.

2. BACKGROUND

2.1. Software as a Service

For many years, software has been produced in a supply-side oriented manner. A software vendor puts effort into the requirements elicitation for a certain problem, develops and tests the software and releases the final product to the market. The customer or the software vendor's support team installs a copy of the software product at the customer's infrastructure after purchasing a licence. While minor software updates are usually conducted via an Internet interface and included in the one-time price, major upgrades often require buying a new software product [3]. The SaaS model, in comparison, is the trend in software engineering of the 21st century that challenges this traditional model [1]. The customer of a SaaS-based software purchases a usage right for a certain time span. In return, the vendor grants access to the online service, often combined with an individual number of accesses depending on the customer's price plan. Since its first mentions in research in the 2000s, SaaS has gained more and more attention both from scientific and production points of view. While different approaches – such as iterative and incremental development processes and modular software products – have been established to address the issues of developing and deploying more complex software products, the SaaS model is a radical shift of the means by which software is engineered. Providing Software as a Service in contrast to a product, at a first glance, is a manner of distribution policy business issues like time to market, customer involvement and release cycles. The service-orientation of software, however, also comes with major paradigm changes regarding the software development. The SaaS model utilizes services as the rudimentary factor for organizing the complexity of software. The underlying principle of software design is Service-oriented architecture (SOA), an architecture in which loosely coupled but strictly separated software components (usually single business functions) interact via public interfaces as composite services. This allows for binding components only when they are needed and in a scalable way. SOA itself is platform-agnostic and does not define the manner of service orchestration, security etc. These services are made available by service providers that come up with the service infrastructure and the implementation and provide the interface description for access over the Internet (web-based). In order to publish and find integration-ready services, a common service directory is needed (see Figure 2). Services itself are composed of other services recursively [3].

2.2. Changed Requirements

Compared to the traditional SaaP model, SaaS relies on a different infrastructure and varies in distribution and access (see Section 2.1). When migrating a software product to the SaaS model, one usually intends to maintain most of the software's functionality [9]. As a result, the differences in the software engineering process narrow down to nonfunctional requirements [2] and other aspects affecting the software development process like operation, management and architecture, albeit not functional requirements. Those aforementioned differences in non-functional requirements are basically due to three factors:

1. SaaS-based software is necessarily hosted in cloud environments either operated by the software vendor itself or by a third party offering PaaS solutions (see Section 1). A few very large companies offering software services unify the PaaS part and the SaaS part under a single roof, such as the on-demand video streaming platform Netflix. These companies act as platform providers for themselves.

2. Such software is primarily distributed as a web-based application using the Internet and associated protocols for data transmission.

3. A high proportion of software offered as a service is realized as browser-supported applications, meaning that no dedicated software is necessary on the client's device except the already existing web browser.

Factor 1 results in a focus of the non-functional requirements on security, data confidentiality, privacy and compliance, since the server location determines legal aspects such as data protection laws and a company's compliance regulations [7]. In addition, a cloud service provider is a more probable victim of security attacks than a

decentralized structure or a company's private network. Albeit it is harder to conduct successful attacks on professional cloud service providers, special precautions have to be considered. Most differences in non-functional requirements are a consequence of Factor 2: Multi-tenancy, user concurrency, configurability, scalability, reliability, performance, availability, compatibility, interoperability, portability, effciency and immediacy [5] [7] [8]. Other aspects include continuous evolution, the involvement of a higher number of stakeholders and increased usage monitoring [7]. Special demands on aesthetics and user interface design and the limitations of browser-supported applications are influenced by Factor 3.

3. RELATEDWORK

Back in 2000, Bennett et. al [3] have recognized trends in software development that are influenced by the emerging Internet. They develop a future vision in which software is flexible, interactive, personalized and self-adapting and the software engineering is demand-led, service-oriented and focusses on the requirements elicitation. In their conclusion the authors point out that future work should focus on the necessary changes in the software engineering processes. Seminal work on the basic concepts behind SOA. As one of the first authors he described the effects of SaaS on business processes and on software engineering. This conclusion that the SOA requires strong alterations in software design. Olsen, the author of has investigated necessary paradigm changes from a business point of view. He outlines that a SaaS-based software system creates a very different customer relationship than a SaaP-based. Olsen makes the update mechanisms responsible as they require long-term commitment of the vendor and facilitate non-disruptive upgrades. The author also points out the advantages of modularity and rapid releases for the customers. In their study [1] Armbrust et al. present dentitions for the different aspects of the topic cloud computing. They locate the role of SaaS and list benefits as well as obstacles and demonstrate means of how to avoid them. Since these seminal works, research has made a lot of progress. In their study [7] Kumar and Sangwan present traditional software engineering process models and main concepts (e.g. iterative development). They continue collecting aspects which make the development of web-based applications different from traditional software. According to the authors the main aspect is the continuity of the process that also requires a systematic, repeatable and iterative process. Together with lists of attributes and characteristics of web-based applications they provide a very general adaption of a traditional software engineering process model towards a model which is suitable for web-based applications. However, the authors fail to present a detailed process as a result that can be used for developing such applications. They found out that the research interest has increased over the last years. They identify the main challenge for cloud-based software engineering to be the lack of standardization. E.g. choosing a PaaS provider may result in platform lock-ins where customers cannot easily switch to another service provider. Besides a grouping and the presentation of challenges for SaaS de

