Lappeenranta University of Technology

LUT School of Industrial Engineering and Management

Software Engineering and Information management

Master's thesis

# Software development methods and quality assurance: Special focus on South Korea

Examiners: Professor Kari Smolander

D.Sc. Ossi Taipale

Jesse Yli-Huumo

Orioninkatu 13 B 22

53850 Lappeenranta, Finland

Jesse.Yli-Huumo@lut.fi

Telephone: +358408210488

# ABSTRACT

Lappeenranta University of Technology

LUT School of Industrial Engineering and Management

Software Engineering and Information management

Jesse Yli-Huumo

**Software development methods and quality assurance: Special focus on South Korea**

Master's Thesis

2013

90 pages, 39 figures, 7 tables and 1 appendix

Examiners: Professor Kari Smolander, D.Sc. (Tech) Ossi Taipale

Keywords: software testing, quality assurance, agile methods, South Korean software industry

The purpose of this study was to explore software development methods and quality assurance practices used by South Korean software industry. Empirical data was collected by conducting a survey that focused on three main parts: software life cycle models and methods, software quality assurance including quality standards, the strengths and weaknesses of South Korean software industry. The results of the completed survey showed that the use of agile methods is slightly surpassing the use of traditional software development methods. The survey also revealed an interesting result that almost half of the South Korean companies do not use any software quality assurance plan in their projects. For the state of South Korean software industry large number of the respondents thought that despite of the weakness, the status of software development in South Korea will improve in the future.

# TIIVISTELMÄ

Lappeenrannan Teknillinen Yliopisto

Tuotantotalouden tiedekunta

Ohjelmistotuotannon ja tiedonhallinnan laitos

Jesse Yli-Huumo

**Ohjelmistotuotannon menetelmät ja laadunhallinta: Tarkastelussa Etelä-Korea**

Diplomityö

2013

90 sivua, 39 kuvaa, 7 taulukkoa ja 1 liite

Tarkastajat: Professori Kari Smolander, tekniikan tohtori Ossi Taipale

Hakusanat: ohjelmistotestaus, laadunhallinta, ketterät menetelmät, Etelä-Korean ohjelmistoteollisuus

Tämän tutkimuksen tarkoituksena oli kerätä tietoa Etelä-Korean ohjelmistotuotanto-alasta. Empiirinen datankeruu suoritettiin kyselyllä, jossa kiinnitettiin huomiota kolmeen pääaihealueeseen: ohjelmistotuotannon elinkaaren mallit ja menetelmät, ohjelmiston laadunhallinta ja laatustandardit, sekä Etelä-Korean ohjelmistoteollisuuden vahvuudet ja heikkoudet. Kyselyn tulokset osoittavat että ketterien menetelmien käyttö ohittaa perinteisten menetelmien käytön niukasti. Kysely paljasti myös kiinnostavan tuloksen, sillä melkein joka toinen Etelä-Korealainen yritys ei käytä minkäänlaisia laadunhallintasuunnitelmia projekteissaan. Suuri määrä vastaajista uskoi, että nykyinen Etelä-Korean heikko ohjelmistotuotannon tila tulee kuitenkin parantumaan tulevaisuudessa.

# AKNOWLEDGEMENT

I would like to express my thanks of gratitude to my supervisor D.Sc. Ossi Taipale who gave me the opportunity to do this thesis about South Korean software industry. He also helped me a lot by giving me guidance and instructions when I needed them. During this project I have learned a lot and I am really thankful for that.

Secondly I want to thank my family who have always given me guidance on my life and that is probably the biggest reason why I am here now. I also want to thank my girlfriend who helped me countless of hours in translation and data collection, that was backbone of this research, and also for supporting me every day.

# ABBREVIATIONS

| | |
|---|---|
| CEO | Chief Executive Officer |
| CTO | Chief Technical Officer |
| DSDM | Dynamic Systems Development Method |
| FDD | Feature-Driven Development |
| GDP | Gross Domestic Product |
| ICT | Information and Communications Technology |
| IEC | the International Electro technical Commission |
| IEEE | The Institute of Electrical and Electronics Engineers |
| IEEE-SA | The Institute of Electrical and Electronics Engineers Standards Association |
| ISO | the International Organization for Standardization |
| IT | Information Technology |
| LSD | Lean Software Development |
| OSEC | Office Suisse d'Expansion Commerciale (Switzerland Global Enterprise) |
| QA | Quality Assurance |
| ROI | Return on Investment |
| RUP | Rational Unified Process |
| SME | Small and Medium-Sized Enterprises |
| STX | Software Testing for Intended Quality -project |
| SQA | Software Quality Assurance |
| SQAP | Software Quality Assurance Plan |

| | |
|---|---|
| SQuaRE | Software product Quality Requirements and Evaluation |
| TEKES | Teknologian ja innovaatioiden kehittämiskeskus (Finnish Funding Agency for Technology and Innovation) |
| UAT | User Acceptance Testing |
| XP | Extreme Programming |

# LIST OF FIGURES

# LIST OF TABLES

# CONTENTS

# 1 INTRODUCTION

*"All software should be produced using some kind of methodology"*; when large and small pieces of software are developed with methodology in mind, it can improve development (Munassar & Govardhan, 2011). Software quality is the result of project management and software engineering. With the use of quality assurance it is possible to make the infrastructure to support software engineering methods, project management, and quality control actions (Pressman, 2005).

