HELSINKI METROPOLIA UNIVERSITY OF APPLIED SCIENCES

Master's Degree in Information Technology

Multimedia Communications

Master's Thesis

IMPROVING CONTACT CENTER PERFORMANCE BY LEAN SOFTWARE DEVELOPMENT PROCESS

Author: Besnik Doroci
Instructor: Ville Jääskeläinen, LichSc (Tech)

14. 05. 2013

Metropolia

PREFACE

At first, it was hard for me to choose a topic for my final thesis. At that time, our company started to apply lean methodologies in software development. This “Lean” subject was new to me, I was lost and at the same time I was concerned for the performance of the product, especially as to how the performance would fit in this new methodology. This subject was introduced at the course for “Research Methodologies”, and I was encouraged by Marjatta Huhta to take this topic for my final thesis.

I would like to thank Ville Jääskeläinen for his effort and time he dedicated to reviewing the paper and helping me to finalize this. Furthermore, I would like to thank Jonita Martelius for her help with the English language.

Secondly, I would like to thank my colleagues at the SAP labs, who helped me with getting material, encouraged me and also participated in the interviews.

Finally, I would like to thank my family for supporting me in general. With their support, I had more time to spend on my studies.

Helsinki, May 14, 2013

Besnik Doroci

Metropolia

ABSTRACT

Name: Besnik Doroci

Title: Improving Contact Center Performance by Lean Software Development Process

Date: May 14, 2013 **Number of pages:** 75

Degree Programme: Master's Degree Programme in Multimedia Communications

Instructor: Ville Jääskeläinen, Head of Programme

Instructor: Imran Razzaq, Senior Developer, SAP Labs Finland

Software development is the process of developing software in an organized way through predefined phases.

Lean methodology can bring flexibility into the software development process and enable customers to add or change the requirements throughout the development cycle.

Contact center solution enables a superior customer experience by allowing companies to ensure the availability of their services and personnel independently of time, location and contact channel. Since in the contact center solution most of events are live events, measuring the performance of it is very critical.

Performance of software is very important when determining the quality of the product. Performance of the system describes how well the system operates under certain workload. The performance testing can reveal how much resources the system uses when loaded with a certain load. Performance needs to be observed from a quantitative perspective and qualitative perspective.

The present study looked for means to how to apply the contact center performance aspects into every cycle of the software development based on Lean methods as references to performance in Lean literature are few.

Qualitative methods were used for the study. The methods used in other aspects of software development were applied to performance testing. Additionally, internal expert interviews were used to collect data of the most relevant practices in the field.

As a result, two approaches are introduced: The first approach is to include performance testing as a part of every development cycle. The second approach is to have a separate team for performance testing. Advantages and disadvantages of these approaches are discussed.

Key words: Lean, Performance, Contact center, Methodologies.

TABLE OF CONTENTS

ABBREVIATIONS AND ACRONYMS

CC	Contact Center
IP	Internet Protocol
LPD	Lean Product Development
XP	Extreme Programing
TDD	Test Driven Development
PAU	Performance After Upgrade
ITU	International Telecommunication Union
IT	Information Technology
VoIP	Voice over Internet Protocol
UI	User Interface
CPU	Central Processor Unit
PSTN	Public Switch Telephony Network
SIP	Session Initial Protocol
QoS	Quality of Service
SSO	Single Sign On

1 INTRODUCTION

The core idea of the Lean methodology is to maximize customer value while minimizing waste, such as producing something that no one wants, making mistakes that create no value, etc. Simply, Lean means how to create value for customers with fewer resources. Lean is applied in every business including product development. Lean is not a tactic or cost reduction program but it is the way of thinking and acting for the entire organization. The idea behind Lean thinking is to let customers have what they want as fast as possible so they do not have time to change their minds.[1]

Software development is the process of developing software in an organized way through phases. Typical software development phases start with software requirement identification, followed by analysis, detailed specification, software design, coding, testing, and finally maintenance of software.

In software development, the way to deliver things rapidly is by delivering them in small packages. Packages can be grown by adding more functionality. The bigger the programs with more features the longer it takes to decide what is needed and get it developed, tested and deployed.

Performance of software is very important when determining the quality of the product. There are usually many issues concerned when discussing performance, e.g. how many users can run the software simultaneously, what delays there are in the software, does the software meet its requirements, etc. Since Lean methodology is based on a limited amount of delivery, having performance figures is crucial at this stage.

Contact center (CC) is a centralized office used for handling telephone calls, e-mail, live chat, video calls, and faxes. CC can be also Internet Protocol (IP) based communications solutions. The operator independent solution replaces the traditional telephone system, seamlessly integrating fixed and mobile communications with IT infrastructure and applications. CC solution enables a superior customer experience by allowing companies to ensure the availability of their services and personnel independently of time, location and contact channel.

Lean methodology has been applied on other fields for a long time but its applications in software engineering are still under construction. There are aspects, such as security issues and performance and scalability, that have not yet been considered as they should have.

1.1 Goals and Research Question

The goal of this thesis is to define the processes that integrate performance processes within the Lean processes during the development cycles. Because Lean methodologies in software engineering are yet in their development stage, this methodology needs research attention. Therefore the study asks:

How should (or: could?) Lean methodologies be implemented for contact center performance processes?

The subject is vital for a company that develops VoIP solutions and is exploring various options for software development.

Lean and performance studies are based on the literature as well as information gathered from a company internally. Internal information is based on the processes used for Lean software development and on the performance of the software in certain business cases within defined limits.

This study is qualitative, based on existing knowledge on Lean methodology, especially in software engineering. Internal expert interviews were used to collect data of the most relevant practices in the field.

The focus of the study revolves around two issues: the performance for contact center solution and its implementation, and performance processes in software development based on the Lean methodology.

Performance is a make-or-break quality for software. Poor performance costs the software industry millions of dollars annually in lost revenue, decreased productivity, increased development and hardware costs, and damaged customer relations. Nearly everyone runs into performance problems at one time or another.

Today’s software development organizations are being asked to do more with less. Operational specifications in web applications often stated in terms on to improve response time or throughput, or both.[8]

In order to reply the research question, it is necessary to study Lean methodologies, how Lean is implemented in software development, software methodologies, key performance measurements for the contact center application, performance processes during software development, and finally include best practices on how to fit performance when applying Lean in software development.

1.2 Outcome

The outcome of this thesis is to define the structure of Lean software development where performance measurement of the software is part of this structure.

Performance processes are defined for the contact center software products developed with Lean methodologies. However, in this thesis there are no details of actual performance figures because that was out of the scope of the study.

Deriving approaches on how to implement performance in software development by Lean solves many concerns about the performance aspects of the solution. These approaches guide how to build a Lean agile team with performance aspects in it making sure that all interactions will work and after each delivery cycle the product has performance built in it.

These approaches can be used in a way that whatever Lean methodology is used as bases for software development the performance aspects of the solution are always taken into consideration from the day one.

People’s opinion about the approaches based on the interviews is that the approaches tend to solve many issues related to the performance of the software e.g. starting with communication issues and going on with performance tasks.

2 LEAN

Lean is the name given to the Toyota's method for producing and developing cars.[2] Toyota Production System in essence shifted the focus of a manufacturing engineer from the individual machines and their utilization to the flow of the product through the total process.

2.1 Lean Thinking

Lean is a method that applies to product development and production[1]. Lean simply means creating value for customers with fewer resources.

Lean thinking is based on components such as goal, foundation, respect for people, continuous improvement, fourteen principles and production development. Lean thinking house is built from the ground (foundation) to the roof (goal), as illustrated in Figure 1.[2]

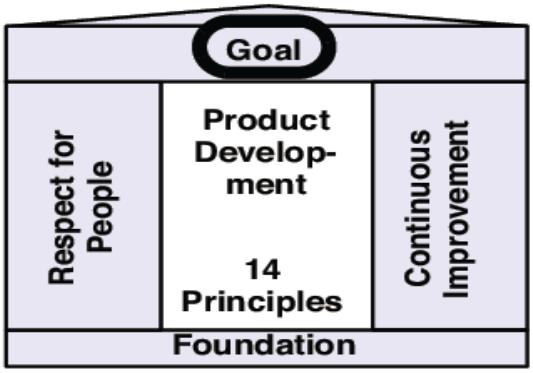

Figure 1. Lean thinking house

In the Lean thinking house it is important that all components work together as a system. Lean thinking house components are explained more in detail in the following chapters.

2.1.1 Goal in Lean Thinking

Two main processes for achieving the goal in Lean are the development by learning from the competition and the production improvement with a focus on short cycles.[2]

The goal of the Lean thinking house is to deliver in a short time value for all processes, focusing on a high quality product. The other goal is to build partnership based on trust.

2.1.2 Foundation in Lean Thinking

The foundation is the base of the Lean thinking house. The motto for the foundation in the Lean thinking house is a good thinking, a good product.[2] People first go through several months of education in order to learn the foundations of Lean thinking, problem solving through hands on improvement experiments, how Lean thinking applies in different domains, and last but not least, the continuous improvement.

2.1.3 Respect for People in Lean Thinking

One of the goals in the Lean thinking house is gaining the respect for people. Respect for people in the way that managers understand and act on the goal of eliminating waste.

Do not trouble your “customer”.[2] Respect for people starts from the customer. Customer means anyone who consumes the work or decisions in question. Not to disturb the customer, the work needs to be analyzed and changed in advance. Customers must not be overloaded by too large releases nor kept waiting for a long time for a new release or a release that has defects that could have been easily tackled.[1]

Develop people and then build the products. Key in this is that managers act as teachers, not directors. People need to be mentored for years in engineering and problem solving, as well in analyzing root causes and making problem visible, by this people learn how to improve.[2]

Teams and individuals develop their own practices and improvements. Management challenges people to change and improve but people learn problem solving and reflections skills and then decide how to improve. Managers understand and act on the goal of eliminating waste and continue improving their work in their own actions and decisions, and employees should see this. Build long relationships with partners based on trust, and help partners to improve and to stay profitable.

2.1.4 Continuous Improvement in Lean Thinking

Go-See is a principle described as critical and fundamental in the Lean thinking house. In the Toyota way Go-See is highlighted as the first factor for success in continuous improvement.[2] An example of Go-See is that managers regularly visit and sit down with the software developers in order to understand the actual problems and see the opportunities to improve.

Another method is Kaizen, which is both a personal mindset and a practice. As a mindset, it suggests “My work is to do my work and to improve my work”, and continuously improve it for its own sake.[2]

Challenge in perfection is the third element of continuous improvement in Lean. In order to improve products and processes, the change in practices is necessary for going towards perfection. Changes in practices can be done by having high expectations, challenging team members, partners, and ourselves to levels of skill and vision beyond the current state. In continuous improvement, there is no final process but rather a continuous improvement and change.

2.1.5 Fourteen Principles in Lean Thinking

Besides the two lean pillars *respect for people* and *continuous improvement* there are other lean principles that form the overall system of Lean. Part of this broader system is covered in the fourteen principles described in the Toyota Way book that comes out of decades of direct observation and interviews with Toyota people. Table 1 explains these 14 principles.[2]

Table 1. The fourteen principles

1. Base management decisions on long term philosophy even if the company expects short term financial goals.
2. Move toward flow; move to smaller batch sizes and cycle times for fast value delivery and expose weakness.
3. Use pull systems by deciding as late as possible. Pull means no storage in inventory until there is a customer order.
4. Level the work. Reduce variability and remove overworking.
5. Cultural building on fixing problems and stopping on time in order to understand the root causes of problem.

6. Master norms to enable kaizen the employee empowerment.
7. Simple visual management for coordination and problem solving.
8. Use only well tested technologies that serves people and processes.
9. Grow leaders who understand the work and teach it to others.
10. Develop people and team who follow your company's philosophy.
11. Respect for partners and help them improve. Bring partners into Lean thinking.
12. Go see for yourself at real work place in order to understand situation and help.
13. Make decision slowly by consensus; toughly consider options for implementing rapidly.
14. Become and sustain a learning organization through relentless reflection and kaizen.

As can be concluded based on the list in Table 1, the organization can become strong if these principles are in mind when planning the future and making decisions.

2.1.6 Product Development in Lean Thinking

Lean product development (LPD) focuses on creating more useful knowledge and learning better than the competition.[2] Creating more value is done by focusing on certain things, e.g. implementing, testing risky features or collecting feedback earlier. It can be very costly to discover during stress performance testing, after 18 months of development, that a key architectural decision had a flaw. In Lean development, short cycles with early feedback loops are critical; by implementing less predictable features early and in short cycles that include testing, the cost of delay is reduced.

