Expertise
and insight
for the future

Konstantin Lobkov

Methodologies of Acceptance Testing in SaaS Environments

Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor's Thesis

09 November 2019

Author	Konstantin Lobkov
Title	Methodologies of Acceptance Testing in SaaS Environments
Number of Pages	33 pages
Date	09 November 2019
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	Smart Systems
Instructors	Antti Piironen, Principal Lecturer

Modern business is more competitive than it has ever been before due to the increasing spread of technological achievements. Therefore, when all available options are equally competent in terms of provided functionality, the quality of the delivered service becomes the primary criteria of choice for the customer. This factor becomes especially important for any SaaS company as such a business model assumes making architectural and infrastructural decisions on the behalf of of the client. Thus, a thorough acceptance testing of a software application is as important as developing it.

The study focuses on researching different methodologies of software acceptance testing and their impact on existing business processes. The study is primarily dedicated to acceptance and performance testing carried out after major software upgrades and customized minor changes. The empirical part was done in a Finnish SaaS company, which provided an opportunity to try different approaches and improve ineffective processes.

Different reoccurring obstacles in effective testing processes were identified. Major encountered challenges include lack of communication and technical complexity, resulting in inefficient planning. Several vectors of improvement were suggested to improve the software quality.

Keywords	

Contents

List of Abbreviations

SaaS – Software as a Service

IaaS – Infrastructure as a Service

PaaS – Platform as a Service

CRM – Customer Relationship Management

ERP – Enterprise Resource Planning

UAT – User Acceptance Testing

OAT – Operational Acceptance Testing

ISTQB – The International Software Testing Qualifications Board

1 Introduction

Exponential growth and widespread adoption of cloud computing technology has made more business opportunities possible, where Software as a Service (SaaS) plays a very important role [1]. Nowadays, SaaS has already found common use in certain areas, such as “computerized billing, invoicing, human resource management and service desk management” [2]. In the SaaS model, the application and its relevant data is hosted in the cloud by the software vendor, enabling customers to use the application over the internet and pay only for the actual usage [3]. Simplicity of installation and configuration, high reliability and availability greatly contributed to the popularity of SaaS. Consequently, such interest in SaaS also means increased demand in software testing, specifically tailored for SaaS environments.

This study was done at a SaaS company (later referred to as case company), which provides supply chain solutions for store replenishment, workforce optimization and space planning, mainly targeted at retail companies. The application itself is composed of two distinct parts: a core product and a customizable part. A centralized team develops the core product, which provides the base platform for application functionality, similar for each installation. The customizable part is implemented for each client separately by the responsible customer project team.

In the case company, each customer has its own dedicated project team, which is responsible for maintaining their respective SaaS installation. Each person involved with customer related work can potentially participate in several projects at the same time. This approach combined with high configurability of the software provides high operational flexibility, which often results in a unique ecosystem of solutions being developed within each new project, affecting multiple areas of continuous service at once. Eventually, with the increasing number of new customers, the differences between various projects begin to take a logistical toll on case company’s business processes, increasing the workload and, as a result, potentially reducing the quality of the end product for individual clients.

Another layer of complexity for the existing issue is introduced by the case company’s bigger customers. Naturally, large businesses have their own strict service level requirements for SaaS products, which include certain software testing standards. This

problem manifests even more with clients that also happen to be one of the company's older projects. Such cases often have an extremely customized SaaS installation with their own specific business routines, which inevitably results in high complexity of any related software development or maintenance. Consequently, carrying out a software test of any type becomes a very non-trivial assignment, which may greatly differ from a similar procedure performed on any smaller client.

In spite of the ongoing company-wide effort to create generalized guidelines for building and maintaining a standardized SaaS environment, there is still a lot of room for improvement. The present study focuses on researching the inner acceptance testing processes accompanying each software installation or modification, putting the emphasis on user acceptance testing and performance testing. The primary goals of the thesis include identifying existing issues in current acceptance processes and providing several suggestions on how the processes can be made more efficient.

Therefore, the thesis aims to answer the following questions:

1) What are the business benefits of efficient testing processes?

Efficient acceptance testing procedures lead to better test coverage performed within the same time window and with same human resources. This eliminates the need for "time sacrifices" and reduces the potential for human errors, as a result leading to better service quality, which is a core requirement for a successful SaaS business.

2) What are the challenges in the current acceptance testing processes?

While having every possible exceptional case covered is an ideal outcome of software testing, allocated project time is unfortunately limited and thus this result is impossible to achieve. Some scenarios generally have higher priority, for example, ensuring that the environment in question can be reached by the client side to begin with, but this does not mean that less critical test cases can simply be neglected. Time sacrifices and inefficient prioritization naturally build up, which inevitably leads to worsening SaaS delivery quality in certain aspects. Non-uniformity of the software stemming from its natural flexibility and various communications gaps between internal team and with the end users pose another major challenge to existing processes, further increasing the complexity or allocating available resources in an efficient way.

3) How can current acceptance testing practices be improved?

Any change or improvement, regardless of how much value it could potentially bring for the company, is challenging to apply on an existing business model. There are always some hidden costs involved, which could be related to limitations and other requirements to customer contracts, legacy code that could not be easily replaced, etc.

The thesis consists of four major sections. The introduction gives a short background on the main goals and motivations of the study. This part is followed by a background overview of the subject, which primarily focuses on technical description of relevant topics such as SaaS structure, cloud computing and software quality. The following section, empirical study, describes and analyzes software acceptance testing processes in a real SaaS company. Conclusion provides a summary about the study results and aims to answer the questions posed in the introduction.

2 Background

This chapter consists of four parts. First, in Section 2.1 the concept of Software as a Service is presented. Section 2.2 provides a brief definition of software quality and lists several benefits of keeping it up to the customer's expectations. Software testing and its different types are discussed in Section 2.3. Finally, the summary of the literature review is given in Section 2.4.

2.1 Software as a Service

To better understand the specifics of the SaaS development cycle, the concept of cloud computing and its different service types are briefly explained in Section 2.1.1. Thereafter, Section 2.1.1 presents the benefits of the SaaS model from the customer's perspective and lists several data models of a SaaS application.

2.1.1 Cloud Computing

In a widely used traditional model, everything is owned and controlled by the organization using the software: the organization owns and maintains the necessary infrastructure and purchases software licenses from vendors, which results in exponential growth of capital expenditure. A rapid increase of the number of personal computers and transformation of the internet into its modern state have signaled the evolution of cloud computing. The cloud provides clients with a remote access to practically unlimited computing power over the internet with the possibility to pay only for what they actually use.

Different web applications are commonly used to provide the end user with an entry point to cloud services. Three widely known service models include Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). The relations between the different models and the cloud stack are demonstrated in Figure 1 below.