The purpose of this master's thesis is to study software life cycle models and methods and quality assurance in South Korea. We explore which software life cycle models and methods are being used in South Korea and especially how the use of agile methods affects to software projects. In software quality assurance the focus is on quality standards and software testing. The rest of the study describes South Korean software industry, focusing on its current state and future and also its strengths and weaknesses. This study was conducted in South Korea and all the empirical data is gathered from companies located in South Korea. Data is gathered by conducting a survey that was sent to the companies. There are three research questions that this study is trying to answer:

1) What software development methods and quality assurance plans are South Korean companies using during software development life cycle?

2) How the use of agile methods affect software development life cycle in South Korea?

3) What are the biggest strengths and weaknesses of South Korean software industry?

This master's thesis is part of the STX (Software Testing for Intended Quality) project. STX is a three-year international research project on software development, testing and quality. The objective of the STX project is to increase productivity in software engineering by lowering testing and development costs. The research goal is to explain

how software development, testing and quality depend on one another. The results allow companies to produce targeted level and type of quality more efficiently. STX is carried out by the Software Engineering Laboratory at the Lappeenranta University of Technology together with partners from industry and international academia. STX is funded by the TEKES (Finnish Funding Agency for Technology and Innovation) and the project partners. STX started in August 2011.

The thesis includes 6 chapters. The first chapter is introduction that tells the basic information related to this research. Second chapter, the theory part introduces agile methods and software quality assurance. Third chapter introduces the methodology of research that is being used in this thesis. Fourth chapter includes the empirical data of the survey and the results. Fifth chapter discusses about the results and sixth chapter contains the conclusion of the study.

# 2 SOFTWARE DEVELOPMENT METHODOLOGIES

According to Munassar & Govardhan (2011) all software should be produced using some kind of methodology. When large and small pieces of software are developed with methodology in mind, it can improve development a lot. A methodology is a systematic way of doing things where the process approaches through the early stages of software development to the working product. A methodology should specify what the process will produce when it is followed. A methodology also includes techniques for resource management, planning, scheduling and other tasks related to management. Munassar and Govardhan (2011) also add that good and widely available methodologies are essential for a mature software industry.

O'Docherty (2005) explains that a good methodology will address at least the following issues: Planning, scheduling, resourcing, workflows, activities, roles, artifacts, education. Guimaraes & Vilela (2005) say that there are a number of phases common to every development methodology, starting with requirements capture and ending with maintenance. During the last few decades a number of new software life cycle models have been introduced within the software engineering community. O'Docherty (2005) adds that with the traditional approach, you are expected to move from one phase to another in order. With the new modern approach, you are allowed to perform every phase more than once and in any order you want.

## 2.1 Traditional software development methods

Traditional methodologies are considered to be the traditional ways of developing software. These methodologies are based on a sequential series of steps, such as requirements definition, development, testing and deployment (Yu et al, 2012). There are many different traditional methods, in this theory part there will be only introduction of waterfall and spiral models. According to Nikiforova et al. (2009) the traditional software development methods are dependent on a set of predetermined processes and on-going documentation which is written as the work progresses and guides further development. The success of a project that is approached with traditional software development method relies on defining and knowing the requirements before

development phase. Implementing a change can be difficult. However with traditional methods it is easier to determine the costs of the project, set a schedule and allocate resources accordingly (Gornik, 2001).

Yu et al. (2012) defines four phases which are characteristic of a traditional software development method. The first step is to define the requirements for the project and design the schedule which will be used in various phases of the project. Once the requirements are defined, the next step is the design and architectural planning phase. In this phase a technical infrastructure is produced in the form of diagrams and models. These bring to the surface potential issues that the project may face as it progresses and provide a workable road map for the developers to implement (Yu et al, 2012).

After the team is satisfied with the architectural and design plan, the project moves to next phase, where the code is produced until it reaches defined goals. Usually development is broken down into smaller tasks that are distributed among different teams based on their skills and knowledge. After development there is usually testing, but the testing phase often overlaps with the development phase to ensure that issues are found earlier. When the project is coming to the end and the developers are close to meeting the project requirements, the customer will join the testing and feedback cycle and the project will be completed to final form (Yu et al, 2012).

