ENGINEERING AND TECHNOLOGY

Changes in Requirements Engineering After Migrating to the Software as a Service Model

Dr. Madan A Sendhil, Dr. Ganeshkumar G

Department of Information Technology

Rathinam Technical Campus, Coimbatore, Tamilnadu, India

Abstract

In the near future, service-oriented architectures will predominate in the field of software engineering, according to the consensus. Because of the possible benefits, software product makers are interested in migrating to cloud environments. When an existing software system is transformed from the Software as a Product model to the Software as a Service model, the software engineering process is modified. Although sufficient research has been conducted on the process in general, very little effort has been devoted to comprehending how the influence would be felt throughout the requirements elicitation phase. In this paper, we analyse the required changes that need to be made to the requirements engineering process and provide a methodical approach for successfully implementing such changes. In addition, it discusses the new benefits associated with the cloud-native elicitation of requirements. The paper then examines the problems that were found and the solutions that were developed in connection to the derived guidelines and best practises. When transitioning from a typical software product to a model based on software as a service, we have concluded that the requirements engineering process can benefit from a methodical change.

This topic is connected with the terms Software Engineering, Requirements Engineering, Software as a Service (SaaS), Cloud Environment, and Reengineering

Introduction

Recent data indicates that just 20% of IT companies see the use of software as a service (SaaS) as crucial or extremely important. The vast majority of people who operate in the field of information technology deem the issue to be of moderate or lesser significance [21]. Nonetheless, this is the situation due to concerns surrounding security (75 percent), performance and availability (63 percent), and interface with existing systems (61 percent), as described by these organisations [21]. According to the conclusions of yet another study, a client's decision to lease software rather than purchase it can result in a 45 percent savings in costs over a three-year period .

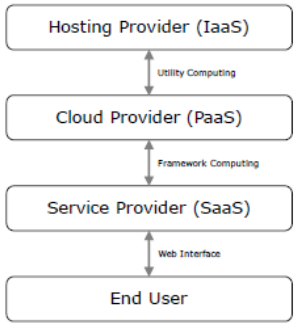

Figure 1: The cloud computing model

978-81-933187-0-6 © 2023 SEEEPEDIA.ORG
Society for Engineering Education Enrichment

SaaS is one of the components of internet-based cloud computing, which contains SaaS as one of its components. According to the National Institute of Standards and Technology (NIST), a cloud computing system is characterised by on-demand self-service, broad network access, resource pooling (using multi-tenancy), rapid adaptability, and measurable service. In addition to SaaS, the NIST has identified two other components necessary to complete the cloud architecture. Platform as a Service is a marketing term referring to the provision of runtime environments, libraries, and several other services and software tools by a particular provider (PaaS). The customer of a PaaS has very no management control over the underlying platform's components, but complete control over the deployed apps . Infrastructure as a Service (IaaS) is the final component of the cloud computing idea and is positioned one step further away from the end user. The customer is responsible for the installation and running of all software, including operating systems, whereas the IaaS provider is responsible for the storage, network, and hardware components. In the SaaS model, both the actual software system and user data are hosted and stored in a central location. The customer rents a software system, IT infrastructure, and auxiliary services from the vendor rather than purchasing the product, and the user is frequently charged on a pay-per-use basis [10]. In a number of ways, however, a software system designed using the Software as a Service (SaaS) model is distinct from one developed using the Software as a Product (SaaP) model. Its architecture focuses mostly on databases, middleware, PaaS, and service orientation . Therefore, the needs for non-functional parts of typical software solutions will vary . When converting a software product into a software service, the software provider must account for changes in the software's design and requirements [4, 7], as well as throughout the software development process . In this study, we collect the numerous software requirements differences between the two models. In order to attain this objective, we have reviewed the relevant published research. In addition, we examined the research that had been conducted on the existing software migration procedures and drew recommendations on how to account for variances in requirements while updating such processes. This endeavour aims to give a broad method for software developers who are interested in moving their software product to a SaaS model but need assistance doing so. Section 2 discusses the necessary background information for understanding the SaaS model and the associated changes in software requirements. Section 3 describes the tasks involved in the requirements engineering process in a system utilising SaaS. In Section 4, the necessary modifications to a requirements engineering process are outlined, and a transformation of this type is organised as a system. In Section 5, the established technique is dissected and discussed utilising derived best practises and concepts. Section 6 discusses the limitations of the proceeding, while Section 7 contains both conclusions and recommendations for future study.