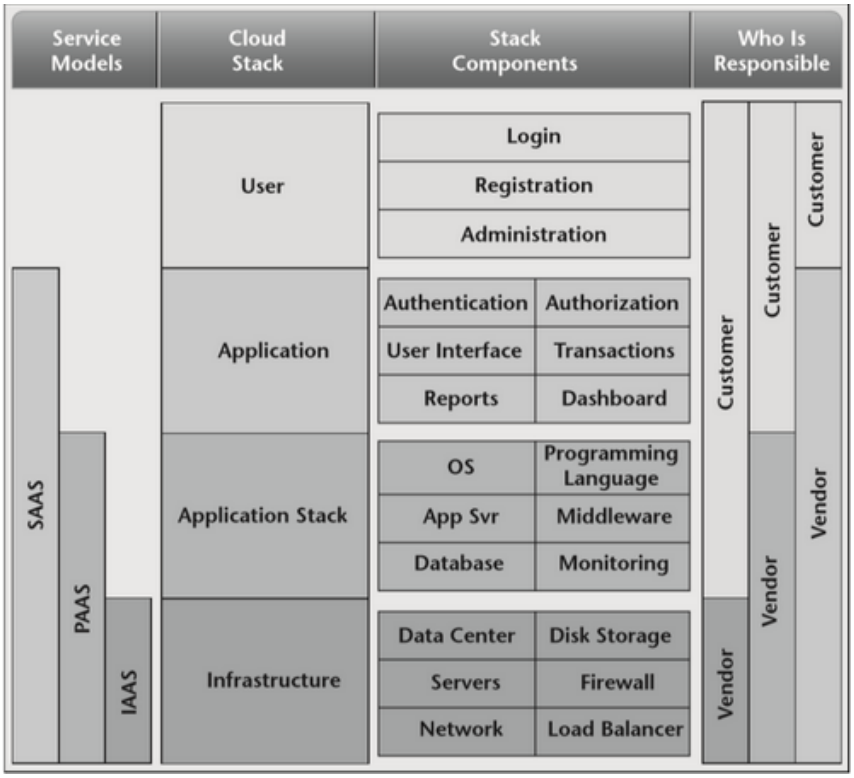

Figure 1: Structural comparison of known cloud service models. [4]

IaaS is a cloud service model where various physical infrastructural components are delivered like data storage space or servers. Such a model allows the organization utilizing the service to save operational costs by using the vendor's own infrastructure while retaining control over software application building and management.

PaaS is an advanced form of IaaS: in addition to providing infrastructure, the PaaS vendor delivers an application platform. In such a model, the client has no need to develop or maintain basic application components, such as operating systems or databases. Delegating this responsibility to the PaaS vendor allows the organization to concentrate on building the required business logic and performing minor configurations. [4]

The Software as a Service model provides the organization with a complete software application. In such a scenario the software vendor takes care of everything ranging from infrastructure setup to software configuration and deployment, while the organization is only allowed to perform limited administrative tasks e.g. changing user-specific settings or managing user accounts. [4]

2.1.2 Outline of Software as a Service

As briefly mentioned in preceding section, Software as a Service is a cloud service model where access to a software application is delivered to the customer via the internet. The traditional software model assumes the customer to be the primary maintainer of said software: related infrastructure and installation of licensed software are the client's responsibility. Therefore, the license price of software is just a fraction of all applicable additional costs combined [4]. Figure 2 below shows that while SaaS may initially look like a less desirable option when comparing only known immediate expenses, it is very likely to actually be a more profitable solution, when taking hidden costs into consideration.

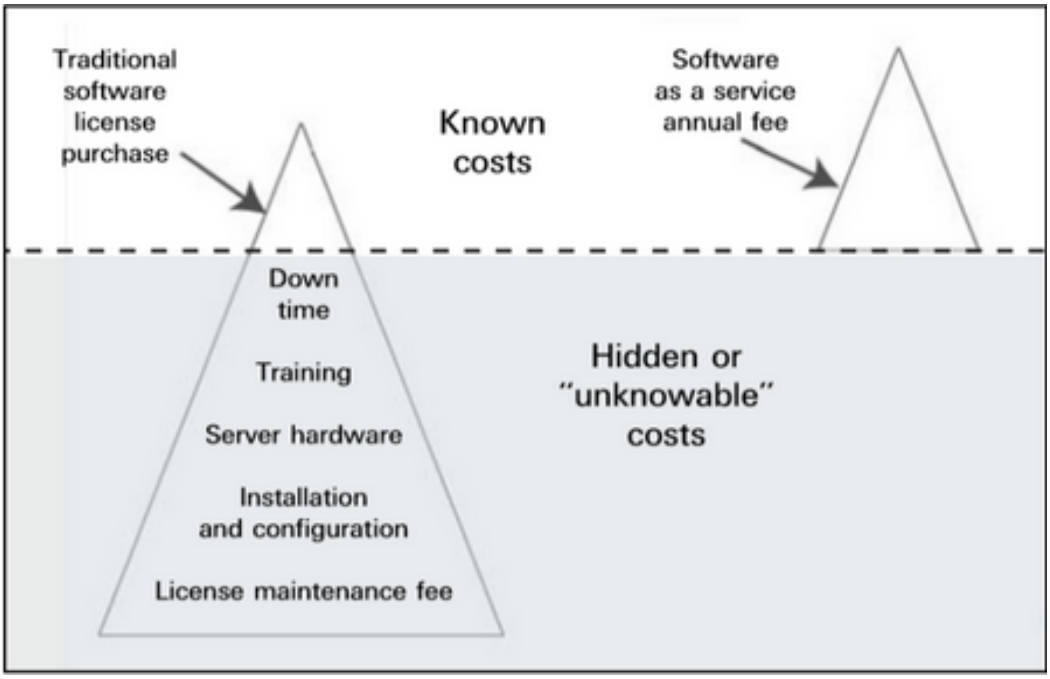

Figure 2: Comparison of costs between traditional software model and SaaS. [5]

Higher cost is not a single disadvantage of the traditional model compared to SaaS model. According to Waters [5], 85 percent of all software projects end up being delayed and development costs quickly surpass the predicted budget, whereas SaaS software is more reliable and allows for a much quicker setup, allowing the customer to start benefiting from the project faster. Furthermore, in traditional model managing the software still remains primarily the client's responsibility, including tasks like software upgrades, access and user control, hardware purchase and installation when required. In SaaS model, all of these examples are instead the vendor's responsibility and the

customer simply gets the end product, knowing exactly what the final cost is going to be. [5]

SaaS applications are usually used in areas where the organization does not have any experience, providing an easy way to outsource such functionalities but at the same time remain in control of them. Example areas for such applications can include customer relationship management (CRM), enterprise resource planning (ERP), workforce and financial management. [4]

Several models of computing power distribution exist in the SaaS model: single-tenancy, multi-tenancy and hybrid model. Multi-tenancy means that several customers share the same application and database servers, whereas single-tenancy model provides a customer with dedicated application and database. Figure 3 below demonstrates “total isolation” model provided by single-tenancy. The benefits of such an approach include a higher degree of privacy and independence as only one organization has access to the application and data servers at any time. Furthermore, such a system has high scalability, as computing power is not used by other tenant, allowing unrestricted access to available resources. Unfortunately, each customer requiring their own resources greatly increases operating and infrastructural costs, and increasing the total number of servers introduces management complexities. [4]