### 2.1.1 Waterfall model

The waterfall model is the traditional model of software engineering. The waterfall model is one of the oldest and most used methods and it has been widely used in government projects and in many major companies. Feature of this model is planning in early stages to reveal design flaws before they develop. Also model encourages intensive documentation and planning, that makes it work well for projects where quality control is important factor (Munassar & Govardhan, 2010). Punkka (2005) explains that waterfall takes the process and divides it into separate phases which follow each other. Normally phases include requirements, analysis, design, implementation and integration, but there can also be different variations. Proceeding to the next phase is sequential and it requires a certain criteria to be completed. The problem with a sequential model is that it is combination of different fields; its process is inherited from

engineering and production process model. This leads to situation where software's unique nature is not taken under consideration.

According to Petersen et al. (2009) the waterfall model used at the company runs through the phases: requirements engineering, design & implementation, testing, release, and maintenance. Quality check is being done between all the phases and for moving to the next phase it needs to be passed, this approach is referred to as a stage-gate model. Figure 1. shows an overview of the process.

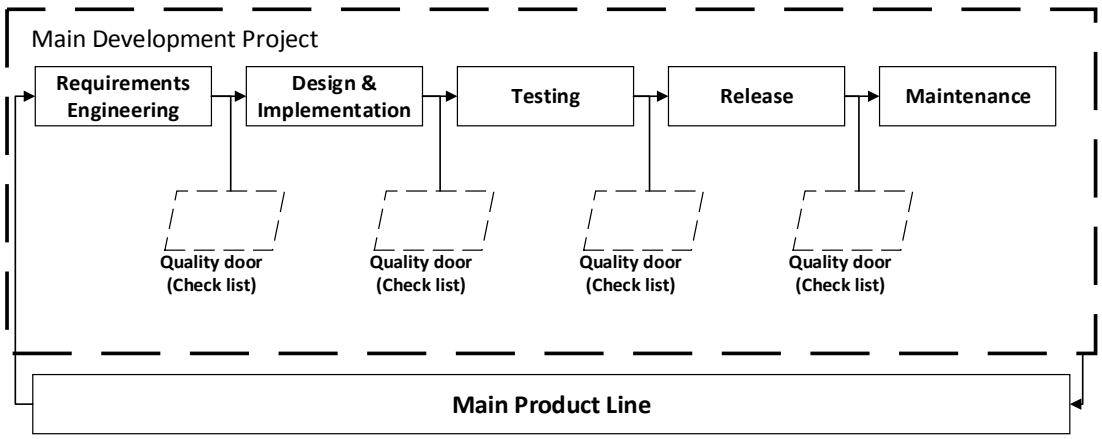


Figure 1. Waterfall development at company (Petersen et al., 2009)

Petersen et al. (2009) explain the steps of waterfall model as following:

*Requirements Engineering:* In this phase, the needs of the customers are identified and documented on a high abstraction level. After this, the requirements are refined more accurate that they can be used as input for next phase. All of the requirements are stored in a requirements repository. At the first quality gate (also other gates) it is checked whether all requirements are understood, agreed upon, and documented.

*Design and Implementation*: After defining requirements, next phase is design and implementation. In the design phase the architecture of the system is created and documented. After design of the architecture is completed, the actual developing and coding starts. The developers also conduct basic unit testing before passing the developed code over to next phase. The quality gate checklist verifies whether the

architecture has been evaluated and does it differentiates from the previous quality gate decision.

*Testing*: In this phase the system integration is tested regarding quality and functional attributes. Purpose of this phase is to make decision whether the developed system can be deployed. Measures of quality and functional attributes are collected in the quality system. In a case where company provides complete solutions (includes hardware and software) the tests have to be conducted on a variety of hardware and software configurations because those might differ between customers. The quality gate checklist is reviewed to see whether the developed system has been verified and does it differ from previous quality gates. Quality gate checks whether the outcome of the project meets the customers' requirements.

*Release*: Before the final phase, the product is now in a shippable state. Release documentation is now finalized (e.g. installation instructions of the system for customers and user-guides). At the quality gate it is checked if the outcome meets the customers' requirements, has the customers accepted the product, and whether the final products outcome was done in time and does it fulfill its quality requirements. A post-mortem analysis is also conducted at the end of this phase.

*Maintenance*: During the last phase the product has been released to the customer and it has to be maintained. In a case where customer discovers problems related to product they report them to the company and get support. If the problems are due to mistakes in the product, updates of the system are delivered to the customers.

There are also some problems regarding waterfall. Brooks (1987) states, that properties of software are complexity, conformity, changeability, and invisibility. When looking at the principles of traditional software development methods, they do not quite meet the properties of the software. Also Munassar & Govardhan (2010) say that many people believe that this model cannot be applied to all situations. For example, when using the waterfall model, the requirements must be defined before the design, and the complete design must be stated before development. This results that there is no overlapping between phases. It is not prohibited returning to an earlier phase in the waterfall method, for example, return from the design phase to the requirements phase. Problem is that returning can involve costly rework. Because the actual development comes late in the process, one does not see results for a long time. Munassar & Govardhan (2010) also

add that although the waterfall model has its weaknesses, it is instructive because it underlines important and basic phases of the project life cycle.