Velopers, the authors provide definitions of the terms SaaS and SOA. They conclude that a research gap exists regarding the formalization of a complete reengineering process in terms of reconstructing the software for a new platform. Balian and Kumar [2] group and review studies in the field of SaaS development. They introduce literature which focusses on development from scratch as well as studies for migration and reengineering. Furthermore, the authors discuss research on quality models for SaaS and draw the conclusion that the adaption of software engineering process models, quality models and metrics for SaaS is not sufficient. The most relevant recent works have been conducted in the field of comparing the software engineering processes of the SaaP model and the SaaS model. Tariq et al. address the impact a cloud environment has on the requirements of an application. They list technical non-functional requirements, legal concerns and other issues from the data management. The authors then categorize these topics and identify the new stakeholder cloud service provider. As a result, they propose an addition to the Capability Maturity Model Integration (CMMI) reference model that includes a checklist for the new stakeholders. Research has provided detailed descriptions of the SaaP model and the fundamentals behind the SaaS model. Recent work also exists which covers the transformation of a service oriented system into cloud-based software that follows the SaaS model [4] [5] [9]. However, there is no process support for migrating an existing software product into such service-based software. Furthermore, we could not find any migration strategies that cover the differences in the requirements elicitation process. This paper intends to fill this gap by providing a systematic and generic approach for sustainably migrating a traditional software product to the SaaS model. This approach covers the software adaptions as well as the necessary changes in the existing software engineering process in a clear and repeatable way with focus on the changed requirements elicitation.

4. REQUIREMENTS ENGINEERING PROCESS FOR SaaS

4.1. Differences Between Processes

This section strictly focusses on the requirements engineering process. However, some aspects affect different phases of the software engineering process as well and others are as a matter of fact just side issues from requirements' point of view. Nevertheless, all aspects are included in the enumeration in order to provide a holistic view of the differences between the traditional and the SaaS-based requirements engineering process. This understanding of the requirements and their importance is

crucial for the general software development process. First of all, in comparison with the SaaP model, SaaS involves more kinds of stakeholders. Kumar and Sangwan [7] identify those as analysts, graphic designers, customers, marketing, security experts etc. But the requirements engineering process not only has an expanded stakeholder basis. As mentioned in Section 2.2, SaaS comes with a stronger customer involvement and longterm relationships between the SaaS provider and the end user. The user is motivated to provide feedback – directly or indirectly via usage monitoring –, since a feature enhancement can be expected and he/she will profit from it without extra cost and in foreseeable time. The integration of bug fixes and new features is seamless and without interruptions because the software is centrally hosted on the company's servers instead of on the customer's infrastructure. They are furthermore integrated without time delay since time to market is reduced significantly due to the fact that new versions are released early and often and are not considered to be a distinct software product. The difference between enhancements and bug fixes becomes indistinguishable to the end user. These less disruptive updates, which respectively approach just a few problems but in return happen more frequently, require fewer amount of retraining on the end user's side. Moreover, the centrally hosted, multi-tenant software as a service offers additional opportunities in testing new features. The acceptance can be evaluated by rolling out the feature to just a selected proportion of users and awaiting their feedback. Even providing two or three variations of a feature to several user groups is possible and allows for comparing differences and selecting the implementation with the highest user approval.

4.2 A systematic transformation of the requirements engineering process follows these steps:

Step 1: Establish a paradigm change with respect to highly fluctuating requirements. Developers who are used to traditional software products need to adapt to the non persistence of requirements in the SaaS context. The vicinity to agile development and the new methods of elicitation make the requirements volatile.

Step 2: Integrate requirements engineering into an iterative and incremental software engineering process. Such a software engineering process is not a unique characteristic of service-based software and can be found in traditional software development as well. However, the volatile nature of the requirements and the frequent release cycles demand such iterations and regular software increments.

Step 3: Identify and prioritize stakeholders using systematic methods.

Step 4: Involve customers through integration into the requirements engineering process. Invitations for featureand bug reports are crucial for taking advantage of the migration to SaaS. The users need to get the feeling that their involvement can have an impact on future feature enhancements and short-term bug fixes.

Step 5: Implement instruments for user feedback (e.g. usage monitoring, feedback forms). In order to encourage customers to provide feedback (see Step 4), such a culture needs to be established. This can be achieved by e.g. providing feedback buttons on single features, offering side-wide available feedback forms and by using the many ways of usage monitoring offered by cloud software.