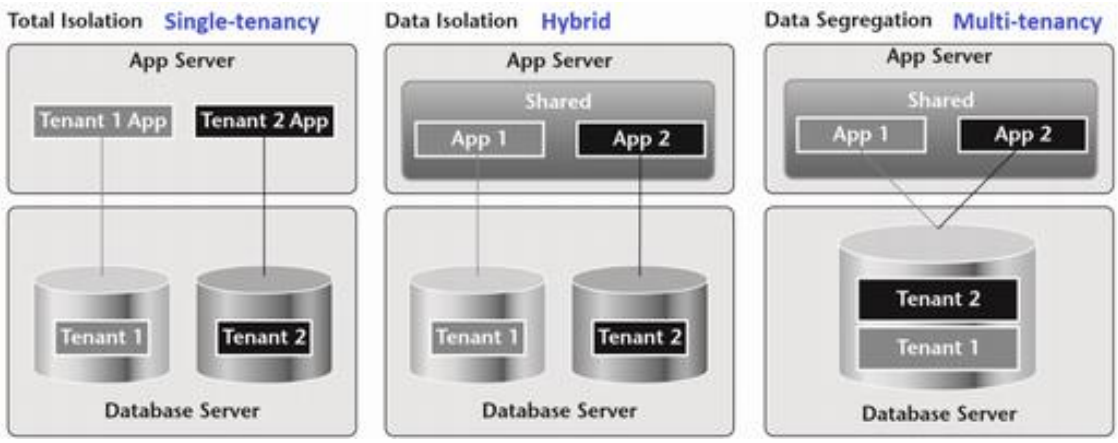

Figure 3: Different tenancy models. [4]

The multi-tenancy model depicted in Figure 3 is also described as the “data segregation” model. In this scenario, both the application and data servers are shared by two or more tenants, but operate different database schemas. Such an approach is considerably more cost-effective and easier to maintain in a longer term than single-tenancy,

because less servers are required. Multiple organizations share computing resources, which also allows the highest rate of capacity reuse. However, sharing the application server may have a negative impact on the performance. Moreover, the scalability potential of an individual tenant suffers as well. [4]

The hybrid model shown in Figure 3 is presented between single-tenancy and multi-tenancy models, as a compromise “data isolation” scenario. It is a mixed model, where the approach to the application side is similar to that of multi-tenancy, whereas the database servers are dedicated to each individual tenant as in single-tenancy. This model allows reducing the complexity of the system and related costs, but also achieves privacy and independence benefits. The vendor can select different tenancy approaches in different customer cases. [4]

The increasing popularity of the SaaS model stems from a multitude of factors. Nowadays, each user most likely has a connection to the internet and a web browser, which all use the same web service and communication protocols. This fact makes it incredibly easy to distribute the software over the internet to multiple users, who then can just access the data via web applications. This means that it no longer matters where thee data is located physically, it can be secured equally regardless if it is local or accessed remotely. The maturity of the software business in general is an important factor as well. As both sides grow more familiar with their rights and obligations in such a business, contracts become easier to make. [5]

2.2 Software Quality

There is no unambiguous definition for quality. Two major ideas dominate the field of software quality: in Crosby’s product-centric view a quality product fulfills its specified requirements and is planned in special way to avoid defects, whilst Juran’s user-centric perspective points that a good quality means that a product has to be usable and meet all of its user’s needs. [6]

According to Kavis [4], not each defect leads to a negative impact on the quality of software. Measuring the absolute number of defects uncovered during testing and development process should be replaced with measuring the relative impact of each defect on end users. Improvements that address quality should focus on deployment

problems and errors that end users could potentially see [4]. Several points on the importance of delivering good quality software are shown in Table 1 below.

Table 1: Main reasons to develop good quality software. [5]

Reason	**Explanation**
Maintaining reputation	Defective software may quickly lead to a negative customer reputation
Successfulness	Quality plays a major part in evaluating success of a project
Keeping customers	Missing technical requirements, security issues and bad customer support experience are three biggest reasons for customers to stop using SaaS
Technical debt limitation	Low quality products require more maintenance and therefore are usually more expensive in longer term than good quality products
Achieving software certificates	In many areas, specific quality requirements must be met by software to achieve a certification important for business.
Getting new deals	Having certain certificates serves as a proof that the company can adhere to common software development standards, and this helps the vendor in winning new clients.
Legality	Many countries force software vendors to follow specific requirements, punishing organizations for software which is risky to use by its users.
Ethics	Purposefully releasing low quality software is an unethical way of working.

When an organization purchases SaaS, they expect they will be provided with a finished product, which can be immediately integrated into current business processes. Therefore, extensive testing and quality assurance processes must be in place to ensure the software meets the users' expectations [3]. As a web browser is used to access such software, data security and general service availability become more important factors for evaluating SaaS quality [8]. Different factors which are considered important during SaaS quality measurement are demonstrated in Table 2 below. It should be noted that it is highly improbable and in the absolute majority of cases im-

possible for a software application to meet all the existing requirements, so these requirements should be assigned different priority levels. [7]

Table 2: Factors influencing quality measurement of SaaS. [7]

Factor	**Explanation**
Features and functionality	Software meets customer requirements from functional perspective
Contractual flexibility	Business contract can be adjusted if the need arises
Software scalability	Software can be scaled to meet new customer's needs and also scaled based on runtime requests.
Rapport	Good personal relationship between the vendor and the customer, receiving tailored support from the vendor
Recoverability	Software can quickly recover from a failure
Reliability	Service is provided in an agreed schedule with minimal interruptions in operational flow
Vendor responsiveness	Customer support response times and availability
Software responsiveness	High software performance and availability
Security	Promised security level kept
Usability	Software is simple to understand and use
Efficiency	Software is effective to use
Maintainability	Software is easy to modify, test and maintain
Portability	Software is easy to install
Isolation	Change in one environment should not affect other environments

Each of the factors listed in Table 2 causes a noticeable impact on the software quality in general. However, the weight of this impact differs in each business case as the customer's requirements can also vary. Feature-rich software with extensive functionality can be the main goal of one customer, while another values reliability and security of their data above everything else. No uniform approach exist in evaluating the importance of each factor and, while common patterns in recognizing good quality software exist, each business scenario is unique and requires a tailored approach for the best results.

2.3 Acceptance Testing

This section consists of five parts. First, Section 2.3.1 defines the concept of software testing and explains the importance of a comprehensive testing plan. Second, Section

2.3.2 covers user acceptance testing, followed by Section 2.3.3 where various challenges this testing introduces are described. Section 2.3.4 lists several subtypes of performance testing and the necessity of conducting such test. Finally, several challenges of performance testing are demonstrated in Section 2.3.5.