### 2.1.2 Spiral model

According to Boehm (1986) the spiral model of the software life cycle methods (Figure 2.) has evolved from various refinements of the waterfall model. Munassar & Govardhan (2010) explain that spiral model is similar to the incremental model, but it emphasizes more on risk analysis. The spiral model is divided into four phases: Planning, Risk Analysis, Engineering and Evaluation. A software project passes these phases in iterations that continue repeatedly. These iterations are called Spirals in this model.

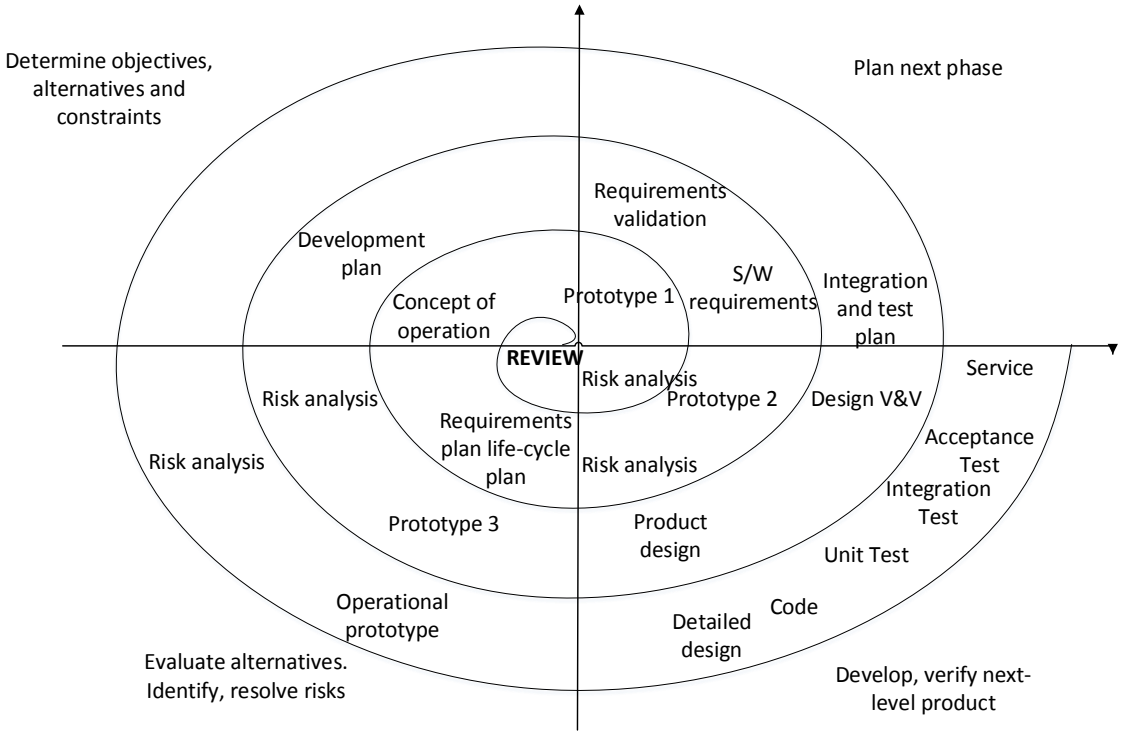


Figure 2. The spiral model diagram (Sommerville, 2004)

Boehm (1986) explains phases of the spiral model. In typical cycle of the spiral each cycle begins with the identification of three different parts:

- The objectives of different part of the product being elaborated (performance, functionality, ability to accommodate change, etc.)
- The alternative requisites of different parts of the product.
- The limitations imposed on the application of the alternatives.

The next phase is to evaluate the alternatives compared to the objectives and constraints. The purpose of this is to identify areas that are significant sources of project risk. If any kind of risk is found, the next step should involve cost-effective strategy for resolving the sources of the risk. Strategy might involve prototyping, simulation, benchmarking, reference checking, administering user questionnaires, analytic modeling, or combinations of these and other risk resolution techniques (Boehm, 1986).

After the evaluation of the risks, the next phase is determined by the relative remaining risks. If performance or user-interface risks strongly dominate program development or internal interface-control risks, the next step may be an evolutionary development one: a minimal effort to specify the overall nature of the product, a plan for the next level of prototyping, and the development of a more detailed prototype to continue to resolve the major risk issues. If this prototype is operationally useful and robust enough to serve as a low-risk base for future product evolution, the subsequent risk-driven steps would be the evolving series of evolutionary prototypes going toward the right in Figure 2 (Boehm, 1986).

In a case where previous prototyping already resolved all of the performance or user-interface risks, and program development or interface-control risks control, the next step follows the basic waterfall approach. In this case each level of software specification in the figure is then followed by a validation step and the preparation of plans for the succeeding cycle. In this case, the options to prototype, simulate, model, and so on are addressed but not exercised, leading to the use of a different subset of steps (Boehm, 1986).