Lower cost in LPD is achieved by focusing on large scale to the test automation to learn about defects and product behavior. The cost of frequently executing automated tests is usually insignificant in comparison to the value of early feedback.

2.2 Lean Implementation on Software Development

The goal of Lean implementation is to bring the Lean thinking house into life.[2] That is done by questioning beliefs, attitudes, daily business behaviors and asking what are the internal customer expectations and needs.

In general, people tend to start working on methods, tools, processes and organizational structures. It is relatively easy to see people performing mechanical tasks, this is due to the fact that mechanical work is something people are familiar with. In the other part of the Lean there are behaviors, beliefs and attitudes that relate to the feelings that people have about themselves as formed early in life.

A person with an attitude that serving customers is important is more likely to answer repeated customer requests for information and assistance than someone who does not share the same beliefs and attitudes. This means, that there is a need to start dealing with the behaviors, attitudes and beliefs of a company's personnel in order to make a long lasting change possible and to develop the Lean culture of its own.

In order to make Lean work one needs to understand what "Respect for People" and "Kaizen" mean. This is the basis for a successful implementation and the further development of Lean. To be able to show the behavior that is expected from people in Lean it is important to understand what kind of behavior is required.

Respect for people is described in Chapter 2.1.3. A good start for the Lean implementation for this area would be to question people on experiences when did they get, or did not, respect from managers. Further questions would be that what they thought, felt and did as a result.

Kaizen refers to the continuous improvement of processes in manufacturing. Kaizen implies mastering techniques first before starting the improvement, that is, the improvement cannot happen if baseline is not in place. Kaizen is an ongoing activity and implicates small experiments in order to improve practices. Kaizen refers to the continuous repetition of experiments in order to make a problem visible and find the root cause, so this helps to learn process improvements.

Continuous improvement is described in Chapter 2.1.4. In continuous improvement it is important to find out what is working (e.g. the entire production flow), and then continue on doing that, always by adding small improvements in the direction of the future.

Analysis of the problem can help with technical problems but it rarely helps in finding the solution when problems are about people. People are their own worst critics. Managers are better at improving performance if they emphasize the positive part and let people handle the negative by themselves.

The coaching culture is essential in respecting people and continuous improvement. Coaching is a practical way to implement Lean culture. Telling (demanding) people to change (e.g. their way or working) creates resistance because they assume that they are criticized. To take committed action people need to think things through and choose for themselves how they want to work. In the respect for people it is stated that teams and individuals evolve their own practices and improvements.[2]

Meeting culture suggests that all participants in a meeting are equal and important. Everyone gets turn to speak and is allowed to finish without interruption. Everyone in the meeting is expected to be honest and express his feelings. Shearing good experiences in work from past, present and as well future to further enhance the work.

There are a few details which teams need to fulfill in order to start Lean. The team needs to know what to do, to have clear deadlines, to make sure that information sharing is working, to have a lot of interaction between team members, good collaboration and effective communication. The team needs to have an organized way of working, e.g. have enough information to be able to plan and do their own work, know what has been done and what needs to be done, complementing each other’s work and skills.

Team members are professionals, they should trust each other and that they can solve all problems together. The team understands market and customer needs, participates in the whole process from specification to implementation, and believes that customers like their product. The team needs to have challenging work and freedom to implement their own ideas. The best way to do that is by applying short projects.

2.3 Software Development Methodologies Based on Lean

Software development technology is very demanding starting from the code quality implementation while software is written, and other industry practices for enforcing and monitoring the software without increasing the programmers workload. Lean provides the holistic view with the visibility of the problem and it gives some practices on how to solve it. Lean provides the visibility to the problems, as it focuses in business, value and delivery across the organization.

Software development should be about business with value in it and incremental delivery with value increased and not the software development iteration. Lean provides that vision and then explains what to do with it. Basically it tells how to remove the waste. The waste in the workflow is what hurts. Focus on workflow will help people getting organized and the work done.

In Agile there is a team of a few people for solving a problem which in the end is a local problem. Local thinking does not solve the global problems. Expanding agile in all processes not only in some areas in a company by creating cross teams is difficult as in an enterprise there are different dynamics. In Lean one takes a system thinking approach from end to end in all areas in the company. The entire concept of agile is almost an iteration inside of Lean. Business value is how to get more value quicker, shorten time to market and most important thinks to deliver faster. Lean agile will help on getting fast delivery.

Lean provides a bigger picture view and as well as a method for education. Lean is more of the mindset, respect for the people and their psychology. In software development one needs to understand the big picture. Lean is not interested in who is doing what but more in what should be build, idea to definition, construction validation and deployment. Lean is interested in the work flow in general, having the best team working on tasks and getting work done.

The methodology of Lean software development is the application of Lean manufacturing principles when developing software. A software development lifecycle process or a project management process could be said to be Lean

if it is aligned with the values of the Lean software development and the principles of Lean software development.

Figure 2 illustrates how Lean thinking is the base for most common methodologies and techniques used in software development.[2]

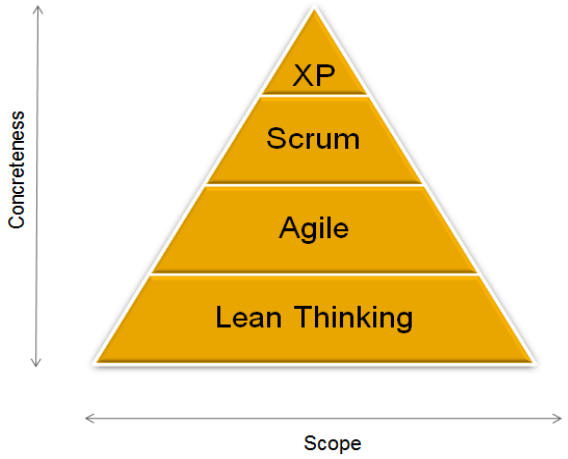

Figure 2. Lean thinking as a base for other methodologies and techniques

Many successful methods used in manufacturing are adopted to the software development. These techniques, such as scrum and extreme programming in the Lean agile, are introduced in the following chapters.

2.3.1 Agile

One of the Lean thinking tools is being agile. Be agile rather than go agile.[2] Term agile was chosen in 2001 at the Utah workshop by a group of modern methodology leaders, two alternative names were considered adaptive or agile. Both names emphasize flexibility.[2]

Agile is a quality of the organization and its people, to be adaptive continuously improving and evolving. Be agile with the goal of competitive business success and rapid delivery of valuable products and knowledge.

Agile means agile, the ability to move quickly, to accept change and to become master of change by being able to change faster than your competition does. This agility is supported by Lean and agile practices.[2]

Agile manifesto values and principles

Agile development is based on four sets of values that support and encourage agility. These values are very important for organizations who want to go agile and scrum. This four agile manifesto values are:[2]

- Individuals and interactions over processes and tools.
- Working software over comprehensive documentation.
- Customer collaboration over contract negotiation.
- Responding to change over following a plan.

Beyond the four agile values are the twelve agile principles that support being agile:[2]

- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customers competitive advantage.
- Deliver working software frequently, from couple of weeks to a couple of months, with a preference to the shorter time scale.
- Business people and developers must work together daily throughout the project.
- The most efficient and effective method of conveying information to and within development teams is face-to-face conversation.
- Working software is the primary measure of progress.
- Agile process promotes sustainable development. The sponsors, developers and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity, the art of maximizing the amount of work not done is essential.

- The best architectures, requirements, and design emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Additionally, there are nine agile management principles that summarize key principles to be agile in a short list:[3]

- Deliver something useful to the client; check what they value.
- Cultivate committed stakeholders.
- Employ a leadership-collaboration style.
- Build competent collaborative teams.
- Enable team decision making.
- Use short time boxed iterations to quickly deliver features.
- Encourage adaptability.
- Champion technical excellence.
- Focus on delivery activities, not process-compliance activities.

Teams in agile should summarize principles mentioned above. A team needs to share an understanding of the principles, and the impact these principles have in daily business, and analyze if the team practices are aligned with these principles.

2.3.2 Scrum

Scrum is one of agile methods which is more related to the faster delivery and higher quality.[2] Scrum as an agile method is used in software application development. It is important to understand these five scrum values: Commitment, focus, openness, respect, and courage.[2]

Commitment. Scrum provides all authority needed for a team to meet their commitments. Scrum is based on self-organized teams that decide which items to pick on from the backlog list provided by product owner. No items

are pushed to the teams and teams are not told on how to do their work. This gives the opportunity to the team to have real commitments, and by this, teams can control how they will do their work. With two or four week iterations they can also be more realistic with their commitments

Focus. In scrum, the teams need to be 100 percent committed to their work, and there should not be multitasking, that is, team members should not be working with items that are not on the team's task list. This means that each person is fully focused with all efforts and skills to do committed work for the goals of getting the good product. This focus leads to the quick delivery and productivity.

Openness. In scrum, the projects are visible to everyone. Daily scrum of each team is an open event where team members share their thoughts about tasks they are working on. Everyone is invited but only scrum team members can discuss while others are listeners. No closed door meetings.

Respect. In scrum, the team's respect for people is very important. All members in the team are equal, they share common goals, and rewards for the achievement goes to the team and not to the individuals. Teams need to understand this, and in order to succeed, the team needs to understand strengths and weaknesses of each team member.

Courage. In scrum, it is important to have courage to commit, act, and be open. Courage helps to follow scrum rules, courage to change the organization and to confront the challenging goal. Teams need courage to explore, learn, decide and act for solving the problems and not wait for others to do that for the team.

2.3.3 Extreme Programming

Extreme programing (XP) is a set of techniques for extending developers' creativity and minimizing the administrative load.[4] XP as form of agile process is defined with four characteristics: communication, simplicity, courage and feedback. Communication refers to the communication between development and customers and to their continuous interaction. Simplicity refers to the simple design and implementation in order to satisfy customer needs. Courage refers to the commitment on functionality delivery

in early stages and often. Feedback is given during the development process on various activities.

XP combines and intensifies proven software development practices with the aim to improve software quality and responsiveness to change customer requirements. Figure 3 illustrates the XP practices.[5]

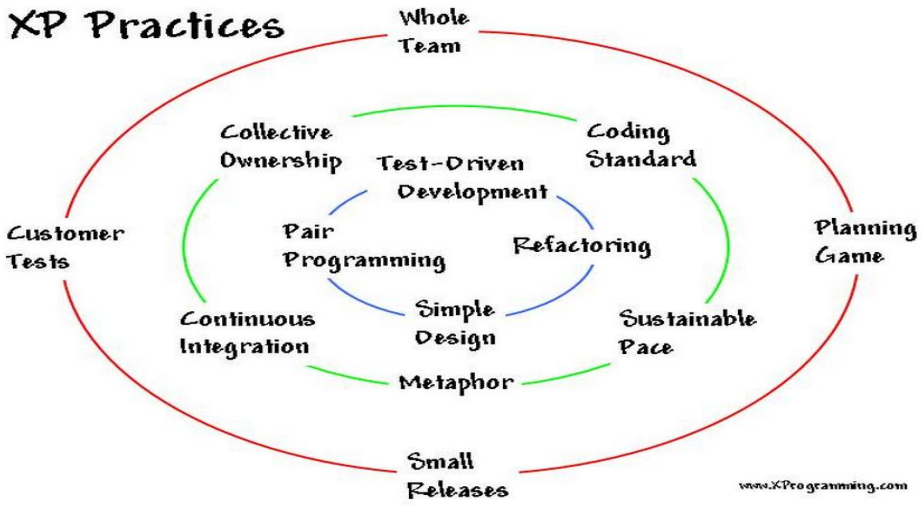

Figure 3. XP Practices

Figure 3 shows that in XP, the team is working as a whole, so that all contributors including customers are in the same team. Release and iteration planning are part of the planning game in XP practices. Small releases are made after each iteration and those releases are tested by the customer.[5] Coding practices are done by having simple design for software. Test Driven Development (TDD) refers to the making of a test case first and then the coding to make the test case work. The TDD ensures a hundred percent code coverage and that all code is unit tested. Pair programming refers to working side by side for having better design and better code. While design evolves the code refactoring must be done for making code cleaner and clearer.

In XP, the system is fully integrated. This integration is achieved by running daily builds in order to verify that system is integrated and there are no issues in the build. Collective code ownership and common code standards are used for simplicity. The team has a common vision on how the program works.

2.4 Traditional versus Lean Agile Software Development

Traditional methodologies are plan-driven in which the work begins with a complete set of requirements, followed by architectural and high level design development, solution building, testing, and finally deployment.