Step 6: Develop mechanisms for seamless update integrations. As stated before, a major benefit of cloud-hosted software is the deployment in the hand of the software developers. Thus, the integration of updates comes handy: The new software pieces only need to be installed once and on a predictable server environment – the companies cloud server – and not the client's infrastructure. In addition, the high frequency of small updates makes it easy to integrate without shutdown times, since the number of lines of code or the changes in the database design are proportionally smaller. The short downtime of parts of the system is less noticeable than a traditional maintenance downtime of the whole system.

Step 7: Develop support for software variations per user group for acceptance testing reasons. The new requirements engineering process makes it possible to develop multiple versions of unknown acceptance and roll out the variations to different user groups. Acceptance can then be tested using the methods of Step 5.

5. DISCUSSIONANDBESTPRACTICES

The main goal of this paper was to outline the differences between the requirements engineering process of a traditional software product and the process of a software service and to provide a systematic approach for migrating from one to the other. Following the presented approach reduces the risk for leaving out necessary changes in the requirements engineering process. For those who consider migrating a software product but have not decided yet, the approach defines the scope of changes which would be intrinsic to a planned migration. As such this paper's approach others benefits that cannot be found in literature as of today. In order to accommodate the transformation approach we provide a collection of best practices, which came across during literature review, for some of the steps: Step 2 is suited best by applying agile software development methods, such as Scrum. The stakeholder analysis of Step 3 is well conducted when using socio-diagrams or power matrices. The authors of [6] provide an extensive description of their process of stakeholder identification and impact analysis. The measurements already mentioned in Step 5 for motivating users to provide feedback have been successfully

conducted in practice and can be recommended. That is implementing application-wide feedback forms and applying usage monitoring.

6. LIMITATIONS

This approach covers the requirements engineering process, which is only one part of others in the whole software development process. The migration of software products to the cloud can still fail due to other implications of such a process migration. Another limitation of this approach is the focus on webbased service-oriented architectures. This circumstance is owed to the experiences from the literature review. Most research does not differentiate between SaaS and web-based systems, which makes the development of a transformation approach generalized for other kinds of customer interface nearly impossible. 7.

7. CONCLUSION AND FUTURE WORK

The way we practice software engineering has changed dramatically. Developing SaaS is one of the reasons why changes in software engineering processes are indispensable. The requirements elicitation of a software realized as a service differs to the traditional product in many ways, some of them are fundamental (see Section 2.2). However, the differences come with numerous advantages, such as longterm customer relationships, focus of resources and more frequent feature enhancements. The requirements engineering process requires transformation when migrating from an existing software product to the SaaS model. This paper has offered a systematic approach for this transformation that can be used by software developers who want to adapt the way they determine and meet the requirements of their software system. Future work is to research on how to combine the benefits of these new requirements elicitation methods with agile software engineering processes that already focus on iterative and incremental development.

REFERENCES

[1] M. Armbrust, A. Fox, R. Griffth, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al. A view of cloud computing. Communications of the ACM, 53(4):50–58, 2010.

[2] N. Baliyan and S. Kumar. Towards software engineering paradigm for software as a service. In Contemporary Computing (IC3), 2014 Seventh International Conference on, pages 329–333. IEEE, 2014.

[3] K. Bennett, P. Layzell, D. Budgen, P. Brereton, L. Macaulay, and M. Munro. Service-based software: The future for flexible software. In Software Engineering Conference, 2000. APSEC 2000. Proceedings. Seventh Asia-Pacific, pages 214–221. IEEE, 2000.

[4] M. A. Chauhan and M. A. Babar. Migrating service-oriented system to cloud computing: An experience report. In Cloud Computing (CLOUD), 2011 IEEE International Conference on, pages 404–411. IEEE, 2011.

[5] M. A. Chauhan and M. A. Babar. Towards process support for migrating applications to cloud computing. In Cloud and Service Computing (CSC), 2012 International Conference on, pages 80–87. IEEE, 2012.

[6] Khajeh-Hosseini, D. Greenwood, and I. Sommerville. Cloud migration: A case study of migrating an enterprise it system to iaas. In Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on, pages 450–457. IEEE, 2010.

[7] S. Kumar and S. Sangwan. Adapting the software engineering process to web engineering process. International Journal of Computing and Business Research, 2(1), 2011.

[8] J. Y. Lee, J. W. Lee, D. W. Cheun, and S. D. Kim. A quality model for evaluating software-as-a-service in cloud computing. In Software Engineering Research, Management and Applications, 2009. SERA'09. 7th ACIS International Conference on, pages 261–266. IEEE, 2009.

[9] G. Lewis, E. Morris, and D. Smith. Service-oriented migration and reuse technique (smart). In Software Technology and Engineering Practice, 2005. 13th IEEE International Workshop on, pages 222–229. IEEE, 2005.

[10] D. Ma. The business model of software-as-a-service. In Services Computing, 2007. SCC 2007. IEEE International Conference on, pages 701–702. IEEE, 2007. [11] P. Mell and T. Grance. The nist definition of cloud computing. Special Publication 800-145, 2011.