Munassar & Govardhan (2010) describe the advantages and disadvantages of the spiral model. Advantages include high amount of risk analysis, good fit for large and mission-critical projects, and the software is produced early in the software life cycle.

Disadvantages of the spiral model are that it can be a costly model to use, risk analysis requires specific expertise and project's success is dependent on the risk analysis phase.

## 2.2 Agile software development methods

Agile software development methods are based on iterative and incremental development using short development cycles (Highsmith, 2002). The most important priority of agile methods is to make the customer satisfied with early and continuous delivery of software. Fowler & Highsmith (2001) define four main values that agile methods emphasize:

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

Agile methods identify the importance of process and tools, but interactions between skilled individuals are an even greater asset to the project. Also documentation is not necessary bad thing for the project, but the main focus must remain in the final product delivery. Therefore, every project team needs to determine what documentation is necessary. Communication between customer and development team during the entire project life cycle to know what customer really wants is more important than contract negotiation. Although some of the internal and external contracts are necessary to do. Following a plan is important aspect of software development, but without ability to respond to changes, project can turn in to failure (Fowler & Hightsmith, 2001).

Fowler & Highsmith (2001) explain in agile manifesto the twelve principles that describe what it means to be agile.

**1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.** Customer value principle is hard to implement. Usually assumption in traditional project management is that completed project equals customer value. Agile methods demands that customer value needs to be evaluated frequently and

fulfilling requirements that were received in the first meeting do not equal successful project (Fowler & Hightsmith, 2001).

**2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.** Instead of resisting changes during the software life cycle, agile methods are trying to understand and agree with the changes (Fowler & Hightsmith, 2001).

**3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference for the shorter timescale.** Agile methods use an incremental or iterative style of software development, with multiple deliveries of functionality. It is important to remember that delivery is not the same as release (Fowler & Hightsmith, 2001).

**4. Business people and developers work together daily throughout the project.** When beginning the use of agile methods focus on high-level requirements and keep the amount of low-level requirements down. This is the way how agile methods prepare for changes in the requirements. Communication between business people and developers when using agile methods should be frequent and meetings should be arranged on daily basis to keep developers informed about requirements. This is also the way of keeping the customer included during the project life cycle (Fowler & Hightsmith, 2001).

**5. Build projects around motivated individuals, give them the environment and support they need and trust them to get the job done.** In the project you have to trust your people that are working. You have to give them the tools, environment, technologies, and processes. Trust is the biggest problem between people. Decisions must be made by a person who has the most experience of the situation. This means that managers must trust their staff to make correct decisions (Fowler & Hightsmith, 2001).

**6. The most efficient and effective method of conveying information with and within a development team is face-to-face conversation.** The goal of agile methods is to transfer information efficiently between the teams included in the project. Danger when exchanging information in the form of documents is that information is misunderstood. This is why in agile methods the communication is preferred face-to-face (Fowler & Hightsmith, 2001).

**7. Working software is the primary measure of progress.** There are lots of projects where the team doesn't realize that they are in trouble until the delivery time of the software. Requirements, design and coding might be done in time, but testing and integration are taking more time than planned. This is why agile methods prefer iterative software development where it is possible to set up milestones during the process.

**8. Agile processes promote sustainable development. The sponsors, developers and users should be able to maintain a constant pace indefinitely.** In the software development there will always be errors and bugs, and that is the reason why people have to work long nights and weekends. Agile methods prefer people that are creative and able to work the whole software life cycle. Sustainable development means that employees have the same working pace (40 or so hours a week) all the time. This results that team can sustain over time and remain healthy (Fowler & Hightsmith, 2001).

**9. Continuous attention to technical excellence and good design enhances agility.** Agile methods are seeking good quality in design, because it is essential maintaining agility. One of the tricky aspects is that agile methods encourage the alteration of requirements; this is why design can't be fully completed. Instead design phase is continuous activity during entire project, where every iteration will have own design work (Fowler & Hightsmith, 2001).

**10. Simplicity the art of maximizing the amount of work not done is essential.** In agile methods simplicity is particularly important factor. With simplicity it is easier to adapt to changes in requirements. It is easier to add something simple to the project rather than take away something complicated (Fowler & Hightsmith, 2001).

**11. The best architectures, requirements and designs emerge from self-organizing teams.** Best architectures are made in iterative development. The second point of the principle is that emergent properties are best generated from self-organizing teams in which the interactions are high and the process rules are few (Fowler & Hightsmith, 2001).

**12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.** Agile methods are not something that you just pick and follow straight. This is why it is really important to check your team

practices and search for possible improvements that could be made (Fowler & Hightsmith, 2001).

### 2.2.1 Scrum

Scrum belongs to the family of agile software development methods that have attracted significant attention among software practitioners (Mahnic & Drnovscek, 2005). Kane and Schwaber (2002) describe Scrum as product development methodology that consist of practices and rules to be used by management, customers and project management to maximize the productivity and value of the development. Sutherland (2010) defines Scrum as an iterative and incremental framework for software development. The idea of Scrum is to develop software in cycles called Sprints. These iterations are less than one month long and usually measured in weeks.