In Agile software development, requirements change during development based on the customer needs, short work iterations and continues deployment. In the following chapters, these methodologies are described in more detail.

2.4.1 Traditional Software Development

The traditional way to build software was through lifecycles known as “the waterfall”. The required steps of the waterfall model for software lifecycle are shown in Figure 4.[4] This way to build software was used both in big and small size companies.

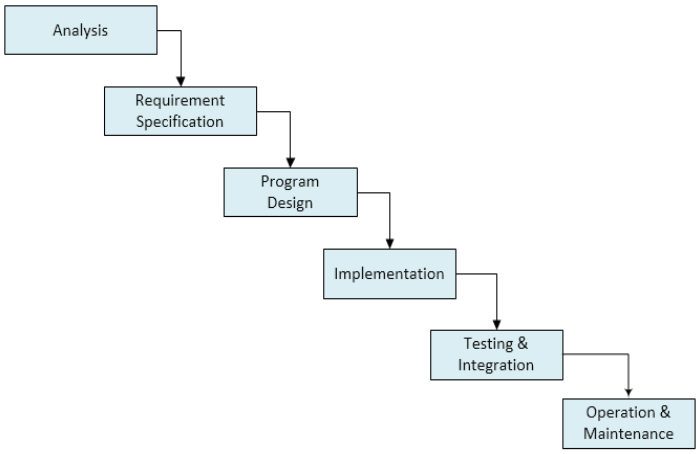

Figure 4. Traditional waterfall model

There are more similar kind of traditional models but all of them start with a detailed planning phase, at this stage the final product is designed and documented in details. So the analysis and requirement specification are done at the planning phase. After the planning phase, the product design phase starts, as well as estimations on how long development will take, and what is needed to execute the design. After design approvals, the actual works starts, and in the implementation phase the code will be written and implemented in the production line. After the implementation is done, the integration and testing phase starts. Testing team will complete testing for product verification validation, and prepares for handover to customers. This

process is monitored all the way to make sure that what is designed is in the product.

This approach requires that product improvement ideas have to come at the beginning of the development cycle and they must be included in the plan. However, the reality is that ideas come throughout the release cycle, and in the waterfall type the ideas are a threat if they come too late to the development. This model is change-resistant and makes creative people lose their passion and skills. Future predictions of people is not possible especially in software development, knowing the exact tasks to be performed in advance for a coming year is far from the reality.

In the traditional waterfall project, risk remains high throughout the majority of the development lifecycle. It may only be at the implementation or integration stage that a problem is exposed and this can have a major impact on all work undertaken thus far. This often causes major delays, or in worst case, to the cancellation of the project.

Even though customers get what they have asked in the first place, after a long development cycle the customer needs may have changed, and the final product may not satisfy the customer. This might make people think that one should plan more, document more, do it differently, and so on. People learn and discover things all the time and all this needs to be part of the development cycle. This leads to the need on fundamental change from the traditional development.

2.4.2 Lean Agile Software Development

Customers' business needs change over time. There may be a newly discovered need, or the need changes due to market. Lean agile methods can bring flexibility in the development process that enables customers to add or change the requirements throughout development cycle.

In Lean Agile development there are deliverables after iterations. Figure 5 illustrates a Lean Agile simplified software development cycle.

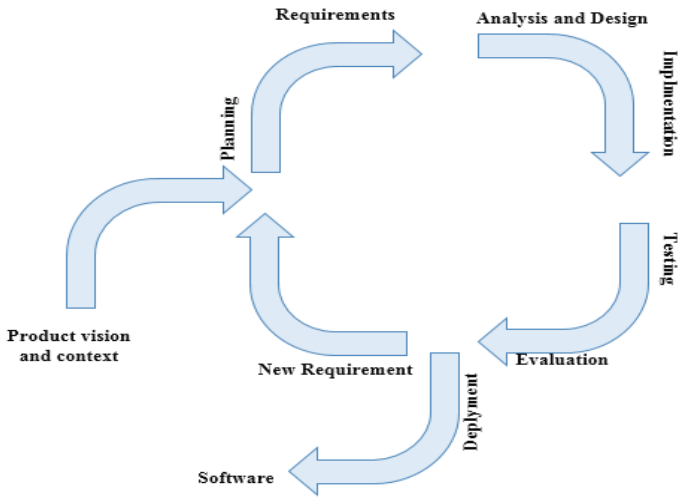

Figure 5. Lean Agile simplified software development cycle

The process starts with product vision and context. Product owners, customer and the development team state assumptions about the key product features, and the value of the product to the company.[6]

At the planning phase, the product features and values are turned into user stories. The user stories should have simple description of a feature or functionality to be added in the real use. These user stories are added to backlog, wherefrom they become to software requirements and will be turned into software.

During the analyzing and design phase, the stories are further discussed and analyzed, and the acceptance criteria of the story are identified. At the design phase there will be more details on how the feature described in the story will be built, including the technical design. User acceptance test cases are created at this stage based on the assumption on how the feature will be used and what are the expected results.

User stories are created throughout the project. Comparing to the requirements in the traditional model, the user stories are not completed until they are taken from backlog and analyzed again by the team before the implementation phase.

Implementing of the user story in general is done by defining a few alternatives for implementation depending on the complexity of story and by estimating the business value of the story.

After implementation the testing phase starts. User acceptance tests are created from the complete user stories before the implementation phase and user story can have one or many acceptance tests in order to ensure that the functionality works. Acceptance testing of a user story verifies that the user story is correctly implemented and the desired customer functionality is in the product. Based on the final acceptance testing, the story can be part of the working software or it will be returned back to the requirement phase. So the evaluation of the user acceptance testing for the stories is taking place and as a final result there is the software deployment or/and new product requirement/s.

Software is developed in stages based on the iterative approach. The core functionality is developed first and additional features added as the product evolves. The early iterations produce only limited working functionality therefore the importance in putting the technical infrastructure in place starts on early phases. These early iterations serve on finding the product risks at the early phases of the product lifecycle, this is a very important benefit aspect.

These software development stages are known in the Lean as iterations. Each iteration has a limited amount of story points to be developed. Target dates for iterations depends from estimated work amount that teams can handle. For example, if a team can deliver 50 story points in two weeks iteration and there are in total 200 story points to be achieved, team will need 4 cycles of 2 week iterations to complete the job. These short iterations enable fast feedback on tasks done and confirmation that customer needs are met. By the end of each iteration the results are analyzed and story points are fulfilled. If they do not require other actions, new stories are created on top of those and they are about to be part of the coming iterations.

In Lean Agile development there are continuous deliveries, which does not mean that what is done is final and sellable. In Lean Agile it is difficult to understand the acceptance part as per user acceptance is one approach, customer acceptance is a different thing, and the budget acceptance is other way of acceptance. In Agile development people are not willing to think in terms of budget, shipment dates and so on but only on story deliveries.

3 PERFORMANCE OF SOFTWARE

Performance of the system describes how well the system operates under a certain work load. The performance testing can reveal how much resources the system uses while loaded with certain load. In addition, other kinds of performance related parameters such as reliability, speed and scalability can be defined with performance testing. Benchmarking performance tests against competition can help in evaluating the salability of the product and needs for performance enhancements.[7]

The system can be loaded with different types of loads depending on the type of the test. Typically these loads are created by feeding the system input with multiple simultaneous events and monitoring the output and performance of the system.

Monitored metrics vary between tested system and test type. Some measurement results can be defined from input and output of the system (black box testing) and in other measurements information on transactions and events inside the system is needed.

3.1 Quality Assurance Levels in Software Development

Performance of the system can be measured on all of the following quality assurance levels in software development.

Unit testing. Unit testing is the first level testing, it is a process for testing the individual subprograms, subroutines, classes or procedures in a program.[8] Unit testing is important part for developers toolbox in order to achieve reliable application. It is very important to find defects of the developed code in application at the early stage, finding them later may be costly and challenging. Unit testing is performed usually by developers.

Module testing. Module testing is very similar to unit testing but in module testing the module is tested as a whole. Modules have internal architecture and external interfaces.[8] Module testing helps on verifying changes in code dependencies (common code, compilers). Some issues are hard to find in integration testing so module testing helps in finding those unwanted issues.

Integration testing. After the individual module tests have been done the next step is to integrate modules together.[4] Integration testing is testing two or more modules together without the need of all modules at the same time. System architecture will be tested in more detail at this stage.

System testing. System testing is the first level where the system is tested as a whole.[4] The system is tested to verify if it meets the functional and technical requirements and also enables us to test, verify and validate the software requirements. The system is tested in an environment that closely resembles the production environment where the application will be finally deployed. Possible external systems (e.g. audio gateways) need to be part of the environment during testing. System testing is covered with functional and non-functional testing.

Functional system testing. The goal of functional testing is to verify implemented features according to the Software Requirements Specification.[4] The testing is done by executing test cases which have been created to cover all Software Requirements.

Non-functional system testing. This is the testing of everything that does not relate to the functionality of the system.[4] This type of testing covers aspects such as ease of use and performance. The system may provide all the necessary functionality but if it is not easy to use or does not perform very well, it will not fit for its purpose.

Some non-functional testing types are: User interface testing, Browser testing, Documentation testing, Infrastructure Compatibility List (ICL), Security testing, Technical language testing, Accessibility, Recovery testing, Usability testing, Performance, Load and stress testing.

Non-functional system testing is more concerned with how well a system performs its function. All systems are written with a purpose in mind usually with the intention of making money and being profitable. The non-functional attributes give the system a competitive edge over competitors.

Acceptance testing. Acceptance testing needs to validate that all defined customer requirements are in the final product.[4] The acceptance testing checks individual functions and applications and also the integration of functions working along product. The robustness and correct functioning of

the system are tested against specifications in the test cases as well as against the expectations of the customer and expert testers.

3.2 Software Performance Testing Types

Performance is an indicator of how well a software system or component meets its requirements. Software performance testing is typically divided into testing types based on the performance output. Common testing types are described below.

3.2.1 Load Scaling, Stress and Capacity

In load scaling one tests the performance of the system and its components are monitored while the level of load is increased. With the information from load capacity testing scalability testing gives a good estimation on how the system behaves while the load level increases.

Load capacity testing contains tests for defining the maximum quantities for different types of system input events that system can handle without failing. Also failing mechanisms should be investigated when the system is stressed to breaking point.

In load capacity testing the level of load is increased between test runs until the breaking point of the system component is found. The breaking point is considered to be a point where the system does not work properly or does not fulfill all requirements defined for its acceptable performance.

The purpose of load capacity testing is to test how much load essential system components can handle. The testing of each component will be performed by loading the complete system with set of common system input events while monitoring the performance of tested component.

Load capacity testing contains both burst and steady load tests. The goal is to define maximum load levels for each tested component / complete system in both cases. For some components maximum levels cannot be achieved due the restrictions of other system components.

Load scaling tests are done with a standard system setup. In some special cases some specific component may be isolated to run on their own server.

Scalability and load capacity testing is done with both burst and steady loads**.**

3.2.2 Stability and Memory Consumption.

Stability and memory consumption tests contain tests running for a longer period. Possible memory leaks and other issues related to repetitive events are checked.

Stability testing refers to the overall reliability, robustness, functional and availability of a system under a variety conditions. Acceptance levels for stability of the system should be determined prior to the production release.

3.2.3 Scalability of System

Resources/Hardware scaling testing contains tests where the performance of the system and its components are monitored while system resources are scaled up by adding new instances of system components to run on additional server.

The purpose of the scalability of the software is to investigate the behavior of system with simultaneous events of different types, how system / component performance increases while resources increase and as well verifying of the performance against requirements.

With resource scaling tests it can be shown that the system is able to handle up scaled levels of load in up scaled environment. Resources scaling tests are done by adding resources to standard system setup by e.g. adding more core components in the application server. Scalability testing is done with both burst and steady loads.

3.2.4 Flexibility and Failure Recovery

Flexibility and failure recovery refers to the system maintenance. Flexibility and failure recovery tests test the operation of the system in certain failure situations and monitor how the system recovers from that. In system requirements it has been stated that specified system maintenance operations shall be able to be performed while the system stays active.

The correct operation of the system will be tested while functionality is added /modified/removed from the system while the system is in use (active system

simulation). Hardware is added to the system while the system is in use (active system simulation).

Operation of the system shall be tested in different types of failure scenarios e.g. automatic switchover to a redundant server without losing calls. Depending on the failure scenario, system core functionalities (e.g. calls) should not be affected or system should recover in acceptable time.