2.3.1 Definition of Software Testing

A process where the software is continuously executed and brought to a failure state in order to find defects is the definition of the term “software testing”. Software testing is required to validate the software quality and to ensure that all customers’ functionality requirements are met. Software tests consist of different tests cases, which are collections of input and output sets. To be considered passed, the observed output of a taken test case should match the expected output, in other case the test has failed. Different cases which are relevant to each other can potentially be grouped together as a test suite. [9]

Software testing is the most important and the most critical step in the entire software development process, which at the same time consumes the most resources and requires huge time investments. Typically at least half of development work is spent on various testing. Consequently, due to such constraints, it is nearly impossible to perform a complete software test. Therefore, testing can be used to detect and subsequently fix various defects, but it can confirm the software is completely defect-free. [9]

Software testing is required during the whole development life cycle. First of all, unit tests are executed for a software component that has been somehow modified. In this case, such tests are performed in isolation from the rest of the software to ensure that the component’s functionality remains true to the specifications. Once this is confirmed, integration tests are carried out. The difference with the previous step is that integration testing aims to ensure different components of the software work correctly together. The next level after this step is system testing which focuses on testing the entire software against the customer’s requirements as one whole product. The final step of software testing is acceptance testing. All the tests before this step are executed internally by the vendor in specialized testing environments, whereas acceptance testing can be performed by both the customer (for e.g. user acceptance testing or UAT) and the vendor (e.g. operational acceptance testing or OAT). Before the software is taken to pro-

duction use, it is tested against customer's specific requirements and is ensured that it works correctly in customer's own environment. [7]

Another form of acceptance testing is field testing, which is also commonly referred to as alpha or beta testing. For this type of testing, the vendor builds a pre-release software version to be utilized either in the customer's own environment or in the vendor's internal testing environment. The main goal of this process is to find defects that would have been otherwise missed if tests were executed only in the development environment. Testing becomes much more useful with test results being much more informative if these fields testing environments represent the specific market the software is targeting. Alpha tests are executed in the software vendor's environment, whereas beta tests are performed by the customer, this being the main difference between alpha and beta testing. [7]

2.3.2 User Acceptance Testing

UAT, which is also often referred to as "beta testing", is a process of verifying whether the developed solution works for the customer and meets the customer's expectations. It is not system testing, which is performed primarily to make sure the software does not crash and generally meets the requirements described in the technical specification.

Once all the previous steps of software testing (unit, integration and system testing) have successfully passed the quality control, it may seem redundant to perform an additional step in the form of acceptance testing. However, it is required because developers include features and change the software according to their own understanding of what the customer has requested. Ineffective communication can easily lead to a situation where the implemented functionality is not what the client has actually asked for. Different changes in the contract and over the course of the project may also not be effectively communicated to the developers. Uncovering and fixing defects resulting from this problem is much cheaper and less time consuming during the testing stage of development, whereas discovering that the software does not perform the functionalities mentioned in the customer contract after it has been signed off by the quality assurance team and deployed to production is going to be considerably more costly and can potentially lead to legal consequences. [7]

The list below shows the main prerequisites that must be met before user acceptance testing can be performed. Generally, it means that all system components work well together, the system does not crash and is supposedly able to provide all requested functionality. [11]

- Business requirements from the customer must be ready and available
- Software application code must be complete and fully developed
- All previous testing phases must be successfully completed
- No critical defects or other blockers are allowed, only minor "cosmetic" errors are allowed before UAT
- All reported defects should be fixed before the UAT phase begins
- UAT environment must be ready and the testing team should clearly communicate that the system is ready for test execution

2.3.3 Challenges in User Acceptance Testing

In engineering and in the software development field in particular acceptance testing is a test carried out to evaluate whether the requirements of a contract or customer's specification are met. Performance testing is commonly conducted as part of the whole acceptance testing process as customers of SaaS often put specific requirements on the system's overall performance, scalability, responsiveness and stability. The International Software Testing Qualifications Board (ISTQB) defines acceptance testing as "formal testing with respect to user needs, requirements, and business processes conducted to determine whether a system satisfies the acceptance criteria and to enable the user, customers or other authorized entity to determine whether to accept the system. [10]

Acceptance testing introduces certain challenges for the company aiming to successfully execute such test suite. As it was stated in Section 2.3.2, UAT is an extremely crucial and important step of the software application release cycle. Issues such as poor user involvement in UAT or reluctance to perform the testing, inefficient and improper test planning are among the most common problems software vendors face during the development and failure to properly handle them is likely to negatively im-

pact the quality of the software. Obstacles most commonly encountered during the UAT phase of software development are briefly listed and described below:

1) Improper test planning

UAT is naturally the very last stage of the entire software testing process and at the same is the most important. Therefore, continuous delays from preceding testing stages result in less time for user acceptance testing. Unnecessary pressure caused by these obstacles can potentially lead to a situation where the software is installed to a special UAT environment without even passing the functional testing step, which is almost guaranteed to cause inaccuracies or failures in the application. A solid test plan must exist, which should also contain different test cases based on real usage of the software. [11]

2) Insufficiently qualified UAT users

Inefficient planning of the testing process can lead to suboptimal delegation of testing responsibilities in an attempt to push the software release within specified time limits. Inadequately trained testers do not have the business or technical knowledge required to properly evaluate the behavior of the software in question. [12]

3) External communication gap

Unclearly communicated customer requirements or ambiguity in their requests can potentially lead to releasing an application which is not functional or completely unusable from the client's perspective. Good relations with the customer and general maturity of established development practices in the company both greatly contribute to breaching arising communication gaps and help to interpret the customer's needs in the most appropriate and beneficial way. However, sometimes the software just gets rejected by the customer for no apparent reason. Such a scenario is extremely rare and is usually a result of certain internal politics of organizations in question and can potentially be avoided by building positive and trustful relationships between the parties involved in UAT process. [13]

4) Internal communication gap

Generally, there is always a communication overhead between different technical and testing teams, especially if they operate at different geographical locations. Therefore, proper planning and effective communication is the key to effective test execution. Using special collaboration tools can be extremely beneficial in establishing an efficient connection between various teams, so that all participants can easily share testing notes and defect logs. [12]

2.3.4 Performance Testing

As mentioned in Section 2.3.2, performance testing is often included as part of the whole acceptance testing plan. Such testing is conducted to evaluate the performance of the software in terms of stability and responsiveness under a particular workload. It may also be utilized to measure or verify other properties and attributes of the software, such as its reliability, scalability and resource usage. If performance testing is omitted from the test plan, the software is very likely to experience the following issues: slowness and unresponsiveness of the system when several users are trying to use it at the same time, inconsistencies across different hardware or operating systems and otherwise generally poor usability of the product. Furthermore, applications released to the public with poor performance measurements due to negligence during the software testing stage are likely to accumulate bad customer reputation and fail to meet projected sales goals.

Table 3 below lists several types of performance testing that could be conducted during software development and provides a brief explanation for each.

Table 3: Types of performance testing. [14]

Type	**Explanation**
Load testing	Checks whether the application is able to perform under specified user loads. The main goal is identifying different performance bottlenecks before the application is integrated into a business

Stress testing	Measures software performance under extreme workloads which are unlikely to happen on a regular basis. The main objective is to identify the breaking point of a software application
Endurance testing	Performed to ensure that the software application is able to handle expected user load over a longer time period
Spike testing	Somewhat similar to stress testing, spike testing aims to evaluate application's response to a sudden large increase in traffic generated by users
Volume testing	Application's database is populated with a certain volume of test data in order to check software's performance with varying database volumes
Scalability testing	Is used to measure software's readiness and effectiveness in scaling up its calculating resources to handle possible increases in user-generated traffic. This helps in long term capacity planning for the overall system

Several types of performance testing demonstrated in Table 3 aim to measure the system's stability and responsiveness in different situations. While each test case is targeted towards its own relevant component of the entire application (e.g. user authentication service or data storage), all scenarios attempt to evaluate the system's readiness to extreme and unpredictable situations, which otherwise can cause major business risks if the application behaves incorrectly.