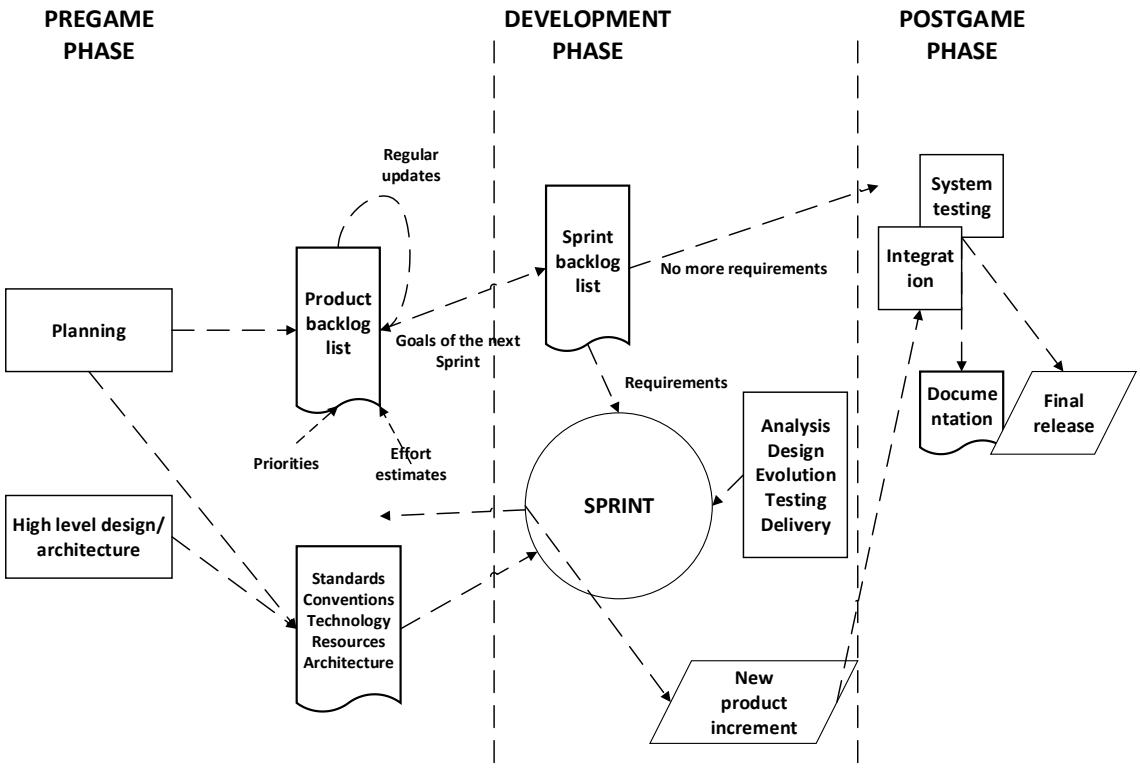


Figure 3. Scrum process (Schwaber, 1995)

Schwaber (1995) divides Scrum process into three separate phases (Figure 3.):

**Pregame phase:**

*Planning:* In this phase product backlog is defined, with an estimate of its schedule and cost. If a new system is being developed, this phase includes both conceptualization and analysis. If an existing system is being updated, this phase consist only of analysis (Schwaber ,1995).

*Architecture:* After planning phase the backlog items implementation will be designed (Schwaber ,1995).

**Development phase:**

*Development Sprints:* Development of software with constant respects to the variables of time, requirements, quality, cost and competition. Development phase includes multiple Sprints that are used to develop the system (Schwaber ,1995).

**Postgame phase:**

*Closure:* In the last phase the product is ready for release and final documentation and pre-release testing are done (Schwaber ,1995).

Sutherland (2010) identifies five steps inside one Sprint:

**The Product Backlog**: A Scrum project is led by the Product Owner and documented in to the Product Backlog. The Product Backlog is list which includes requirements of the project. The Product Backlog evolves over the software development lifetime and items in it are continuously reprioritized. **The Sprint**: Sprints are iterations of the work. They are typically 1-4 weeks long. The Sprints end on a specific date whether the work has been completed or not. **Sprint Planning**: At the beginning of every Sprint, the Sprint Planning Meeting takes place. Product Owner and the Scrum Team review the Product Backlog and sets up the goals for the upcoming Sprint. Each item picked up from the Product Backlog is broken down into a set of individual tasks for the team (Sutherland, 2010).

**Daily Scrum Meeting**: During the Sprint, the Scrum team keeps Daily Scrum Meetings. This is a short meeting that happens every workday. At this meeting, the progress of the items is presented. Based on the information received from the meetings,

further discussion or re-planning on items can be done. **Sprint Review and Retrospective**: At the end of the Sprint, there is the Sprint Review, where the Scrum Team and stakeholders inspect what was done during the Sprint and what to do next. Following the Sprint Review, the team gets together for the Sprint Retrospective where the team discusses what are positive and negative things in the project (Sutherland, 2010).