In the failure recovery tests the failure mechanisms and breaking point analysis should be performed during load capacity tests, network problem recovery tests and module failure recovery tests.

Flexibility and failure recovery tests mainly contain steady load tests with some special events (e.g. 1000 active calls + maintenance operation/failure occurs in some system component).

3.2.5 Performance after Upgrade

The goal of performance after upgrade (PAU) testing is to measure the performance of the system after upgrade with some standard tests and compare the results with ones from earlier tests with previous software version.

Changes in performance are monitored and detailed testing is performed if notable changes are discovered. PAU testing does not include tests with very high levels of load. The purpose is to test the system at the normal level of load.

Performance after upgrade testing mainly consists on steady load tests as the results between tests runs are more easily comparable. However, also burst tests may be executed in order to get more information on performance of upgraded software.

Standard system setup is used in performance after upgrade tests. In some special cases performance change of specific component can be investigated in detail by isolating it from standard system setup to its own server

3.3 Software Performance Testing Cycles During Development

All performance testing types mentioned in Chapter 3.4 need to be repeated in particular testing cycles. Performance testing is a cycle repetition of defined steps. Figure 6 illustrates the main events of the performance testing cycle.[9]

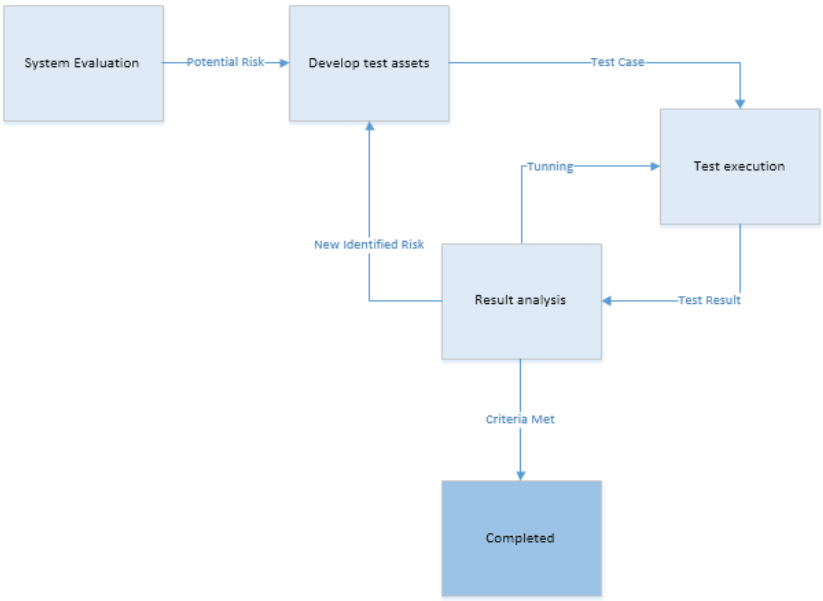

Figure 6. Performance testing simplified cycle

As illustrated in Figure 6, performance testing starts with system evaluation. Collection of information about the project as a whole, the functions of the system, the expected user activities, the system architecture, performance requirements and any other details that are helpful in creating the performance testing strategy specific to the needs of the particular project.[9] Outputs for the system evaluation are risks as they are potential problems.

Performance testing is based on identification of risks, e.g. the program being too slow, memory leaks, cannot handle particular amount of simultaneous users, failovers and recoveries and so on. Risks can be of various types on software projects but as for the performance testing cycle commonly in question are technical risks. Technical risks such as platforms, methods, processes, standards and functionality may result from defined parameters, other dependencies, lack of experience, tools in use. So the list of identified risks needs to be analyzed and prioritized based on the potential risk impact to the solution. In the risk analyzing phase the estimation needs

to be done on the probability that risk may result in a loss, in impact of that loss if risk turns into problem and the time when risk needs to be addressed.

At the develop tests assets phase the test running environment and test cases need to be prepared for test execution and analysis. The test running environment is achieved by experimental design. Experimental design is done based on the identified risk. In experimental design it is important to make sure that performance testing can be run due to the change on the software. There may be one or more factors that change during a performance test. Therefore, it is good to explore more with these factor changes in order to understand what impact each change may have to the performance of the software.

Test cases are created based on design, of course tools and skills are needed in order to have probability to catch the errors. When creating a test case, the idea on how the software may fail, and if does so, how to catch it, needs to be used. Test case should be neither too simple nor too complex. Sometimes it is necessary to combine test cases into one big test case e.g. for testing software capabilities. This big test needs to be created in such a way that it is not too complicated to execute nor to understand the output. During test case creation the expected output or test result needs to be noted down as reference for executing the test case.

After test case creation the test execution will follow. It is very essential that the software is tested in appropriate way. Most important rule is that test case procedure will force the software to use data entered correctly. Right after the execution of the test case the test case validation takes place, and if necessary, debugging is done. During this phase the initial baselines are taken for future testing. The results from these first tests are used for benchmarking purpose in order to help in result analyzing for all following tests to be analyzed in the result and analyzing phase. For the benchmarking, it is important to identify the business, system and user requirements that define it, and system usage and key metrics for measuring it.

In order to verify that requirements of the function or performance tests are met, the results need to be analyzed. Depending on the test result, the testers can be satisfied or dissatisfied to the outcome. When satisfied, the

outcome means that requirements are validated and therefore criteria are met. Testing is completed and test results are printed in the final report. When dissatisfied, the outcome means that there is a bug in design, test case, or in execution part, or the issue can be in analyzing tools, or there is a new risk to be identified. The performance team should be able to detect the source of issues, and start tuning in order to meet the acceptance criteria.

By tuning activities the system is modified. The modification made must improve performance of the system, if the modification does not fix, or improve the performance, then the modification is removed from the system. In case there is a new risk identified, the new risk will get back to the test assets.

Test analysis report is necessary for documenting the test result, in case of failure it provides the information on locating the source problem, repeating the problem, as well as fixing the source of problem.

Performance testing is a repeated cycle which gets complicated by unknowns and estimations. Performance testing is based on lots of variables. Before a system goes live it can be only guessed how the users will use the software. Test runs are based on guesses and approximations (e.g. in what hardware or cloud the software will be run etc).

4 CONTACT CENTER SOLUTION

Contact center (CC) is a place where various types of customer contacts can be handled for various customer needs. Contact center is defined as “A coordinated system of people, processes, technologies and strategies that provide access to the organizational resources through appropriate channels of communication to enable interactions that create value to the customer and organization”.[10]

Contact centers are guided by three level of strategies: Business, contact center, and technology.[11] Based on these strategies the CC can have different operation perspectives based on the industry, sector, size, location, channels, tasks, volume, and function.

From the CC operation perspective and user needs, the operation classification is demanded firstly based on the industry (e.g. Media, Hospital, Hotels, Banks), then on communication sector (e.g. business to business or business to customer), then by contact center types (e.g. inbound, outbound or both types), serving location (e.g. on-site or remotely), serving hours, business tasks (e.g. reservation, orders, billing), call volumes (e.g. number of serving agents per day, calls handled per day) and finally from the communication channel point of view. In today’s CC solution there are supported various types of communication channels such as voice (telephony), e-mail, chat, SMS and fax. CC service requires highly skilled agents for serving and handling customer needs.

Figure 7 shows a simplified contact center solution with very basic features. When drawing Figure 7 all extra features such as e-mail, chat and fax were ignored. Figure 7 illustrates the basic telephony services and features which are similar to the contact center solution developed in the customer our company.

Lean methodologies applied to the performance of the contact center solution related to this thesis will be concentrated on only in the telephony part.

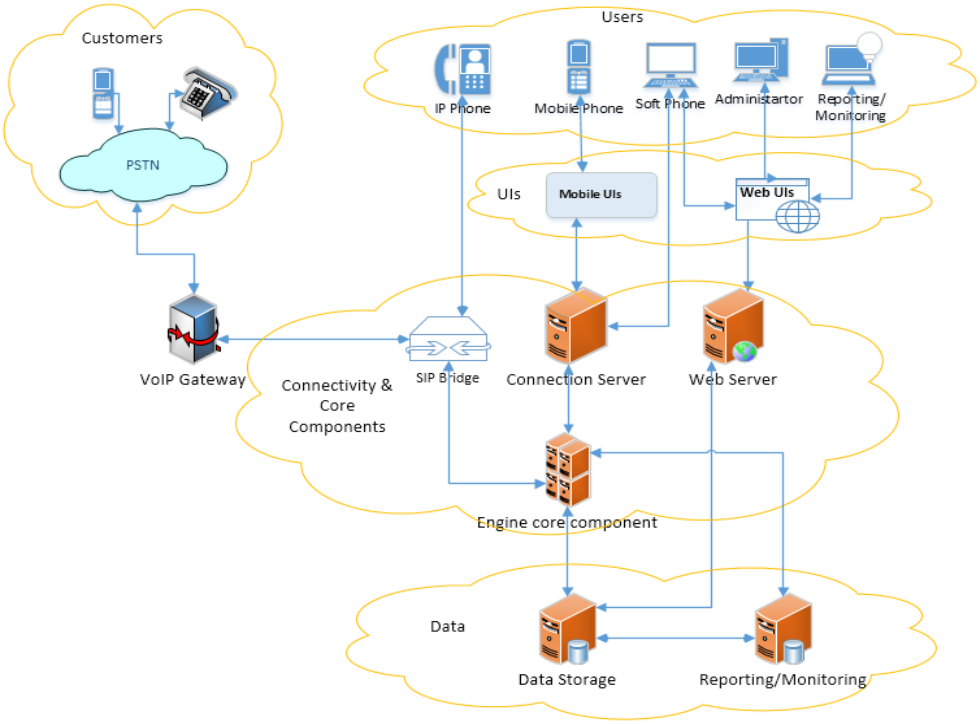

Figure 7. Simplified picture of contact center solution inter connections

The telephony features of the contact center solution are fully based on voice over internet Protocol (VoIP) technology. The system is easy to configure according to communication business needs.

A contact center agent can login to the system from different terminals such as Web User Interface (Web UIs) or IP desk phone. Contact center agents are internal users of the system having login accounts created by the administrators of the system. Contact center agents handle incoming or outgoing contacts.[10]

Communication from the customers to the system goes through VoIP gateways. VoIP gateways converts VoIP stream from data packet switch to a circuit switching format and vice versa.[10] Circuit switching refers to the Public Switching Telephone Network (PSTN) developed to support voice traffic.[11]

Web services and VoIP gateways (such as SIP or H323) are used as UIs for communication with the engine components that handle connections, call routings, call switching and other functionalities. Different terminals offer different functionalities and thus load of the system is changing due to the connection type.

Figure 8 illustrates the simplified picture of a basic contact center.

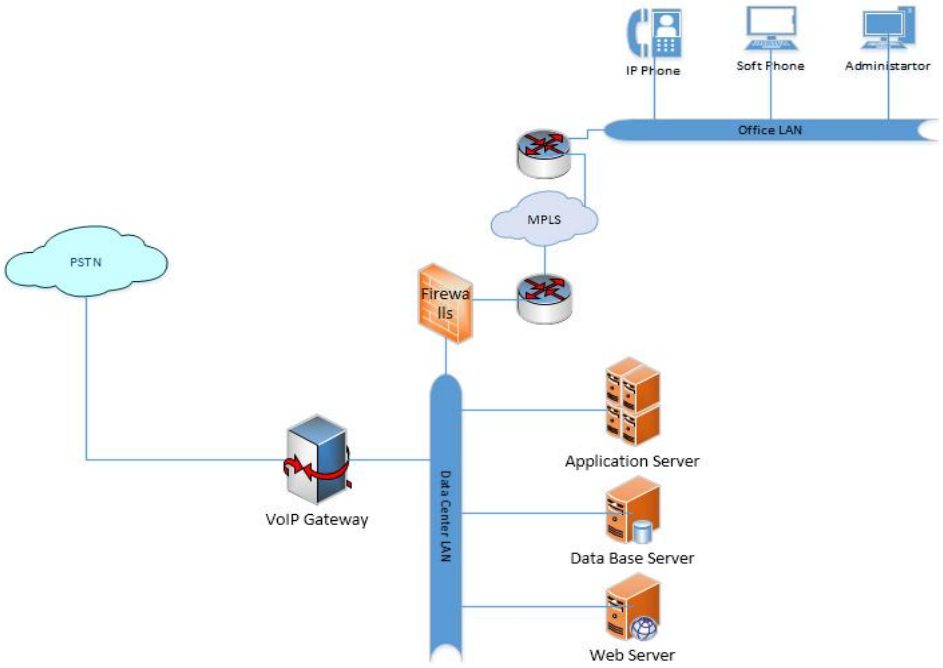

Figure 8. Simplified picture of a basic contact center

Figure 8 shows how data center and office are connected through internet routers and firewalls while customers and data center are connected through PSTN and VoIP gateways.