2.3.5 Challenges in Performance Testing

As it can be seen from the main goals of the existing performance testing subtypes, most software performance issues revolve around load and response time, speed and poor scalability of the application. Speed is widely considered one of the most important characteristics: a slow application will lose potential users, while a fast applica-

tion will attract more. Problems encountered the most often when evaluating the performance of software are listed below:

1) Long load time

Load time is a measurement of how long it takes for an application to start and generally should be kept as low as possible. The necessity to just wait until the software loads increases customers' dissatisfaction level as it reduces general usability of the software. [14]

2) Bottlenecking

Bottlenecks are certain blockers or obstructions in the system, which cause a negative effect on overall system performance. It commonly occurs as a result of a human error in coding or various hardware issues and it is not uncommon for just a single faulty component to degrade the entire system's performance. Common performance bottlenecks include disk usage, poor memory or CPU utilization, certain OS limitations, etc. [14]

3) Poor response time

Response time is the time it takes from the point of user inputting the data in the application until this application's response to the user's request. This should generally be very short as even a delay of half a second of an application in certain actions can be extremely frustrating for the end user. [14]

2.4 Summary

The cloud computing technology provides unique business opportunities for both software vendors and their customers. The most comprehensive model of the three main cloud computing models (IaaS, PaaS and SaaS) is Software as a Service (SaaS). This model is valued by the customer more than the traditional software model because of higher reliability, lesser cost and much simpler installation and integration.

It was established that software testing, especially in the context of SaaS, is an essential part of any successful software development project. The most important stage of the testing process is acceptance testing, which is executed after the software passes all relevant unit, functional and system tests. User acceptance testing (UAT), along with performance testing, are the two main focus points of this thesis, as the results provided by carrying out different test scenarios in their context are the closest to reflecting actual user experience of the software.

The objective of acceptance testing is to evaluate software's readiness for usage in a real world business scenario with real users. Different quality and performance metrics affect the customer's satisfaction level: software loading times and user interface responsiveness, provided functionality and its adherence to the requested specification, etc. Happy customers are much more likely to keep using the same SaaS solution for prolonged periods of time, which causes a beneficial business impact on the vendor company.

User acceptance testing is composed of sets of steps, which are used to confirm whether specific requirements are appropriate for a certain customer. The main difference with the functional testing is that while it aims to conclude that the software meets the requested specifications, a functional test does not verify that the solution actually works for the end user. Therefore, it is important to carry out UAT by people who will eventually be working with the software in a real world situation, as they are usually aware of certain business requirements unknown to developers. The language barrier between the end users and software implementers can potentially render the software unusable for the customer. The purpose of UAT is to minimize the effects of such communication misunderstandings and as a result to increase the quality of the end product.

Performance testing is widely considered a part of the acceptance testing as well. This step is as equally important as UAT, because slow software which is unable to handle multiple users well, negatively impacts the customer's daily work activities, especially if they rely on the application for performing critical business operations. The main goal of this testing is exposing system performance bottlenecks that can affect the stability and performance of the system. Human errors frequently result in inefficient blocks of code, which can potentially create such obstacles; however, faulty or untested hardware may also be the reason.

3 Empirical Study

This chapter is divided into five parts. Section 3.1 introduces the case company, provides a short technical background of the utilized software and lists several major challenges in establishing an effective software testing process. Section 3.2 explains the specifics of the acceptance testing process in the case company. The general workflow of the acceptance testing process is described in Section 3.3. Special importance of performance testing of the software in the case company and its current principles are shown in Section 3.4. Finally, Section 3.5 concludes the chapter and provides a summary of the previous sections, listing potential issues challenges to improving the existing processes.

3.1 Case Company

The case company provides various software based solutions in the retail planning area. It was founded in 2005 and since then addresses matters such as space and assortment optimization, supply chain optimization and workforce management. The scope of the present study is within the supply chain solutions area, which includes modules such as automated replenishment and forecast management.

The customers are provided with a single-tenant SaaS model, in which both software and, with rare exceptions, hardware are both hosted by the company. The ERP system of the customer is connected to the software application, to where master data, as well as transaction data, open orders and balances are transferred on a daily basis from the customer's side. The main function of the software is optimizing a range of order and forecast parameters, which in conjunction with previously received business data is used every day to calculate demand forecasts and purchase order proposals. The end users of client companies can use a web browser for accessing the web interface of the software, from where they can review, possibly modify and accept order proposals, which are sent back through the customer's ERP system to suppliers as purchase orders.

The main product consists of two separate parts: one represents the core functionalities, while the other consists of customer-specific modifications. The main part is offered to all customers and is functionally the same in each application environment. It is

divided into separate modules, which can be easily hidden or shown for the end users, depending on the type of license the client company has purchased. The customizations, on the other hand, are maintained separately from the core product and independently for each customer. These changes can be easily implemented via business logic configuration in software user interface, in integration layer between the software and the customer's ERP system or as custom scripts. The level of complexity and the amount of such customizations varied greatly between customers and depends on their business needs. Therefore, each software installation is unique in its own way and is utilized differently by different customers.

The core part of the application is undergoing a constant development cycle, where defects are found and fixed, and new functionalities get implemented. The development team executes the tests on the software during the initial development phase, and defect fixes introduced at later stages of the software release cycle are automatically tested by the QA team. However, these test sets do not take into consideration the customizable component of the application, because testing the software for each possible use case that can appear for each customer is technically impossible. Therefore, before deploying a version upgrade or introducing new business configurations, the software is also tested in the customer's special test environments. This testing is often performed manually by the responsible project team and the end users.

Manual testing is extremely dependent on people and often introduces "information silos", where only a limited number of people have deep knowledge and understanding of certain processes and implementations in a given project. As the company experiences business growth, the number of people to some extent involved with manual testing also grows, increasing the severity of the problem and introducing new challenges for the project. Current UAT practices must evolve and adapt if the company wishes to retain high software quality and avoid unnecessary costs and complexities.

As it has been mentioned earlier, performance testing is an integral part of acceptance testing in general. Main customers of the software naturally produce large quantities of data which needs to be processed as fast and reliable as possible. Thus, a large emphasis is put on overall performance of the software, especially for huge retail customers. All components of the software, including but not limited to specific hardware configurations, various environment settings, user load handling and data processing

speed are continuously tested in different scenarios. Some of the current practices can potentially be improved to increase the quality of test cases even further.

3.2 Acceptance Testing in Case Company

Since UAT stage is aimed to represent actual user experience with the software, such testing is therefore performed by project teams as they have direct communication with the customer and generally have a deeper understanding of possible issues that can arise in their specific use cases. Due to the nature of the application and its high capacity for customization, each use case and test plan is different. Testing all possible combinations and interactions between different functionalities and integration levels of the software by the development team during earlier testing stages is technically impossible, and thus client-tailored UAT plans exist.