According to Deemer et al. (2010), there are three roles in development with Scrum: The Product Owner, The Team, and The Scrum Master. Together these are known as The Scrum Team**. The Product Owner** is responsible for maximizing return on investment (ROI) by leading the Scrum project by prioritizing and updating the product backlog for the next Sprint. In some cases, the Product Owner and the customer is the same person. In other cases the Product Owner works as a project manager, however The Product Owner is different compared to the normal project manager because they interact more with the team (Deemer et al., 2010).

**The Team** builds the product that the Product Owner has defined. The Team in Scrum is cross-functional and it includes people with skills in analysis, development, testing, interface design, database design, architecture and documentation. Team in Scrum is usually seven plus or minus two people. The Team develops the product and gives ideas and suggestions to the Product Owner how to make the product better (Deemer et al., 2010). **The ScrumMaster** helps everybody at the product group to apply Scrum. The ScrumMaster is not the project manager of the Team, but instead, he/she educates and guides the Product owner and the Team in the skilful use of Scrum (Deemer et al., 2010).

### 2.2.2 eXtreme Programming

Extreme Programming (XP) is one of the several popular agile methods. It has proved to be very successful at many companies and projects worldwide. Key behind the successfulness of the Extreme Programming is that it stresses customer satisfaction and it empowers developers for changing customer requirements, even late in the life cycle. Extreme programming gathers managers, customers and developers to a collaborative team (Extreme Programming, 2009).

XP is based on the following values and core principles. Misic (2006) describes 4 basic values that are included in the XP. **Communication** helps everyone to understand what the project is about. This principle shows the fact that developing software is an intensive process between teams and communication helps them to cooperate more effectively and efficiently. **Embracing change** is important part of software development. During the software life cycle requirements, priorities and designs have possibility to change. This is the reason why the development process should be able to embrace change, and still maintain the best possible results. **Feedback** helps the communication to reach its full potential. Last principle in Extreme Programming is **simplicity.** The goal of development is to find out the simplest solution that can work. Simplicity translates into efficiency, but also into effectiveness. These four basic values or principles are the basis of the practices used in the XP.

Beck (1999) introduces in his book 12 practices that are used in the XP:

- Planning – The programmer estimates the effort needed for the implementation of customer stories and the customer decides the scope and timing of releases based on estimates.
- Small/short releases – An application is developed in a series of small, frequently updated versions. New versions are released anywhere from daily to monthly.
- Metaphor – The system is defined by a set of metaphors between the customer and the programmers which describes how the system works.
- Simple Design – The emphasis is on designing the simplest possible solution that is implemented and unnecessary complexity and extra code are removed immediately.
- Refactoring – It involves restructuring the system by removing duplication, improving communication, simplifying and adding flexibility but without changing the functionality of the program
- Pair programming – All production code are written by two programmers on one computer.
- Collective ownership – No single person owns or is responsible for individual code segments rather anyone can change any part of the code at any time.
- Continuous Integration – A new piece of code is integrated with the current system as soon as it is ready. When integrating, the system is built again and all tests must pass for the changes to be accepted.

- 40-hour week – No one can work two overtime weeks in a row. A maximum of 40-hour working week otherwise it is treated as a problem.
- On-site customer – Customer must be available at all times with the development team.
- Coding Standards – Coding rules exist and are followed by the programmers so as to bring consistence and improve communication among the development team.

According to Abrahamsson et al. (2002) there are total of 6 phases in Extreme Programming process: Exploration, Planning, Iterations to release, Productionizing, Maintenance and Death (Figure 4).