4.1 Human Interface Modules

Human interface modules are mostly web browser based user interfaces that provide the end users access to the phone, monitoring, reporting as well as to the administrating functions.

Soft phone is the primary graphical user interface. Soft phone has the basic telephony UI elements, directory, call history functionality and other functions such as presence management, serving state and so on. Soft phones require PC with fast enough CPU, adequate memory, high quality sound card and supported operating system.[11]

Mobile Client is a UI for mobile phones based on mobile platform.

Session Initial Protocol (SIP) Phones are IP desk phones that are connected through network cable to the system.[11]

Monitoring is the graphical UI for monitoring the users and providing current statistic information. Monitoring includes data tracking and analyzing for identifying individual agents, contact center performance, anticipating problems and coaching.[10] In performance measurements we can use this online monitoring feature for monitoring the quality of contact center service.

Reporting is the graphical UI for getting statistical results on software usage and reporting information.[11] In order to analyze the events on contact center solution performance, the reporting tools can be used to get this data from the reporting data storage.

Administrator is using java web star UI for configuring, controlling and monitoring the contact center software.

Customers are the external users who perform the calls to the system. They can use any phone connected to the PSTN directly or indirectly to make a call.

PSTN networks are connected to the IP network through VoIP gateways. Gateways provide the electrical and mechanical adaptation between two networks and perform protocol and voice stream transactions. Gateways are controlled by SIP adapters towards contact center solution which are connected to the core components for call routing and switching call events to the correct extension (Agent or queue). After connection is established, the VoIP gateway routes RTP stream to defined extension.

4.2 Connectivity and Core Components

Connectivity and core components handle connections, call routings, call switching and other functionalities (e.g. data loading, protocol transactions etc.). These components are described below.

Web Servers. Web servers are computers that deliver web pages. Most contact center applications use the client server architecture.[11] Web UIs are using web service components of web server for loading data in UI from requested data storage. This data (user setting, user role, user serving, user profiles directory data) will load the web server and data storage server.

Connection servers. Connection servers provide secure connection between web client UI and core component.[11]

SIP bridges and gateways. Gateways connect PSTN networks to IP networks and enable calls to pass from one to the other.[11] SIP bridge handles initiating, receiving and controlling calls from/to SIP gateways and SIP phones. SIP bridge contains a SIP register and perform protocol transactions from SIP gateways to the core component of the system. In case of SIP phone login to the system SIP bridge handles the transfers of authorization info to the core components of the software. Voice stream is travelling through SIP bridges and gateways.

Because of the real-time requirement of voice streams it is essential to understand the possible voice stream paths and to recognize the possible endpoints. Gateways are voice stream endpoints in the LAN. The other voice stream endpoint is either an immediate endpoint such as IP Phone, Client Terminal, Voice Mail, Interaction voice responds (IVR) or in some cases another gateway.

Core components. This type of infrastructure provides highly integrated capabilities for basic voice switch functions as well as optional sets of contact center capabilities.[11] Core components are used for handling contact request, event auditing, call routing, call switch and control.

Figure 9 illustrates the simple call routing example.

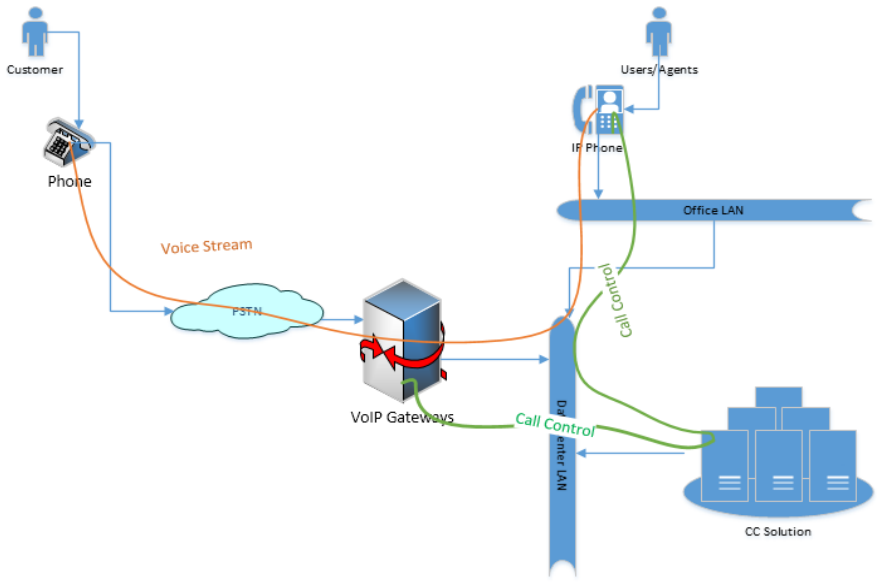

Figure 9.Simple call routing including voice stream and call control

Figure 9 shows that while the call control is handled by the application server core components, the voice stream is having a direct connection from the gateway to the end point.

4.3 Data Storage

Data storage server refers to the file and database server. Data storage servers are used for storing system configuration, user accounts and call details as well as files for voice prompts, voicemail messages, call recording and attachments.

This data can be static or dynamic. Static data refers to system availability and therefore it is very crucial for operation of the software. Dynamic data is not crucial for the operation of the software but is very important for service control, production control and other related issues important to the customer.

Reporting and monitoring databases are as well part of data storage. Quality of service can be monitored by the system, data of events can be analyzed. Event data is stored in reporting databases. All kind of reports based on users, dates, and queues can be received form reporting database.

4.4 Hardware

Hardware needed for the contact center solution depends on the type, scale and size of the implemented solution. Minimum hardware requirements for system installation are one application server and one database server. However, the recommended number of servers for minimum installation is two application servers and clustered database. The recommendation comes from the fact that application components can be backed up in these two application servers in redundant installation where one component instance is active and other one as backup in case the active component fails.

Telephony features are fully based in VoIP. But in order to communicate with the outside world then VoIP gateway hardware is needed. From the client side perspective the agent needs a PC with headset, mobile phone or IP desk phone for communication.

4.5 Network

Network infrastructure is the technology that physically transports the agent/caller interaction.[11] Telephony network refers to the PSTN and mobile network. PSTN connection to the TCP/IP Ethernet network is done with VoIP gateways which transform PSTN call to IP and vice-versa. Mobiles are connected through own network (GSM/3G) to the PSTN and further to the VoIP gateways.

Network is connected to PSTN via E1/T1/J1 connectors and to the CC software via VoIP gateways. Internet connectivity for the web forms is done through firewalls to the LAN.

Network connectivity for the solution users/terminals is done through corporate data network. Network is connected to the solution via TCP/IP Ethernet network (LAN/WAN).

Contact center enables agents to handle phone calls, chats, e-mails and other interactions with customers. In order to enable this communication, all components mentioned above affect on contact center performance. In the following chapter, the performance of the contact center solution will be in focus.

5 PERFORMANCE OF CONTACT CENTER SOLUTION

In the chapters below the performance of the contact center is introduced in general, and the performance requirements and performance analysis of the contact center is described.

5.1 Performance of Contact Center Introduction

The most challenging issue in the contact center performance is the timing issue e.g. answering calls in time, service availability, resource allocations, failure recoveries, and so on.

The contact center performs in real time and there are changes e.g. in the number of agent serving, how agents are allocated, and how good they are performing. Contact center needs to monitor the number of calls arrived to the extensions (queues) in order to avoid long queuing times.

From the contact center performance perspective it is very important to answer to each call arrived to the system. CC supervisor can supervise on how agents are performing and in what state they are, are they serving or are they wrapping up calls, or are they just not currently serving in the queues. Supervisor can inform agents for the need to serve, or even force agents to serve in queues.

5.2 Performance Requirements for Contact Center Solution

Performance requirements for the contact center solution described in this chapter are derived from the end user, application, backend, network, OS, hardware, and software.

5.2.1 End User Requirements

Performance requirements from the end user perspective start from login, user authentication and other requirements relevant for the end user. These end user requirements are described below.

Login. Login is the process for identifying and authenticating the user through credentials for accessing system. Login starts with connection to the system, as an output, the login session is opened for entering credentials. Credential data is sent for the authentication to the database and if the user

is successfully authenticated the session is opened. Login process takes time and may cause delays. This delay needs to be reasonable.

Authentication. Authentication to the system can be done in various methods. Common methods used are login name following with the password, by certificates (e.g. SSO single sign on), and pin codes for phone terminals.

When authenticating with login name and password the client sends user name and password. Strings are processed and passed to the database procedure for verification. Server responds with success or failure depending on result. On success server sends to client the unique user id that will be used with service requests. On failure the request for credentials is sent over again until success or timeout.

Authenticating users with certificates, user specific certificates are required on the user interface module. Pin codes are used for authenticating users using SIP phones.

Web Browser based user interface. User interfaces that provide end users access to the phone, monitoring, reporting and administrating functions.

Throughput. Throughput represents the amount of data that the users received from the server at any given second. This statistic helps to evaluate the amount of load end users generate, in terms of server throughput.

Transaction per second. The number of completed transactions (both successful and unsuccessful) performed during a test. This statistic helps to determine the actual transaction load on system.

Transactions Response Times. Slow applications cause unhappy customers.[8] The average time taken to perform transactions. This statistic helps to determine whether the performance of the server is within acceptable minimum and maximum transaction performance time ranges defined for your system.

Hits per Second. The number of hits made on the Web server by end users. This statistics helps to evaluate the amount of load end users generate, in terms of the number of hits.

Call Connection Response Time. Response time in terms of delays in connection of two ends and the response time in terms of delays in voice stream. Delays in connecting two ends refers to the time that it takes for placed call to be received by the other end (e.g. customer placing the call and waiting for the agent to answer it). This type of delay is not that critical compering to the delays in the voice stream with demands to carry high quality voice over it.

Voice quality. Voice quality in terms of end user experience can be for example on the following levels:

Excellent voice quality. Speech is clear. Volume levels are satisfying and there is possibility to increase/decrease levels from the current level. Voice dynamics are good or excellent.

Good voice quality. Speech is clear. Possibly not the very best voice dynamics, appearance of distant counterpart and/or desire for volume level adjustments.

Acceptable voice quality. All speech is caught, but there are occasionally some minor but clear distortions and/or too low or high volume levels. It might be approved if there are other good reasons to approve it.

Unacceptable voice quality. Most or all of speech is caught, but there are severe distortions and/or too low or high volume levels. This will not be approved.

Availability. General availability for services in the solution in case of service failure and in case of service maintenance. Service availability to the end user must be maintained and managed in order to be available all the time.

In case of application failure in the system, the redundant backup unit of the failed application is activated and thus availability of the system is recovered. In case of maintenance system is switched to the redundant backup units and active units are switched off and are maintained during this time, finally the system maintenance is done and active units are switched on again. From the end user perspective system needs to be available all the time, and switchovers should have minimum impact to the end user.

5.2.2 Application Performance Requirements

Application performance requirements are related to the call events, automated services and routing rules. These application requirements are presented below.

Inbound events. Inbound events refer to the calls that are coming to the system from the external network (PSTN). These incoming events need to be handled by the system without errors or extended delays.

The call routing events in the inbound need to be performed according to the routing rules. Live events need to be handled by the system as designed. These events (e.g. phone calls) are supposed to happen simultaneously during inbound so that in the system there will be a need to handle this events properly. One important aspect in the live events is that they must continue to be active until they are hanged up by the end user or application definition. With increase of events the load of the system is increased and the application performance needs to be verified.

Outbound events. Outbound events refer to the calls that are going from the system to the external network (PSTN). These outgoing events need to be handled by the system without errors or any extended delays. The call routing events in the outbound need to be performed according to the routing rules. Live events need to be handled by the system as designed, correspondingly to the incoming calls.

Automated services. Interactive Voice Response (IVR) applications allow the system to interact with humans using voice and DTMF keypad inputs.[10]

IVR application can be used for example for informing the caller where call is directed. IVR application selects the destination based on the defined number.

IVR applications contain logical functions and states. Each state have specific IVR functionality e.g. playing message from file, recording message to a file, divert the current call to another number etc. IVR application loads the system with events and this IVR application performance needs to be verified.