Project teams performing the UAT consist of people with certain knowledge of business processes or technical implementations of their respective customer cases, but usually not directly involved in the development cycle. Delegating the execution of this testing stage to non-development teams allows managing time and other resources in a considerably more efficient way. First of all, such approach lets the developers focus on tasks that are actually important and relevant for the current business course of the company (e.g. building new functionalities or optimizing existing solutions), instead of testing whether the software works properly for all theoretically possible use scenarios. The goal of the development team is to ensure that the software works in general, not if it works for a particular customer. Moreover, people who have a deeper understanding of the specifics of their project can consequently compile and execute a more refined test plan. This leads to higher software quality and happier customers, and, as a result, contributes to building a trustful reputation on the market and to opening new business opportunities.

However, there are several drawbacks to this delegation of testing responsibilities, which spans into a multitude of related issues. For example, technical work performed for each client is not centralized and therefore is not sufficiently coordinated. Lack of standardization and software's general flexibility reinforce the internal communication barrier between different teams in the company. Customer environments, software application setups, settings, incoming and outgoing data characteristics, business work-

flows, tooling and even formatting for various reports can all potentially be different across different project teams. This diversity slows down knowledge and experience sharing between employees and greatly increases the time needed for implementing software changes. Furthermore, managing the company's infrastructure, service stack and coordinating related functional and operational teams becomes a task of continuously increasing complexity. It is not economically feasible and often downright technically impossible to automate the majority of required UAT cases, resulting in most of such testing to be executed manually. Human errors are more likely to happen under such circumstances, leading to worsening overall software quality.

The majority of test scenarios throughout the company and its customers aim to evaluate the same components of the software. Scenarios most commonly occurring within customers' environments are listed below:

1) Software version upgrade

The development team is continuously working on delivering new functionalities to the software, such as implementing new business metrics, optimizing existing calculations or making improvements to the user interface. The sheer volume of offered features and the amount of different modules and components within the system is large, which makes an efficient upgrade of the core software a logistical and technical challenge on its own. Adding a layer of customer modifications on top of existing functionality can potentially lead to unpredictable results: for example, a certain part of the software may pass all quality control gates of the previous testing stages, but nevertheless fail for a specific customer due to differences in data or environment installation. The goal of UAT in such cases is ensuring that the software can be seamlessly upgraded for each individual customer. This process is different from regression testing, as regression testing just checks whether a certain functionality produces the same result after the software has been updated, while UAT takes the users' view into consideration, ensuring that the upgrade works specifically for them.

2) Defect fixes

When new functionalities and features get implemented, new defects are discovered or introduced. This situation is natural and unavoidable, and even in most idealistic models of software development cycle people involved at each stage of the process are

assumed to make mistakes. These defects can be related to inefficient or broken parts of the code base, hardware failures or matters not directly controlled by the company, like datacenter network connectivity issues. All these cases require certain further changes to be made, which can potentially cause other failures, provoking a malevolent chain reaction. Thus, the UAT is conducted by project teams to ensure that requested fixes work as intended and do not impede customer's daily work.

3) Custom functionality

Utilizing product's high flexibility allows the clients to tailor the software to their specific business needs. This poses a problem encountered both in software upgrade and defect fix scenarios: in many cases it is uncertain how a specific change can affect other parts of customer's SaaS installation. Among most commonly requested custom functionalities are changes to the standard database schemas, including introduction of new data fields, whole data tables and establishing proper correlations between them. The customer project team possesses the knowledge required to implement such changes, which makes UAT primarily their responsibility.

3.3 User acceptance testing workflow

As it was already mentioned previously, any software change for the client like version upgrades or custom modifications is a resource-intensive process often requiring an attention of a full team to be delivered in contractually acceptable way and within established time limits. General workflow for any such change is similar across all customers: first the need for a change emerges, then it is discussed internally and with the client, and based on the outcome of these communications, the change is then outright rejected or implemented, deployed and tested. Each of these steps can have differences and nuances of varying significance, which makes creating a standardized plan a considerably difficult challenge.

The nature of software development industry and SaaS in particular, requires the companies to remain a valid option for potential users by delivering them certain business value. A software application which is competitive on the market should possess high quality standards: be reliable, fast and easy to use and deliver useful and unique functionalities. If the software simply works with no further changes, it becomes quickly out-

dated as new technologies and methodologies of processes develop. Therefore, it is important for each project team to regularly start internal discussions whether the current setup should be upgraded, optimized or modified in other way. Sometimes software change requests come from the customer, which are also initially discussed within the project team. Different origins of such requests create a major problem already at the initial step of the entire implementation process: the vendor and the customer often have conflicting objectives. The software company tries to push the newest version with as little customizations as possible to each customer SaaS environment, whereas the clients want to modify the application for each of their possible business needs, while trying to avoid upgrading the software. Customers try to abstain from any changes and functionalities not directly related to their business needs, as they always pose certain risks and complications for already established and working processes.

Depending on the outcome of these communications, the change is then either rejected or approved. The modification can be too expensive to implement or have too much high technical complexity, which can be valid reasons for both the vendor and the customer to not accept it. When the change is discussed and is confirmed, the implementation phase begins. For performing version upgrades, an internal customer environment deployment tool is used, which simplifies and hastens the entire process while also making it less error-prone by automating the majority of the installation process. However, defect fixes and custom modifications require introducing changes to the application's code or within web user interface, which both are naturally done manually. Generally, this part is the least relevant to the end user experience. Nonetheless, technical issues with the aforementioned deployment tool or complexity of manual changes can potentially delay the release of the software change, indirectly causing a negative impact on the end user experience as they are forced to use the software of a lesser quality for a longer period of time.

All changes to the customer environments are first deployed to special test environments. Each SaaS environment of each client is required to have an almost identical copy, which can be used for extensive testing of any implementations without the fear of disrupting ongoing business processes and end user workflows. The changes are then reviewed by the project team and are either re-implemented if deemed unsatisfactory, or forwarded for the customer's inspection. The main problem in this process is that in the absolute majority of cases, this testing part is carried out manually. The main issue with manually performed operations is the simplicity of omitting certain steps of a

procedure or making mistakes in some processes. Defects that break the software in a visible way and prevent it from operating normally are detected and acted upon immediately. However, due to human errors and sheer volume of data and customer-specific modifications certain issues can slip unnoticed: changes to software components and functionalities rarely utilized by users or data fluctuations. Minor changes to data which are not detected immediately and considered outright defective can exist unseen in customer environments for prolonged periods of time, negatively impacting the customer's business and, as a result, end user experience.

After the change passes the previous testing stage, it is delivered to customer's production environment with real business data and real users. There it undergoes the final round of evaluation, which essentially meant another step of testing. Test instances, while aiming to resemble their respective production environments as closely as possible, still have variances in data and some configurations, potentially rendering previous test results irrelevant as the software can behave in a different and unexpected way. Since this environment is heavily involved in end user's daily work activities, there are certain limitations to the testing capacity, usually leaving time only for checking critical components of the software that can directly prevent the users from accessing it.