I. BACKGROUND

a. Software as a Service

Software has been created with a focus on the supply side for an extended period of time . A software vendor invests time and effort in the process of requirements elicitation for a specific issue, develops and tests the software, and then makes the final product available for sale on the market. After acquiring a licence for the software product, the customer or the software vendor's support staff instals a copy of the software on the customer's infrastructure. Small software updates are often deployed and updated via an Internet interface and are included in the initial purchase price. This traditional paradigm is being challenged by the SaaS model, a new trend in software engineering that is gaining popularity in the 21st century [1]. The customer of SaaS-delivered software purchases a usage right for a set period of time. In exchange, the vendor gives access to the online service, which is sometimes accompanied by a variable individual access limit based on the pricing plan selected by the user. Since its first appearances in research in the early 2000s, software as a service (SaaS) has gained steadily more attention from both a scientific and a business perspective. The SaaS model represents a substantial paradigm shift in the way software is created and produced. Providing software as a service as opposed to selling it as a product is a distribution strategy that influences business concerns such as the time required to bring a product to market, the extent of client involvement, and release cycles . On the other side, the change toward a service-oriented approach in software development requires significant paradigm modifications. The SaaS model decreases software complexity by using services as the fundamental organising principle . SOA, often known as service-oriented architecture, is the fundamental notion underlying software design. SOA is a sort of architecture in which loosely connected but rigorously segregated software components (usually different business activities) communicate as composite services via public interfaces. This enables the binding of components in a scalable manner only when those components are necessary. The SOA standard makes no assumptions regarding the underlying platform and does not define how services should be organised or secured . These services are made available to consumers by service providers, who are responsible for building and implementing the service infrastructure and providing the interface description for internet access (web-based). To publish integration-ready services and to search for such services, a common service directory is required

978-81-933187-0-6 © 2023 SEEEPEDIA.ORG
Society for Engineering Education Enrichment

(see Figure 2). Services can be said to be composed of other services in a recursive manner [3].

a. Changed Requirements

In contrast to the conventional SaaP model, the SaaS model employs a unique infrastructure and offers a variety of access and distribution choices (see Section 2.1). When transitioning a software product to a SaaS model, it is customary to retain the majority of the software's functionality [9]. As a result, the disparities between the software engineering process and other processes that influence software development, including as operation, management, and architecture , are reduced to nonfunctional requirements [2]. The disparities in non-functional needs previously described can be largely related to the following three factors:

1. Subscription-based software must always be hosted in the cloud, either by the software vendor or by a third party that provides platform as a service solutions (see Section 1). There are just a few of very large companies that offer software services that mix the PaaS and SaaS components under one roof. One such company is Netflix, a streaming video on-demand provider. These companies function as their own operations' platform suppliers.

Typically, this type of software is distributed in the form of a web application that utilises the Internet and its associated protocols for data transmission.

A considerable portion of software given as a service to clients is delivered as browser-supported applications This indicates that the customer's device does not require any specialised software in addition to the previously installed web browser.

Due to the fact that the location of the server determines various legal aspects, including those pertaining to data protection laws and a company's compliance regulations, factor 1 shifts the emphasis of the non-functional requirements to security, data confidentiality, privacy, and compliance. Moreover, a cloud service provider is more likely to be the target of a security breach than a decentralised structure or a corporation's private network. Even while professional cloud service providers are more difficult to successfully attack, there are still safety issues that must be taken into account. The majority of differences in non-functional needs are directly attributable to Factor 2: multi-tenancy, user concurrency, configurability, scalability, dependability, performance, availability, compatibility, interoperability, portability, efficiency, and promptness. Other characteristics include continual evolution, increased stakeholder interaction, and heightened usage monitoring [7]. Factor 3 influences the unique aesthetic and user interface design requirements, as well as the limitations of browser-supported programmes.