Routing rules. Calls that have been called to a queue number are delivered to the agents according to the queue settings and schedules. Routings to the queue number can be done based on the automatic call distribution, conditional routing and customer data routing.

Automatic call distribution routs calls to queue based on the destination number. Conditional routing analyses the current situation when making routing decisions and dynamically adapts to the situation. The routing conditionals can be for example purpose of call, time of call, and CC staff availability. Customer data routing requires access to the customer information and uses customer information when making routing decisions.

Calls or forwarded calls to a personal number are treated primarily as the personal reachability service profile or device status defines. Based on the routing parameters and reachability profiles, a call that comes into a queue are forwarded to a personal number.

In case of presence profiles the users are ready and available to communicate. In case of absences e.g. meeting, lunch or vacation, the calls are forwarded to other users, queues, voicemails or a simple IVR informs the caller about the current reachability state based on the routing rules defined.

As per device status system monitors the state of user terminal device and routings and reachability are depending on the device state (e.g. ready connecting, busy).

Routing of the outgoing calls to an appropriate number or gateway. Outgoing calls routing can be based on the call with exceptions, location based routing and least cost routing functionality. Ongoing calls with exception are referring to the redirecting of called number A to the number B. Location based routing select the gateways for routes based on the source number location. This is used when company has offices and gateways in different location. List cost routing is used for reducing carrier charges when call reaches the PSTN.

5.2.3 Backend Requirements

The backend requirements are related to server-to-server connection and high availability. They are presented below.

Server to Server. Servers used for the solution are application and database servers. Application servers with performance focusing on processor and networking. Application servers work with processes between users, modules and data bases by sending retrieving events, storing, changing or removing data. Application servers do not contain any crucial data and therefore they are easy to be replaced.

Database server with performance importance on amount of I/O file and storage capacities. Data base servers contain static data and dynamic data. Static data refers to the configuration and user data which is critical for software operation and therefore availability is a must. Dynamic data refers to the call data, service level control and other business relevant data which is not crucial for the ongoing operations but very critical for the service provider or to the customers or both. Data need to be regularly backed up.

High availability. Software processes in servers are controlled and monitored by availability controllers. In case a component fails the automatic recovery starts by restarting the same component on another server for having the service back.

5.2.4 Network Requirements

The network requirements are related to network traffic and VoIP packages in the network. They are presented below.

Network traffic. Network traffic control is implemented with firewalls and access lists on switches and routers and using VLANs. Different traffic shapes apply to different network boundaries.

Between the management network and the core network, there is management traffic, such as configuration tasks. Between the core network and the access network there are mainly database queries.

Voice streams are typically carried from voice endpoints in the access network over service and access links to voice endpoints (e.g. IP desk phones, VoIP gateways) in the customer LAN.

Voice streams have strict time demands in the network, thus it is recommended to minimize the network distance between endpoints. Minimizing the network distance between endpoints may decrease the

number of services that depend on any particular device and can therefore provide more robustness against voice distortions due to increasing traffic delays.

VoIP packages. In VoIP, the quality of voice transmission and receiving must be consistent in order to replace public switch telephony network with the service of VoIP. VoIP packages should not be suffering from too long one-way delay known as latency, packet loss and jitter.

One-way delay demand should be less than 150 milliseconds in order to carry high quality voice, according to the International Telecommunication Union (ITU).[11] Latency over 300 milliseconds leads to the big delays and echo and is unacceptable due to the bad voice quality.

There should be no packet loss in the network. Packet loss exceeding 0,2 percent causes clipping.[11] This means that words get dropped from conversation. Jitter buffers for compensating varying delays are effective on varying delays less than 100 milliseconds.[12]

VoIP can guarantee high-quality voice transmission only if the voice packets have priority over other kinds of network traffic. VoIP traffic must have certain compensating bandwidth, latency, and jitter requirements. This high quality voice is guaranteed if the Quality of Service (QoS) is applied in the network.[12]

5.2.5 OS/Hardware/Software Requirements

Requirements related to CPU, memory, disc usage, database and processes data are presented below.

CPU. Some server applications cause more strain on the CPU capacity than disk I/O or memory. The acceptable amount of CPU to be used by module tasks is e.g. 40%. For CPU optimal performance at peak times, the average CPU usage must not exceed e.g. 70 percent over a 15 minute interval. Any back end module tasks should not take total amount of CPU.

CPU utilization is referring to the total amount of available CPU that is being utilized. CPU utilization should not exceed e.g. 60 % of total CPU. CPU Burst tolerance referring to the amount of time a process uses the CPU for a

single time job. A process can use the CPU several times before completing the job.

CPU spike is a sudden increase in processor utilization, which can cause temporary or permanent damage to the CPU and motherboard. CPU spikes can be caused by simultaneous running of applications that use a large amount of CPU resources and RAM.

A sustained processor queue length of two or more threads indicating that threads are being kept waiting because a processor cannot handle the load assigned, these typically indicates a processor bottleneck.

Memory. For a process, memory consumption should be steadily growing. After load is over, the memory consumption should return to reasonable state. Memory consumption should be stable.

No memory leaks. A permanent memory leak refers to the memory created by the system which is not destroyed until reload. Temporary memory leaks refers to the memory created, used and finally destroyed when not needed.

Memory available bytes mean the remaining amount of physical memory. Cache bytes mean the static file cache size. A page fault per second refers to the overall rate of error pages. Pages per second if this increase the out of memory increases.

Memory pool It is a pool to accommodate objects created and used by the application and the operating system so filled pool means that the memory leak may occur.

Disk usage. Routine services for using disk should not always grow, there should be some cleaning routines applied.

Database queries. A database query is sent to a database in order to get information back from the database. All queries perform some type of data operation such as selecting data, inserting/updating data, or creating data objects such as tables. DB should be used in optimal way. No peak excessive queries into DB.

Data related to the software processes. The process data contains number of threads active, opened handles, processor activity, as well as physical and virtual memory allocation.

5.3 Performance of Contact Center

Performance of the contact center starts with web logins. During web logins there are data queries and transaction between web client, web server, application server and the database. These queries and transactions create load to the system, this load needs to be measured, analyzed in order to get the performance levels of the solution.

Performance continues with the other components involved in CC solution which enables the communication in the system. Communication in the system takes place with different channels that communicate with each other.

From the user perspective, communication can be internal and external. Internal communication is referring to the communication within same network. External communication is referring to the communication out of the system through gateways to external networks.

This communication system has its own communication logic for connection and routing rules. Communication events load the system that needs to be measured in terms of resource consumption, delays in communication and the data transfer during communication.

To get more knowledge on performance, the system needs to be loaded with multiple communication events. Effects of the events need to be monitored, and the data collected and analyzed.

5.4 Analyzing Performance of the CC

Performance analyzing starts with the collection of data during test runs, analyzing the collected data, and finally results and conclusion. The collected data contain system events such as the time when the events took place, delays of the events, and failure points if any.

The data is collected from the application and database servers in order to analyze the server performance. In addition, network traffic data is collected.

Depending on the test run, additional data is collected for analyzing the sound quality during by having test users manually logged to the system.

The collected data is analyzed, and test results are evaluated and finally test results are ok or failed. This leads to more tests, changes in the test setup, or changes in software modules in order to verify the performance requirements.

5.4.1 Performance Indicators in CC

In this chapter, the performance indicators in CC solution are introduced. These main indicators derive from events, delays, logins, components, network and sound quality.

Events and delays. In VoIP, network delay is caused when data packets (e.g. voice) take more time than expected to reach their destination. There may be different sources of delays in the VoIP networks.[13]

Logins. Web logins are the first events in CC. During login time there are delays in loading for different Web UI views make queries for directories, history, and others. These types of delays have impact only for C solution users. When lots of users log in, the delay has to be at the accepted level.

In the CC solution the communication travels from one end to the other end. Figure 10 illustrates the end-to-end voice flow in a compressed voice circuit.[13]

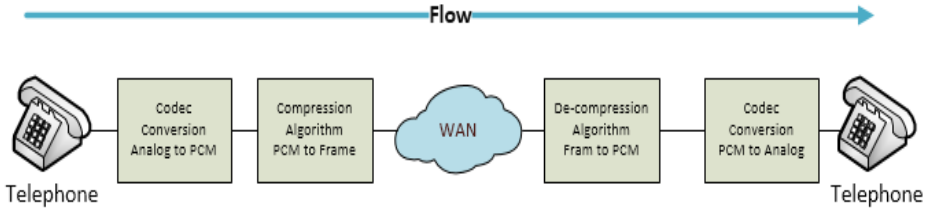

Figure 10. End-to-end voice flow

Communication has delays that have direct impact on communication quality and customer satisfaction. End-to-end software dependent delays need to be considered. The delays need to be reasonable and therefore they need to be followed during performance testing as per delays in RTP packages need to be very controlled since they have direct impact e.g. in the call quality.

CC components. CC components have also performance metrics. This metrics define the maximum number of agents in the system as well as the maximum of simultaneous calls that one individual component can run on a dedicated hardware. These components can be multiplied for extending the number of simultaneous calls, connections and agents in the system. Acceptance levels shown in Table 2 are based on the company internal experts.

Table 2. Single component performance metrics based on internal experts.

Single Component	Maximum Simultaneous
WEB Logins	1000 softphone agents
SIP Phone Logins	1000 desk phone agents
SIP Bridge	1000 calls
Connection Server	500 connection
Core component	2000 agents

Table 2 illustrates the maximum simultaneous events that one single component can handle.

Servers. CC components are installed on application and database servers. When system is loaded with communication events the server workload increases. This increase is monitored through a performance monitoring application, and the saved data is analyzed after each test run. Windows performance monitoring application can be used if the Windows operating system is in use.[14] Performance counters, such as processor utilization, memory usage, disk read and write, and network are commonly used.

In application servers, data related to the each tested process is collected. This process data contains number of threads active, opened handlings, processor activity, as well as physical and virtual memory allocation.

In database servers the data related to the hard drive read and write operations such as the read and write data request, and read and write bytes per second, are collected.

Network traffic. The communication between application and database servers can increase network traffic. The amount of sent and received data over network needs to be collected. LAN needs to be utilized first in order to estimate the amount of other network traffic.

Sound quality during heavy load. Sound quality usually is tested by the physical user logins to the system under test. Users can evaluate the sound quality during the call based on their experience.

5.4.2 Performance Acceptance Levels in CC

Performance acceptance level refers to the criteria of defined metrics for the performance of the CC solution. It is set based on the test run and described in the test case. Acceptance level should meet the requirements of the software. Acceptance level for web logins depend mostly on user experience, and it is acceptable that web logins take longer than opening other web UI after login. Also delays in performing searches in the web UI have to be at a reasonable level.

In CC communication the main delays are the transmission delays and connection delays. Transmission delays are standard transmission time and it has impact in the transmission quality of the user applications. The transmission delays should not exceed the maximum accepted level as shown in Table 3. Transmission delays are critical for having stability and quality in system. Connection delays in CC cannot be too long as the customer is waiting for live communication.

The acceptance level of delays should be defined based on the sources of delays and type of communication. For calls acceptance delays should be smaller than for example in e-mails. Table 3 lists the common types of delays in CC and their acceptance level based on the company internal experts and [13].

Table 3. Delay specification based on [13] and internal experts.

Types of delay	Acceptable level
Web UI login delays	5 – 30 seconds
Delays for another view in Web UI	2 – 10 seconds

Delays on Searching in Web UI	5 – 30 seconds
Transmission delays	0 -150 millisecond
Connection delays	1 - 5 seconds

Commonly used acceptance levels are total processor utilization of 60%, in memory there should be no memory leaks, disk reads and writes per second depend on disc manufacturer, and network utilization is at the maximum 50%.

6 PERFORMANCE OF CC SOFTWARE AND ITS IMPLEMENTATION BY LEAN

Performance in Lean should have focus on early testing and feedback. Measuring performance only once at the end of a long sequential development cycle may cause long delays and extra cost to the project. After many months of development it can be very costly to discover during performance testing that a key architectural decision was defective.

In Lean development, short cycles with early feedback loops are critical. Implementing less predictable things early and in short cycles that include performance measurements, the cost of delay is reduced.

Performance testing is very important for delivering the good product. Performance testing traditionally requires weeks for setups and therefore there is no time to take it from the start of the development. However, now there are many developed tools that enable performance testing from the beginning of the project.

Performance is important everywhere in Lean development lifecycle. Since performance is continual and repeatable process in software development, this continuous improvement is based on repeatable test runs and feedback. Performance needs to be looked at from both quantitative perspective (e.g. numbers, speed, numbers of users and so on) and qualitative perspective (e.g. how happy are the software users).