3.4 Performance testing practicalities

Performance testing is an important part of the whole acceptance testing process. It is given a special notice in the thesis due to the nature of the application: huge and regular volumes of incoming and outgoing data which is used in numerous calculations and optimizations place high expectations on the software's performance. The data must be processed within contractually agreed time limits and the web interface must be fast and responsive, providing better user experience.

The major difference with the rest of acceptance testing process is that most of performance testing is performed automatically in specialized internal development environment, which are not linked to any specific customer and just provide general results based on abstract data. Therefore, there is usually no need to perform dedicated performance testing in customer environments as performance requirements can be adequately estimated based on results gathered from generic testing routine. However, in

rare cases customers have specific performance requirements, which require extensive testing during UAT phase. Such scenarios usually involve the usage of new technology recently introduced to the software, which does not yet have an established history of usage or known benchmarking results.

Customer's performance needs generally always revolve the same point: end users want the software application they are using on a daily basis to be as fast as possible. This includes the time it takes to load the web page with the application itself, how fast the users actions are reflected in the user interface, how quickly the data is processed, etc. The exact requirements to user capacity, application loading and data processing times can differ between the customers. For example, smaller business do not have strict requirements on how many users the system must be able to support at the same time, as such clients usually do not have enough users for this to ever become an issue. On the other hand, large retailers require a stable and responsive system with near permanent uptime, as even a short-term system downtime resulting from inefficiency of the software can cause unnecessary costs due to business opportunities being lost while the software is slow or unusable.

The main objective of automated functional testing is detecting regressions between different versions of the software, which helps to ensure the expected output of the application always remains the same. Additionally, performance between different builds in several generic scenarios is measured. This is achieved by comparing run times of predefined application routines, such as nightly processing of large volumes of data, calculating order proposals and sales forecasts, transferring resulting data to a monitoring tool, etc. These actions are typical for a customer of any size in any business area addressed by the case company and therefore provide decently accurate performance estimation for each already existing or future potential customer case. If an evaluated duration of a certain routine on a specific day is longer than deemed satisfactory, an automated alert is sent to the quality assurance team, which then investigates the situation and potentially redirects it back to the development team to fix the resulting defect. Such approach helps in achieving consistent performance across different software releases, as most of the performance-related questions the customer might ask during the UAT phase can be addressed immediately without having to specify a time slot for dedicated performance testing for this customer's environments.

However, freshly developed components that have never been utilized by actual end users in production environments do not yet have the history of reliable usage and relevant performance statistics. Moreover, sometimes customers have unique performance requests to already existing functionalities, which have not been covered by automated testing. In this case, performance testing is conducted as part of UAT by the responsible project team.

A good example of when performance of the software is tested as part of general UAT is the most recent customer project conducted by the company. This is a new customer for the vendor company, so the SaaS installation is already assumed to go through extensive acceptance testing as the client has strict requests arising from the nature of their business. Furthermore, this client is piloting the newest feature: an ability to distribute user requests to different calculation nodes to reduce the overall stain on the system. Such change requires a complete rebuild of entire software architecture and existing work practices, and can potentially introduce yet unknown defects. Therefore, related testing is tailored specifically for this customer and for the most part is executed manually. From this customer's perspective, the software should be able to handle several hundreds of users working in the system at the same time. The easiest way to emulate this is activating necessary amount of bot users in the testing environment, which will all perform a certain task at the same time. This requires knowing how exactly the customer is planning to use the system: which functionalities and parts of the software they access on a daily basis or what kind of data they input and expect to see. This information can only come from the customer itself, and in the event of inefficient external and internal communication, the lack of such knowledge serves as a potential blocker for the project: it is impossible to test something if there is no agreed plan of what exactly has to be tested.

3.5 Summary

Software acceptance process is an essential stage of any software development cycle. Good quality software is reliable and easy to use; it creates a positive image of the software vendor among its customers, resulting in a beneficial financial impact for the company and wider business opportunities. Releasing bad quality software should be avoided as, apart from degrading the company's reputation and being an outright immoral act, it is actually more expensive for vendor in long term.

Acceptance testing practices in the case company were described in detail. It was confirmed that multiple challenges to successful user acceptance testing exist. Below is the compilation of the most common obstacles, which the company has to interact with regularly:

1) Communication gaps

Inefficient communication is, unfortunately, the largest and the most commonly encountered problem not only in software testing, but in entire software development cycle. Insufficient customer negotiations or lack of internal interaction between development teams lead to valuable time being wasted on focusing on wrong areas of the project or implementing unnecessary features. As it can be seen from description of case company's testing processes, various communication-related issues can appear in some form at each separate stage of even the most technical routines.

The root cause in each instance of external communication problems is the opposing nature of customer's and software vendor's primary objectives in SaaS business. To put it briefly, the client wants to modify the software as much as they need while avoiding all vendor-pushed changes they deem unnecessary and potentially risky, while the software provider strives to have each customer instance as uniform as possible and run the latest version of the software at any given time. Continuous updates to customer environments are difficult to achieve as they always pose a risk of prolonged software downtime or introducing breaking changes, which the client does not want to ever happen. On the other hand, running the same unchanged version of the software for longer periods of time causes many functional components to become technically outdated, which is an extremely important factor in competitive nature of SaaS market and in case company's business area in particular. The solution to this problem is seeking a compromise variant between these two extremities, which can be achieved only by establishing good relationship with the customer. The client should be informed of the value of software upgrades, while the vendor should also respect customer's business practices, treating them similarly to how a person would treat their friend.

Internal communication problems between implementation and development teams are somewhat similar to customer-related issues, with project teams taking the role of the customers in this scenario. End user needs must be clearly communicated to develop-

ers, eliminating any misunderstandings and double meanings of incoming requests. All previous communication must be kept track of, all testing must be properly planned and logged during execution and all customer or development work must be recorded somewhere. This historical information assists in further tests and serves as a possible reminder for any missed steps in currently performed testing.

2) Technical complexity

Technical complexity of performed acceptance testing is another challenge encountered during acceptance testing of changes in customer SaaS environments. Business model of the software assumes a large degree of flexibility to meet customer's requirements. The amount and different scope of such customizations across different environments naturally increases the difficulty of maintaining the entire eco-system of software components. Great variability of possible software configurations makes testing standardization an extremely challenging task. A possible solution is meticulous refactoring of existing modifications for each customer environment involved, which can be difficult to do due to aforementioned communication issues.

3) Insufficient time

Communication gaps and technical complexities, as well as other unlisted issues, have a number of hidden costs, the most valuable of which being time spent on various stages of the project. Inefficient planning during the testing stage and focusing on the wrong areas of the software can potentially waste time as different components will have to be re-tested. The consequences of incorrectly assigned time greatly range in its severity of effects on the customer's and software vendor's businesses. A slight miscommunication about a feature that was improperly implemented, but resolved within the same testing stage can be considered just a minor annoyance, whereas a critical bug causing major data fluctuations that slipped unnoticed due to insufficient manual testing process can cause huge financial losses for the customer and consequently harm the vendor's reputation. It is important to develop and encourage following acceptance test practices which are standard across the company, so that in a scenario of limited time allocation for the project focusing on correct areas of the software becomes easier.