RELATED WORK

Bennett and colleagues [3] discovered software development trends that were influenced by the advent of the Internet in 2000. They envision a future in which software is adaptable, flexible, interactive, and customised, and software engineering is demand-driven, service-oriented, and centred on the elicitation of needs. In their concluding part, the authors reach the conclusion that future research should concentrate on the necessary modifications to software engineering techniques. Papazoglou released the seminal work done on the core concepts underlying SOA. He described the effects of SaaS on software engineering and business operations as one of the first authors to do so. Papazoglou concludes that the SOA necessitates substantial modifications to programme design. Olsen, the author of , has performed research regarding the necessary paradigm shifts from a business perspective. He discusses why a client connection based on a SaaS-based software system differs significantly from one based on a SaaP-based software system. Olsen is responsible for the update mechanisms since they require a long-term commitment from the vendor and allow for enhancements to be made without disturbance. In addition, the author emphasises the benefits that regular upgrades and modular architecture provide to consumers. In their research [1], Armbrust et al. present definitions for a range of cloud computing properties. They define the function of SaaS, present a list of benefits as well as obstacles, and show how to avoid the obstacles. Since the publication of these seminal works, research has advanced significantly. In their research, Kumar and Sangwan present typical software engineering process models and key concepts. (e.g. iterative development). They are continuing to compile the qualities that distinguish the development of web-based applications from those of conventional software. The authors feel that continuity of the process is the most crucial issue, which needs a methodical, iterative, and reiterative approach. In addition to a list of the attributes and characteristics of web-based applications, they provide a highly generic adaption of a conventional software engineering process model towards a model that is suitable for the creation of web-based apps. However, as a result of this, the authors do not provide a specific procedure that may be used to create such apps. Furthermore, Nogueira da Silva and Lucr'edio conducted a complete literature review. They noticed that interest in the topic had increased over the past several years. According to them, the lack of standardisation is the greatest obstacle cloud-based software engineering confronts today. For instance, selecting a PaaS provider may result in platform lock-ins, which hinder clients from migrating to a different service provider fast. In addition to providing definitions of SaaS and SOA, the authors group SaaS

978-81-933187-0-6 © 2022 SEEEPEDIA.ORG
Society for Engineering Education Enrichment

developers into a variety of categories and explore the challenges they confront. In terms of rebuilding software for a new platform, they conclude that there is a need for additional research into the formalisation of a thorough reengineering process. Research on the development of software as a service (SaaS) is analysed and classified by Balian and Kumar [2]. In addition to research on migration and reengineering, they include works that examine the process of development from the bottom up. In addition, the authors examine studies on quality models for SaaS and conclude that adaption of software engineering process models, quality models, and metrics for SaaS is inadequate. They reach this conclusion after discussing studies on SaaS quality models. In comparing these two models, the most recent and significant work in the field of comparing the software engineering processes of the SaaP model and the SaaS model have been made. Tariq et al. address the impact of executing an application in a cloud environment on its prerequisites. They describe any technological requirements that are non-functional, any legal concerns, and any other issues arising from data management. The authors then categorise these topics and identify the new cloud service provider among the stakeholders. As a result, they propose extending the Capability Maturity Model Integration (CMMI) reference model with a checklist for the newly included stakeholders. The research undertaken has provided in-depth descriptions of the SaaP model and the fundamentals underlying the SaaS paradigm. The transition of a service-oriented system to cloud-based software that is modelled after the SaaS paradigm has also been the subject of current research. On the other side, there is no process assistance for turning an existing software product into a service-based application. In addition, we were unable to identify any migration solutions capable of addressing the discrepancies in the requirements elicitation procedure. This article aims to fill this void by providing a logical and generic approach for migrating traditional software products to the SaaS model in a sustainable and efficient manner. The key focus of this method is the collection of updated requirements.