In order to implement performance aspects of a contact center solution in software development based on Lean methodology, a couple of approaches are be introduced in this chapter. These approaches will cover most of the performance aspects and the approaches will be introduced to the case company.

6.1 Approach 1: Performance Testing as Part of Every Development Cycle

It is important to understand that performance is not just an extra work but it is a must in order to be successful. Integrating performance aspect into the development cycles is done by getting performance into user stories. Since user stories have a simple description of a feature or functionality to be added into the product backlog, the backside (e.g. if stories are written in

paper) of user stories can be used for having performance aspects. The backside of user stories is used for noting items relevant to the performance.

In order to integrate performance into the development cycle it is very important to have people with special skills on performance in the team. This team can think of where the application is going to be run and what performance aspects are relevant to be checked.

Performance persons should be involved throughout the development cycle and they work as advisors in the team in order to have performance involved in every part of the development cycle.

Figure 11 shows a simplified picture on how performance could be integrated in the Lean agile development cycle.

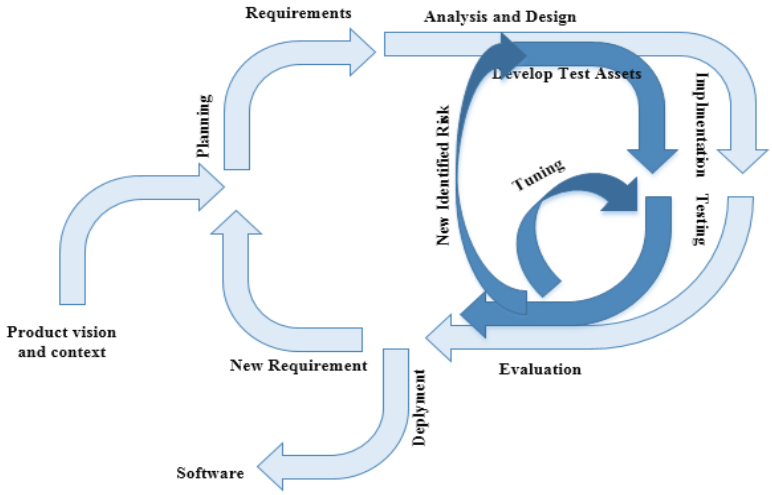

Figure 11. Integrating performance testing into Lean agile development cycle

In the analyzing and design phase in the development cycle, the stories are further discussed and analyzed, and there will be more details on how the feature described in the story will be built including the technical design. User acceptance test cases are created at this stage based on the assumption on how the feature will be used and the expected result.

Parallel with the analyzing and design phase the performance aspects of the product start to be analyzed. Performance relevant stories are created in the back side of the user story, or as a separate performance story. When stories are ready the development of test assets starts and as a result the

performance test cases are created and for the test to be executed the test environment is designed.

During the implementation phase of the development cycle in the performance side of that implementation the test environment is finalized for the test cases to be executed in testing phase.

Test execution phase can be done parallel in separate testing environments. For the user stories acceptance testing is done to verify the desired customer functionality in the product, and for the performance stories the test execution is done for desired test case output.

At the evaluation phase based on the testing result of the user stories, the evaluation process will determine that the user story will be part of the desired software, or in case of failure, it will get back to the requirement phase. As per performance aspect of the user story, if the story criteria are met, the testing is completed and the final report is done. In case of failure the team should be able to detect the source of failure and proceed with tuning in order to meet acceptance criteria. If tuning cannot be done the issue is stated as new risk identified and this new risk will get back to the developer test assets for further analysis.

The number of user stories to be picked up from backlog items will be based on the number of iterations, their length (e.g. 1 to 2 weeks) and the size of the team (e.g. small and cross functional). Target dates for iterations depends from estimated work amount that the teams can handle. The teams need to communicate on daily bases.

6.1.1 Iteration Workflow

Initially the stories with potential risks to the software are created and those stories are part of the back log items to be executed by the performance team. The performance team takes story points from backlog and start to work on them. Figure 12 illustrates a simplified iteration workflow.

The testing environment and test cases are created based on the story points. Execution of the tests cases starts when test cases and test environment is ready. Right after the execution the test case validation takes place and the base line is set for the future similar testing approach and the first test results will be used for benchmarking.

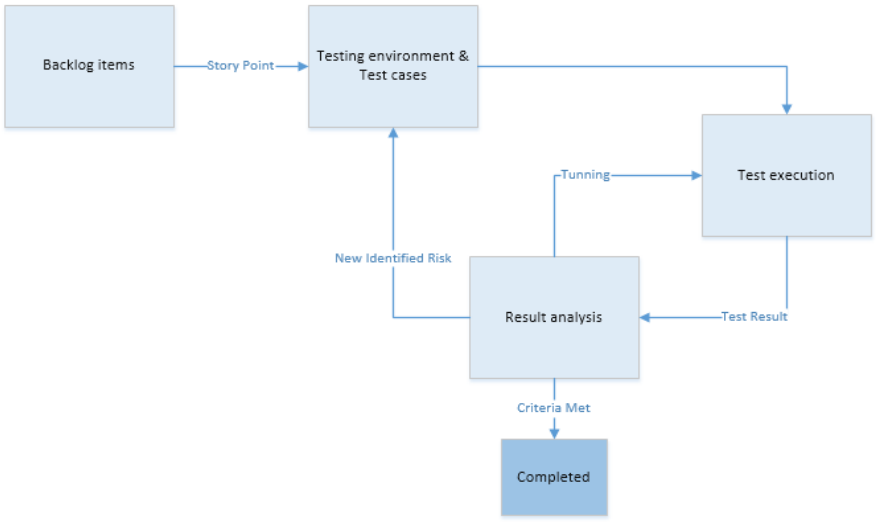

Figure 12. Iteration workflow

The test results are analyzed and classified based on the requirement validation. If criteria are met the test case is completed, and if not, some fine tuning can be done in the system in order to meet the acceptance criteria. If criteria are not met and tuning will not solve the issue, then new risk is identified at this point. Finally, the test result is documented in the test analyzing report. Iteration under work is finalized before going for next iteration.

6.1.2 First Iteration

In the first iteration, developing stories with performance aspects included starts at the unit level. Coding is done by defined coding practices like test driven development, which means that the measurement of performance starts from the unit testing.

Performance tests can be run on the unit testing level by calling functions and classes. After each cycle results are compared in a specific file and checked for big changes in the response time value. This will help to tackle issues, e.g. pointing into wrong DB, in the beginning of development, so that changes can be done before getting into continuous integration or acceptance phase or further.

These stories lead to the use of continuous system build that provides feedback every time new code is checked in the code repository, usually on a daily basis.

6.1.3 Second Iteration

The second iteration with story points with risks related to the software installation and the platform where the software will be run is described in Table 4.

Table 4. Software installation story points

Requirements	Story points
Software Installation	Hardware
	Operating system
	Databases

Story points based on the requirements on operating system, hardware and software under test are described in Table 5. They are based on the introduction in Chapter 5.2.5.

Table 5. OS, hardware and software under test story points

Requirements	Story points
CPU	Acceptable amount of CPU to be used
	CPU optimal performance at peak times
	No module back end tasks takes 100% of CPU
	CPU Utilization
	CPU burst
	CPU Spike
Memory	Stable memory growing
	Stable memory consumption
	No memory leaks
	Memory available bytes

Disk usage	Routine services should not always grow
	Clean routines for the routine services
Database queries	Data base queries data operation ok
	DB should be used in optimal way
	No peak excessive queries into DB
Software process performance data	Active threads
	Opened handles
	Processor activity
	Physical and virtual memory allocation

Table 5 illustrates the story point related to the requirements, as explained in Chapter 5.2.5.

6.1.4 Third Iteration

The third iteration starts with story points concerning end user requirements that are described in Table 6. They are based on the end user requirements explained in Chapter 5.2.1.

Table 6. End user requirements story points

Requirements	Story points
Login	Session opened for entering credentials
	Credential data is sent for the authentication
	Reasonable login delays
Authentication	By username and password
	By certificates
	By pin codes

Web UI	Phone, Monitoring, Reporting, Admin UIs
	Throughput
	Transaction per second
	Transactions Response Times
	Hits Per Second
Call Connection Response Time	Delay in connection of two ends
	Delay in voice stream
Voice quality	Acceptable (Excellent, good, ok)
	Unacceptable
Availability	In case of failure
	In case of maintenance

Table 6 illustrates the story point related to the requirements, as explained in Chapter 5.2.1.

6.1.5 Fourth Iteration

The fourth iteration may have stories with performance story points as listed in Table 7. They are based on the application performance requirements introduced in Chapter 5.2.2.

Table 7. Application performance story points

Requirements	**Story points**
Inbound events	Handling incoming events without errors or any insufficient delays
	Call routings events according to the routing rules
	Handling live events(e.g. phone call)
Outbound events	Handling outgoing events without errors or any insufficient delays

	Call routings events according to the routing rules
	Handling live events (e.g. phone call)
Automated services	DTMF keypad inputs
	IVR application states
	IVR application functions
Routing rules	Automatic call distribution routing,
	Conditional routing
	Customer data routing

Table 7 illustrates the story point related to the requirements, as explained in Chapter 5.2.2.

6.1.6 Fifth Iteration

The fifth iteration may have stories based on the backend requirements. The story points are based on the requirements presented in Chapter 5.2.3. Application performance story points are shown in Table 8.

Table 8. Server to server story point

Requirements	**Story points**
Server to server connection Applications and Database servers	Server processor and networking
	Processes between users, modules and data bases by sending retrieving events, storing, changing or removing data
	Amount of i/o file
	Storage capacities
High Availability	Controlling and monitoring processes
	Automatic recovery

Table 8 illustrates the story point related to the requirements, as explained in Chapter 5.2.3.

6.1.7 Sixth Iteration

The sixth iteration may have stories based on the network requirements. The story points are based on the requirements presented in Chapter 5.2.4. See application performance story points in Table 9.

Table 9. Network requirements story points

Requirements	**Story points**
Network traffic	Traffic between management and core network
	Traffic between core and access network
	Real time demands in network (Voice streams)
	Minimized network distance between endpoints
VoIP packages	Latency
	Packed loss
	Jitter

Table 9 illustrates the story point related to the requirements, as explained in Chapter 5.2.4.

6.1.8 Seventh Iteration

The seventh iteration for the performance team with stories more relevant to the performance testing types, as described in Chapter 3.2.

Load scaling of the system in order to test that system can be scaled by increasing load level.

Load capacity of the system in order to verify how much load essential system/components can handle.

Load stressing the system/components to the point where the system does not work properly or does not fulfill all requirements defined for its acceptable performance.

Stability and memory consumption for verifying the stability of the system and locating the possible memory leak issues.

Scalability of the system in order to verify that scalability of the system increases while increasing resources and as final result the performance of the system increases.

Flexibility and failure recovery. System must function correctly even though functionality is added, modified or removed from the system or hardware is added to the system while system is active. These test are very important for the system maintenance perspective.

Performance after upgrade. Performance of the system after upgrade is measured with some standard tests and the results is compared with those ones from earlier tests with previous software version.

This step finishes all iterations for this approach, and the performance aspects are the elementary part of the product.

6.2 Approach 2: Separate Team Dedicated for Performance

Performance team as a separate team in the software development cycle working on performance of the software. Figure 13 illustrates how tasks flow from the development team to the performance team. On top of the other aspects related to the CC solution, the development team analyses the stories picked from backlog, and in case there are performance concerns, the performance tests are required for that story.

Figure 13 Performance team as a separate team from development

Performance starts by system evaluation so the collection of information about the project as a whole, the functions of the system, the expected user activities, the system architecture, performance requirements and any other details of that kind are helpful in creating the performance testing strategy specific to the needs of the particular project. See Chapter 3.3 for more details.

After the system evaluation, there are a number of stories with potential risks to the software. Starting with the performance requirements of contact center solution in Chapter 5.2 and continuing with the side requirements as the hardware where the system will run, base installation involving operating system and data bases and finally the installation of the actual software and finalizing with the requirements on the performance testing types in Chapter 3.2. The stories are created and those stories will be part of the backlog items to be executed by the separate performance team.

The performance team takes story points from the backlog and starts to work on them. The number of the stories to be picked up from the backlog items will be based on the number of iterations, their length (e.g. 1 to 2 weeks) and the size of the team (e.g. small and cross functional). The target dates for iterations depend on the estimated amount of work that the teams can handle.