4 Conclusion

The magnitude of the effect cloud computing has caused on software development industry is enormous. SaaS model is becoming widely adopted as it provides unique benefits for end users, is reliable to use and is cheaper as a long-term solution. Installation of a SaaS environment for a mature company with established delivery cycles requires very little effort. Moreover, SaaS is easily accessible by end users as they can just type the application's address in their browser and start working.

Software development cycle is an intensive and time-consuming process, where testing has an important role. Software that has never been properly tested poses numerous business risks all parties involved. Bad quality software harmfully impacts customer's business processes and damages vendor's reputation as a result, whereas good quality software ensures friendly attitude with the customer, increases the range of available business opportunities and allows managing development and testing resources more efficiently. Acceptance testing is the most important stage of the testing cycle, as it ensures the system's readiness for the end users, who are paying for the software.

The case company utilizes SaaS model for delivering supply chain management solutions to retail customers of varying size. The nature of this business area adds a layer of unique challenges specific to each customer installation of the software. Flexibility of the application which allows practically unlimited modifications, huge amount of existing core software functionalities common for each customer and communication issues between internal teams and with the end users are among the most influential problems that happen during acceptance testing of various software changes.

As stated in the introduction, three main questions that this thesis aims to answer are:

1) What are the business benefits of efficient testing processes?

Theoretical background and empirical evidence presented in the thesis suggest several business benefits of efficient software testing processes. Business reputation and consumer popularity are among the most important aspects of software quality. Product that does not meet its technical specifications or is frustrating and cumbersome to use will naturally gather bad reputation around it, while a good quality product will likely

draw customer attention and open up better and more luctrative business opportunities as a result of growing reputation.

Furthermore, mature and well-developed testing processes lead to more efficient management of available resources. Effective planning affects the quality of the resulting software in many ways. First of all, a solid test plan allows the development team to prioritize technical tasks better and helps to keep a ”time reserve” in case of a troublesome situation. Secondly, effective time management naturally reduces the workload which leads to less technical debt of the company as more things get done in shorter time periods.

The vendor is not the only side benefitting from good quality software. Customers value products which allow them to work quickly and with as little annoyances and distractions as possible. Quality SaaS applications provide high relevancy of data, stability, availability and performance of the software and are often cheaper than inferior quality solutions, providing higher customer value for lesser cost. Appealing to customer needs leads to generating more positive business reputation, increasing the company’s profitability as a result. The main goal of any commercial company is financial success and released software quality directly impacts it.

2) What are the challenges in current acceptance testing processes?

Multiple challenges in the way of improving existing testing processes exist. The most prominent are ineffective communication and various technical complexities, both resulting in insufficient test planning. The main problem with external communication is the fact that the customer and the software vendor are pursuing two opposite goals in their business relationship. The customer wants the software to be as flexible as possible to satisfy their operational needs and prefers to avoid any risks related to application version updates, whereas the vendor strives to have all customer installations as new and uniform as possible. Increasing the software flexibility naturally helps in increasing customer’s appeal, but rapidly introduces technical complexities to the existing infrastructure as each custom change generally requires a custom test plan. Therefore, a compromise between both participating parties must be found.

Communication gaps pose a serious issue for the entire software development cycle, including UAT, where feedback and comments from the users are required the most. A misinformed testing team can focus their attention towards wrong parts of the software

and let defects slip undetected. Numerous software modifications performed differently in each customer's case increase the technical complexity of the system, making manual testing of the majority of components the only viable solution. Manual testing is very error-prone as it is easy to forget or misplace particular steps. Insufficient planning of test scenarios amplifies the severity of other issues as time for different testing stages or other parts of the development cycle is allocated incorrectly.

3) How can current acceptance testing practices be improved?

Solving the problems occurring during acceptance testing is a challenge of its own. As it was established in the previous sections of this thesis, various communication-related issues can happen at each separate step of user acceptance testing process. There is no universal solution for resolving each miscommunication case, but establishing friendly relationships with the customer and within the company helps in addressing each situation in way which is beneficial for both parties. The case company so far has had great success in managing its external communications with end users, efficiently recognizing customers' needs even when the supporting technical description provides insufficient information.

Technical complexities introduced by the flexible nature of the software also cause a major impact on the acceptance testing process. Multiple potential vectors of improvement to the current software change process and acceptance testing itself exist. First of all, the communication issue must be addressed, both internally and externally. The company currently has great success utilizing the flexibility of the application to win the consumer market, but as the business processes mature, more effort should be put into making each SaaS installation as standard as possible. The technical debt is already an existing issue, and adding more customizations to the existing ecosystem without making further improvements to current processes is going to introduce even more complexity.

References

[1] J. Gao, X. Bai, W.-T. Tsai, and T. Uehara, “Saas testing on clouds-issues, challenges and needs,” in *Service Oriented System Engineering (SOSE), 2013 IEEE 7th International Symposium on*, pp. 409-415, IEEE, 2013.

[2] S. Huang, Z. J. Li, Y. Liu, and J. Zhu, “Regression testing as a service,” in *SRII Global Conference (SRII), 2011 Annual*, pp. 315-324, IEEE, 2011

[3] S. B. Vijayanathan Naganathan, “The challenges associated with saas testing,” *Infosys View Point*, 2011.

[4] M. J. Kavis, *Architecting the clod: design decisions for cloud computing service models (SaaS, PaaS and IaaS).* John Wiley & Sons, 2014.

[5] B. Waters, “Software as a service: A look at the customer benefits,” Journal of Digital Asset Management, vol. 1, no. 1, pp. 32–39, 2005.

[6] N. Walkinshaw, Software Quality Assurance: Consistency in the Face of Complexity and Change. Springer, 2017.

[7] A. Spillner, T. Linz, and H. Schaefer, Software testing foundations: a study guide for the certified tester exam. Rocky Nook, Inc., 2014.

[8] A. Benlian, M. Koufaris, and T. Hess, “Service quality in software-as-a-service: Developing the saas-qual measure and examining its role in usage continuance,” Journal of management information systems, vol. 28, no. 3, pp. 85–126, 2011.

[9] S. Dalal, K. Solanki, et al., “Challenges of regression testing: A pragmatic perspective.,” International Journal of Advanced Research in Computer Science, vol. 9, no. 1, 2018.

[10] "Standard Glossary of Terms used in Software Testing, Version 3.2: All Terms" (PDF). ISTQB. [Retrieved October 20, 2019.]

[11] A. Mathrani and S. Mathrani, “Test strategies in distributed software development environments,” Computers in Industry, vol. 64, no. 1, 2013.

[12] G. D. Everett and R. McLeod Jr, Software testing: testing across the entire software development life cycle. John Wiley & Sons, 2007.

[13] N. Vijayanathan and S. Sreesankar, “Overcoming challenges associated with saas testing”,” Infosys View Point, 2012.

[14] N. Walkinshaw, Software Quality Assurance: Consistency in the Face of Complexity and Change. Springer, 2017.