REQUIREMENTS ENGINEERING PROCESS FOR SAAS

a. Differences between Processes

This section will focus exclusively on the requirements engineering procedure. However, certain variables influence several phases of the software engineering process, and others are, from the perspective of the requirements, merely incidental considerations. The list of differences between the traditional requirements engineering process and the SaaS-based requirements engineering process covers all components, despite the fact that this was done to provide a full overview of the differences. It is crucial for the software development process in general to have a thorough understanding of the requirements and their significance. In comparison to the SaaP model, the SaaS model comprises a greater number of stakeholders. According to Kumar and Sangwan, these persons include, among others, analysts, graphic designers, clients, marketing professionals, and security experts. However, the requirements engineering approach must also accommodate a greater diversity of stakeholders. As described in Section 2.2, software as a service (SaaS) often involves a higher degree of customer involvement and the formation of long-term links between the SaaS provider and the end user . The user is incentivized to offer feedback either directly or indirectly through usage tracking since an improvement to a product can be predicted, and the user would benefit from it without incurring additional expenditures and in a predictable amount of time. As the software resides centrally on the company's servers as opposed to the customer's infrastructure, the integration of bug fixes and new features is seamless and uninterrupted. They are also incorporated without any delays because time to market is drastically shortened due to the fact that new versions are published early and regularly and are not regarded as a separate software product. In addition, there are no time delays in their integration. After their implementation, the end user will be unable to distinguish between updates and bug fixes. These less disruptive upgrades, which respectively address fewer problems but occur more frequently, necessitate a reduced degree of retraining for the end user. Additionally, the centrally hosted, multi-tenant software as a service offers extra alternatives for testing newly installed features. The level of acceptability can be determined by exposing the feature to a subset of all users and waiting for their input afterward. It is also possible to offer two or three distinct variations of a feature to a variety of user groups. This allows users to compare features and choose the implementation with the highest user acceptability.

a. **A systematic transformation of the requirements engineering process follows these steps:**

In the first phase, a paradigm shift must be formed in relation to the requirements' great variability. In the context of software-as-a-service (SaaS), developers who are accustomed to working with traditional software products will need to adjust to the reality that needs change. The proximity to agile development and the advent of new elicitation techniques contribute to the volatility of the requirements.

The second step involves incorporating requirements engineering into an iterative and incremental software engineering methodology. This style of software engineering is not unique to service-based software development; it is also used in conventional software

978-81-933187-0-6 © 2023 SEEEPEDIA.ORG
Society for Engineering Education Enrichment

development. It is one of the distinguishing properties between the two types of software. Due to the ever-changing nature of the requirements and the frequent release cycles, however, such iterations and continuous software increments are necessary.

Step 3: Using rigorous methodologies, identify the stakeholders and rank them according to their significance.

Through integration, customers are integrated into the requirements engineering process in the fourth phase. To maximise the benefits of the switch to SaaS, feature and bug report invitations are an imperative must. The end users must have the sense that their engagement can influence future feature additions and bug resolution speed.

Implementation of user input collection tools is the fifth phase (e.g. usage monitoring, feedback forms). This is required to incentivize customers to offer feedback (see Step 4), which entails establishing such a culture. This can be achieved by placing feedback buttons on particular features, offering feedback forms that are accessible on the side of the screen, and utilising the multiple usage monitoring capabilities that cloud software offers.

The sixth phase is to provide methods for seamless update integrations. As stated previously, one of the key benefits of cloud-hosted software is that the designers of the programme retain control over the distribution process. Therefore, the integration of updates is highly advantageous: The new software components need to be installed once, and on a trustworthy server environment, which is the company's cloud server rather than the client's infrastructure. In addition, the high frequency of small updates makes integration straightforward without the need for downtime. This is owing to the fact that there are proportionally less lines of code and database design changes. Compared to lengthier maintenance outages experienced by the entire system, small outages experienced by individual system components are less noticeable.

The seventh phase is to develop support for software variants per user group for acceptance testing. The new requirements engineering method enables the production of many versions with uncertain acceptance and the distribution of these variants to various user groups. This was made possible by the procedure's capacity to generate several variants. Acceptance testing can then be conducted using the processes specified in Step 5.