Communication between teams in this approach is very critical. The team needs to meet on a daily basis. Performance teams needs to inform the other teams on stories they are working on as well as on what was critical to the product they have thought of. If the teams that develop the product have this knowledge then the performance aspects are taken into consideration during product development in cycles. This approach makes sure that all stories are studied and performance aspects of the product are carefully analyzed.

The iteration workflow is the same as for the first approach but the content is not necessarily the same. The difference in iterations is that by this approach only items relevant to the performance team are placed in backlogs.

By this approach, the story point content in iterations can be planned more in advance compared to the first approach but this does not mean that it is the

final one as per new stories with high priority may come in form development team during their iterations.

7 RESULTS AND ANALYSIS

The advantages and disadvantages for the approaches introduced in Chapter 6 will be analysed in this chapter. Also the results and analyses based on the interviews on the approaches are presented here.

7.1 Approach 1, Advantages and Disadvantages

Analyzing the approach 1 and pinpointing the main positive aspects for recovering the performance of the solution in all levels of development and on the other hand, recovering the main negative aspects which may lead to the bad performance of the solution in general.

Advantages:

- Getting the performance stories into same user stories regarding feature or functionality.
- Integrating performance measurement into the unit testing.
- Taking product performance into account from the day one of the development life cycle.
- Involving performance testers throughout the cycle.
- Reducing cost of quality by fixing performance issues early in the cycle.
- Identifying major application defects and architectural issues early.
- Incremental performance testing of integrated modules. The incremental performance test should be compared in terms of progress.

Disadvantages:

- At early level only part of the developed product is available so iterations need to be repeated more often.
- Balancing workload inside the team, meaning that performance aspects of feature and feature to be developed have same priority.

- Team with people having different backgrounds requires more time on decision making, which tends to reduce efficiency.
- Team members, especially developers, may concentrate on development activities more and may neglect performance testing activities.
- Developers tend to have more creative skills while performance testing requires more of destructive skills.

These positive and negative aspects are part of the approach above. However, if negative aspects are analyzed by the team in advance they can become less critical for the success of this approach.

7.2 Approach 2, Advantages and Disadvantages

The approach 2 is analyzed based on the positive aspects for recovering the performance of the solution in all the levels of development and on the other hand, recovering the negative aspects which may lead to the bad performance of the solution in general.

Advantages:

- A specific team for performance testing provides skilled performance testers and cumulates knowhow.
- Team is not emotionally involved with the team who has built the product.
- Team members can support each other because of their common approach.
- Team develops its own plan for performance testing, self-leading team.
- It is possible to make more on-demand type testing.
- Decision making is easier as this separate team handles performance issues only, and team members have common background.

Disadvantages:

- Finding the critical aspects relevant to the performance testing is difficult as the team members are not involved in development. The team needs to have a performance tester in the development team as well.

- There are no natural, nor reliable ways to communicate possible requirement changes to a separate team. These may have impact to the performance.

- As developers do unit-level, component-level, integration-level testing, the performance team is basically doing the double repetition.

- Product delivery issues in case of issue during performance testing then release will be pending for the issue to be fixed first by development team.

- When to start performance testing is always a challenging decision.

These positive and negative aspects are part of the approach above. In the same way as in the first approach, if negative aspects are carefully analyzed by the team in advance they can become less critical for the success of this approach.

7.3 Results and Analysis Based on Approaches

Having a Lean agile development team and Performance testing team as separated entities will get complicated. After each delivery, the performance team needs to start from the beginning and analyze all variables relevant to the delivered product and the delivery will be postponed due to the performance tests. In addition, as to the end result, if there are performance issues then the product needs to get back to the development for another cycle.

Building a Lean agile development team with performance testers included one makes sure that all interactions will work, and after each delivery cycle the product has performance built in it.

Creating a Lean agile development team and later on trying to add performance in top of that it will not work. Performance testing into the Lean agile development has to start from the user stories and at the unit testing level.

Common aspects for both approaches related to the performance testing types such as the scalability and stability aspects can be tested when entire functionality is implemented and tested.

7.4 Results and Analysis Based on Interviews

The following chapters are based on three interviews with senior developers with various experiences in Lean methodology. The interview topics are listed in Appendix 1. This chapter includes straightforward guidelines proposed by the professionals having been interviewed.

When considering the functionality requirements from the customer perspective, it is important that the customer demands on how the system should behave have been correctly understood. If not, the requirements are false, and require bug fixes or feature improvements. In addition to bug fixing and features improvements, the customers demand new features to the product.

Functionality is the most important customer requirement. Make functionality requirements work and follow up with the bug fixes of the solution. How to implement it is a different topic (e.g. clean code and so on).

Reliability is the second most important thing after functionality. If a component is working fast but crashes frequently then it is not reliable enough and it is not working as desired. Functionality and reliability always go hand in hand with each other and one should always think about reliability while developing functionality.

Performance is very important and it should be included in the original design. In the contact center solution, many components have critical performance aspects. Develop solution functionality first and then follow up with performance if there are performance aspects. Performance of the feature should be determined based on the performance aspects used by customer.

Contact center is a live session and calls are critical, e.g. how many simultaneous users a customer demands in a live system, or the number of ongoing calls the system can handle (e.g. 2000 ongoing calls). These performance requirements can be very critical and have an effect on memory consumption and bigger points such as calls per second, user per seconds or user per time.

All performance starts with clear design, good design. Design the software in order to avoid database roundtrips because they have huge latency. For example, the good and efficient code that runs fast (e.g.in 2 microseconds) but needs to access database often (e.g. 5 times) can result in total to 500 milliseconds. So it really does not matter if the code runs faster in terms of microseconds if database roundtrips take many milliseconds. Instead, for example, drop database roundtrip times from 500 milliseconds to 70 milliseconds instead of developing on decreasing code running.

Data copying is a huge operation CPU-wise and handling lock is other issue. Having efficient code but forgetting locks, for example, having 5 CPU and having 20 threads in application working together can make performance even lousier instead of having 1 thread doing work. All threads try to read or write at the same time leading to the boundary condition that is very difficult to reproduce. This needs multiple time of execution but after a while, it can crash. Therefore, it may appear time to time to customers but not in short tests. In a multithreading environment it is necessary to find where the shared data is located and modify the locking mechanism so that threads are waiting for lock release and then access data in memory. If the question is what is gained by having 20 threads and 99% of the time is consumed for waiting lock mechanism the answer is when designing software consider the performance aspects.

The performance of the product can be tested at different stages. At the unit level, the unit testing performance is different from other performance testing types and therefore it requires a different approach. Code profiling can be used as a tool for measuring performance at the unit level. Checking code efficiency with code profiling tools at the unit test level, for example running code a million times and then checking where it spends most of the time. Code profile is a tool for measuring performance in the smallest possible

level e.g. a testing class by checking how efficient that class is and how the class may affect the total performance.

At the integration level, first identify most critical components and then everything depends on the defined tests. Define what is wanted, e.g. 10000 user logins? How can that be gained? Test a certain path of the product, test one feature of a chain of features such as the login process.

At the final product level, it is important to test the performance of the product as whole with the nonfunctional testing part including system resources e.g. CPU load, memory consumption.

Knowing the waterfall model, having a big project with low/no feedback and very person dependent, the idea of Lean on the other hand is good. Performance aspects need and should integrate on all stages of development. Tools are needed for integration and especially for performance measurement.

Lean methodology is based on development by delivering small tasks and working all the time in all aspects of the product. The approaches for having performance of software in product development by lean tend to solve lots of issues related to the performance of the software. For example starting with communication issues and going on with performance tasks in all development aspects.

When analyzing the approaches introduced it is not that black and white which one is better. In some cases Approach 1 is better and for some cases Approach 2 is better.

However, Approach 1 seems much better for starting to form a unit test level using profiling tools, continuing with integration and acceptance part of the product compared to Approach 2 where developers decide what is relevant for performance.

Approach 1 solves communication problems. People are aware of what others are doing and especially if applying methodology e.g. which uses stickers in board and having a team (developers and performance testers) that does the performance tests and gets that through till the end.

In Approach 2, the performance team has to wait until development team is ready in order to start working on that area, so always there may be a blocker.

Considering the differences of these approaches for overall product performance, Approach 2 seems better, however, for a software development cycle having small stories in all iterations leads to the first approach. The first approach would be better for Lean way of working. Delivering small tasks and working all the time in all the aspects of the product.

7.5 Comparisons of Approaches Based on Interview

Based on the interviews for suggested approaches, Approach 1 is much better at the unit and integration level of the software development. All performance aspects of the software are taken from the unit level, integration and following with the acceptance part of the ready product. However, Approach 2 is better when dealing with the performance testing types at the end of the project. The performance team is specialized in the performance testing types and the result may come faster and more detailed.

8 DISCUSSION AND CONCLUSIONS

Both of the suggested approaches aim to reduce the software development risks. These development risks refer to the requirements, design and development, coding and integrating testing and so on.

In order to be successful, the team's listening skills must be developed. These skills help the team to get complete information about customer needs, user stories, processes, software application and as well final product what management and customer desires.

Lean methodology provides a good ground for software development. It consists of proven principles, methods and tools that create processes for making development more efficient. Lean provides an ability to develop culture for encouraging people's creativity and skills especially in problem solving and continuous improvement.

The suggested approaches are aimed to cover the performance aspect of the contact center solution in software development based on Lean. However, the suggested approaches have not been implemented in software development and therefore the actual outcome is missing and conclusions about the outcome for the approaches is based on the interviews and common sense.

These approaches are good candidates to be implemented by the case company. the people interviewed encouraged the author to introduce the study to the architects and other stakeholders in the company. The stakeholders agreeing on one of the introduced approaches, it could be taken into use for benchmarking against the current, traditional waterfall development processes.

The introduced approaches were created to cover performance aspects of the contact center solution having Lean as a base. Since this is the first time performance testing is being woven in the Lean development methods more research is required on Lean, applying Lean in software development, and finally focusing on the performance of the software during all the stages of development.

REFERENCES

[1] James P Womack and Daniel T Jones. (2003). *Lean thinking.* New York.

[2] C.Larman and B.Wodde. (2009). *Scaling Lean and agile development*. Boston Pearson Education.

[3] Highsmith J. (2004). *Agile project management.* Addison-Wesley.

[4] SH L Pfleeger and J M Atlee (2010). *Software Engineering*. Boston Pearson Education

[5] Extreme Programing. Getting Started.

http://xprogramming.com/index.php

[6] A.Shalloway, G.Beaver and J.R.Trott. (2010). *Lean-agile software development*. Boston Pearson Education.

[7] C Kaner, J Falk and H Q Nguen (1999). *Testing computer software*. New York

[8] G J Myers, C Sandler and T Badgett (2011). *The art of software testing* Hoboken New Jersey.

[9] PerfTestPlus, Inc. A mental model for performance testing.

http://www.perftestplus.com/approach.htm

[10] B. Claveland (2008). *Contact Center Management Directory*. ICMI, Colorado Springs.

[11] Bocklund L and Bengston D (2002).*Contact Center Technology Demystified.* A Division of ICMI, inc Maryland

[12] Cisco. Quality of Service for Voice over IP.

http://www.cisco.com/en/US/docs/ios/solutions_docs/qos_solutions/QoSVoIP/QoSVoIP.html#wp1015329

[13] Cisco. Understanding Delay in Packet Voice Networks.

http://www.cisco.com/en/US/tech/tk652/tk698/technologies_white_paper09186a00800a8993.shtml

[14] Microsoft TechNet Library. Working with Performance Counters.

http://technet.microsoft.com/en-us/library/bb734903.aspx

Questions for Interview

Requirements:

1. How often customer/s demands for changes in the current product functionality?
2. How do you address those problems in terms of performance?

Performance:

3. What about performance of the product in current development process?
4. How do you measure performance at your company?
5. How performance testing is developed at you company?
6. How critical performance is for contact center solution?
7. Considering current software development methodology in use from where the performance measurement starts?

Software Development Methodology:

8. How code deployment is done at your company? Do you use TDD or other methods?
9. Are you happy with the current development process? If yes why? If no why
10. What do you think for the Lean software development?
11. Can we have performance aspects of solution integrated in lean development?
12. How would you implement performance in development by lean?

Approaches introduced for integrating performance in software development by Lean:

13. What do you thing for these approaches?
14. Consider the difference between those approaches. Would you use one of those approaches?
15. If yes then:
    a. What problems does this approach solve?
    b. How this approach fulfill lean way of working?
16. If no then:
    a. What approaches would you suggest?