DISCUSSION AND BEST PRACTICES

This paper's major purpose was to present a strategy for migrating from one type of software development process to another, as well as to describe the differences between the requirements engineering process of a standard software product and the process of a software service. If you adhere to the recommended methodology, you will be less likely to overlook to include crucial modifications in the requirements engineering process. For those who are considering relocating a software product but have not yet made a choice. As a result, the methodology employed in this study offers benefits not previously found in the relevant body of research. In order to adapt the transformation approach, we have created a list of best practises for some of the following processes, which we identified by studying the relevant literature. Agile software development methodologies, such as Scrum, are ideally suited for Step 2, as they give the best fit. Step 3's examination of the stakeholders is conducted effectively when socio-diagrams and power matrices are utilised. The authors of [6] provide a comprehensive summary of their methodology for identifying stakeholders and conducting impact studies. In order to encourage users to provide feedback, the measures outlined in Step 5 have been successfully implemented and can now be suggested. This includes integrating application-wide feedback forms and providing usage monitoring.

LIMITATIONS

This technique addresses the requirements engineering process, which is a subset of the overall software development process. Failure to migrate software products to the cloud may still occur as a result of a number of repercussions resulting from such a migration. The approach's primary emphasis on web-based service-oriented architectures is one of its disadvantages. The experiences that led to this problem were influenced by the findings of the literature review. The vast majority of studies do not differentiate between software as a service (SaaS) and web-based systems, making it extremely difficult to develop a transformation plan applicable to a variety of additional consumer interface types.

CONCLUSION AND FUTURE WORK

The manner in which software engineering is currently practised has experienced tremendous change. The emergence of software as a service (SaaS) is one of the reasons why software engineering techniques must be modified. In many ways, the elicitation of requirements for software delivered as a service differs significantly from that of traditional products; some of these differences are rather fundamental (see Section 2.2). On the other hand, the differences have a multitude of advantages, such as longer-term client connections, a more targeted use of resources, and more frequent feature updates. The requirements engineering process must be transformed when transitioning from an

978-81-933187-0-6 © 2023 SEEEPEDIA.ORG
Society for Engineering Education Enrichment

existing software product to a model based on software as a service (SaaS). This work has offered a methodical strategy for this transformation that may be used by software engineers who wish to modify the manner in which they establish and satisfy their software system's needs. This approach can be utilised by software developers because it has been presented in this article. In the near future, research will be conducted to determine how to combine the benefits of this novel requirements elicitation method with those of agile software engineering techniques, which already emphasise iterative and incremental development.

REFERENCE

[1) Schäfer, J., & Lichter, H. (2016). Changes in requirements engineering after migrating to the software as a service model. Full-scale Software Engineering/ Current Trends in Release Engineering, 25.

2) santhoshkumar, s., & ramya, g. (2017). changes in necessities trade after migrating to the saas model.

3) Chauhan, M. A., & Babar, M. A. (2011, July). Migrating service-oriented system to cloud computing: An experience report. In 2011 IEEE 4th International Conference on Cloud Computing (pp. 404-411). IEEE.

4) Baliyan, N., & Kumar, S. (2014, August). Towards software engineering paradigm for software as a service. In 2014 Seventh International Conference on Contemporary Computing (IC3) (pp. 329-333). IEEE.

5) King, T. M., & Ganti, A. S. (2010, April). Migrating autonomic self-testing to the cloud. In 2010 Third International Conference on Software Testing, Verification, and Validation Workshops (pp. 438-443). IEEE.

6) Melegati, J., Goldman, A., Kon, F., & Wang, X. (2019). A model of requirements engineering in software startups. Information and software technology, 109, 92-107.

7) Vyatkin, V. (2013). Software engineering in industrial automation: State-of-the-art review. IEEE Transactions on Industrial Informatics, 9(3), 1234-1249.

8) Rodriguez, J. M., Crasso, M., Mateos, C., Zunino, A., & Campo, M. (2011). Bottom-up and top-down cobol system migration to web services. IEEE Internet Computing, 17(2), 44-51.

9) Rodriguez, J. M., Crasso, M., Mateos, C., Zunino, A., & Campo, M. (2011). Bottom-up and top-down cobol system migration to web services. IEEE Internet Computing, 17(2), 44-51.

10) Mateos, C., Crasso, M., Rodriguez, J. M., Zunino, A., & Campo, M. (2015). Measuring the impact of the approach to migration in the quality of web service interfaces. Enterprise Information Systems, 9(1), 58-85.

978-81-933187-0-6 © 2023 SEEEPEDIA.ORG
Society for Engineering Education Enrichment