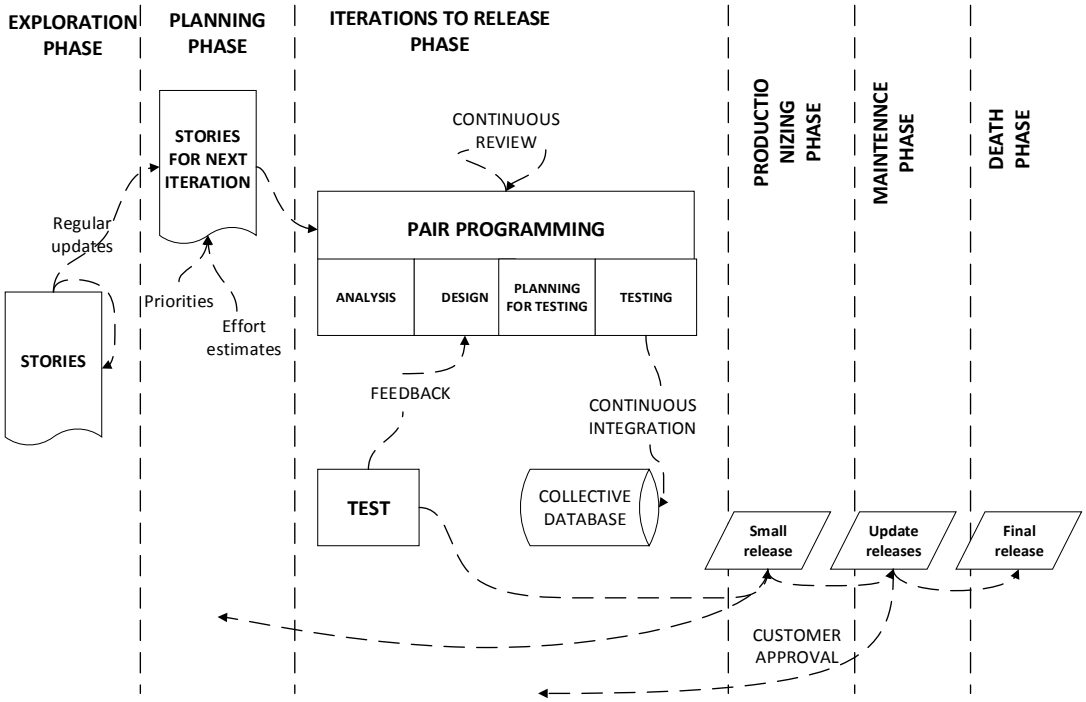


Figure 4. Extreme Programming process (Abrahamsson et al., 2002)

In the **Exploration phase**, the customers write out the story cards that they wish to be included in the first release of the software. Story cards describe features that are needed into the program. Also in this phase the project team familiarizes themselves with the tools, technology and practices that they will be using during the project. Usually the

exploration phase takes time between a few weeks to a few months, depending on how familiar the technology is to the programmers (Abrahamsson et al., 2002).

The **Planning phase** sets the priority order for the story cards. The programmers' job is to estimate how much effort and time each story requires. Based on their estimations, the schedule of the project is made. The planning phase itself takes only a couple of days and the first release of the software doesn't usually exceed two months (Abrahamsson et al., 2002).

The **Iterations to the release phase** includes many iterations of the system before the first release of the software. The schedule that was made in the planning phase is divided into a number of iterations that will each take one to four weeks to implement. The purpose of the first iteration is to create a system with the architecture of the whole system. The customer makes the decision about what stories are chosen for each iteration. The functional tests that are created by the customer are run at the end of every iteration. After the last planned iteration the system is ready for the production phase (Abrahamsson et al., 2002).

The **Productionizing phase** is for extra testing and checking that the system can be released to the customer. At this phase it is possible that new changes might be found and decision has to be made if they are included in the current release. In a case where changes are decided to be added, iterations must be quickened e.g. from three weeks to one week. The changes that are decided not to be included are documented for the later implementation (Abrahamsson et al., 2002).

**The Maintenance phase** is for keeping the system's production version running while also producing new iterations. This phase also requires customer support. Thus, the maintenance phase decreases the development velocity and it is possible that new people are incorporated to the team (Abrahamsson et al., 2002).

The final step is **Death phase** where the customer does not have any more stories to be implemented. This is the time in the Extreme Programming process when the necessary documentation is finally written. It is also possible that death phase might happen if the system is not delivering wanted outcomes, or if the project don't have budget for further development (Abrahamsson et al., 2002).

## 2.3 Difference between traditional and agile software development methods

Agile methods emphasize teams, working software, customer collaboration, and responding to change, while traditional methods focus on contracts, plans, processes, documents, and tools (Fowler & Highsmith, 2001). Nerur et al. (2005) explain the main differences between traditional and agile methods (Table 1.)

Table 1. Traditional vs. agile software development (Nerur et al., 2005)

| | **Traditional** | **Agile** |
|---|---|---|
| **Fundamental Assumptions** | Systems are fully specifiable, predictable, and can be built through meticulous and extensive planning. | High-quality, adaptive software can be developed by small teams using the principles of continuous design improvement and testing based on rapid feedback and change. |
| **Control** | Process centric | People centric |
| **Management Style** | Command-and-control | Leadership-and-collaboration |
| **Knowledge Management** | Explicit | Tacit |
| **Role Assignment** | Individual - favors specialization | Self-organizing teams – encourages role interchangeability |
| **Communication** | Formal | Informal |
| **Customer's role** | Important | Critical |
| **Project Cycle** | Guided by tasks or activities | Guided by product features |
| **Development Model** | Life cycle model (waterfall, spiral, or some variation) | The evolutionary-delivery model |
| **Desired Organizational Form/Structure** | Mechanistic (bureaucratic with high formalization) | Organic (flexible and participative encouraging cooperative social action) |
| **Technology** | No restriction | Favors object-oriented technology |

Boehm & Turner (2003) describes the five axes (Figure 5) that are used to distinguish the difference between agile and traditional software development methods. By using the five axes it is possible to recognize what kind of software development method might be suitable for your project. To see more precise explanation of the difference between agile and traditional methods in five axes see table 2.

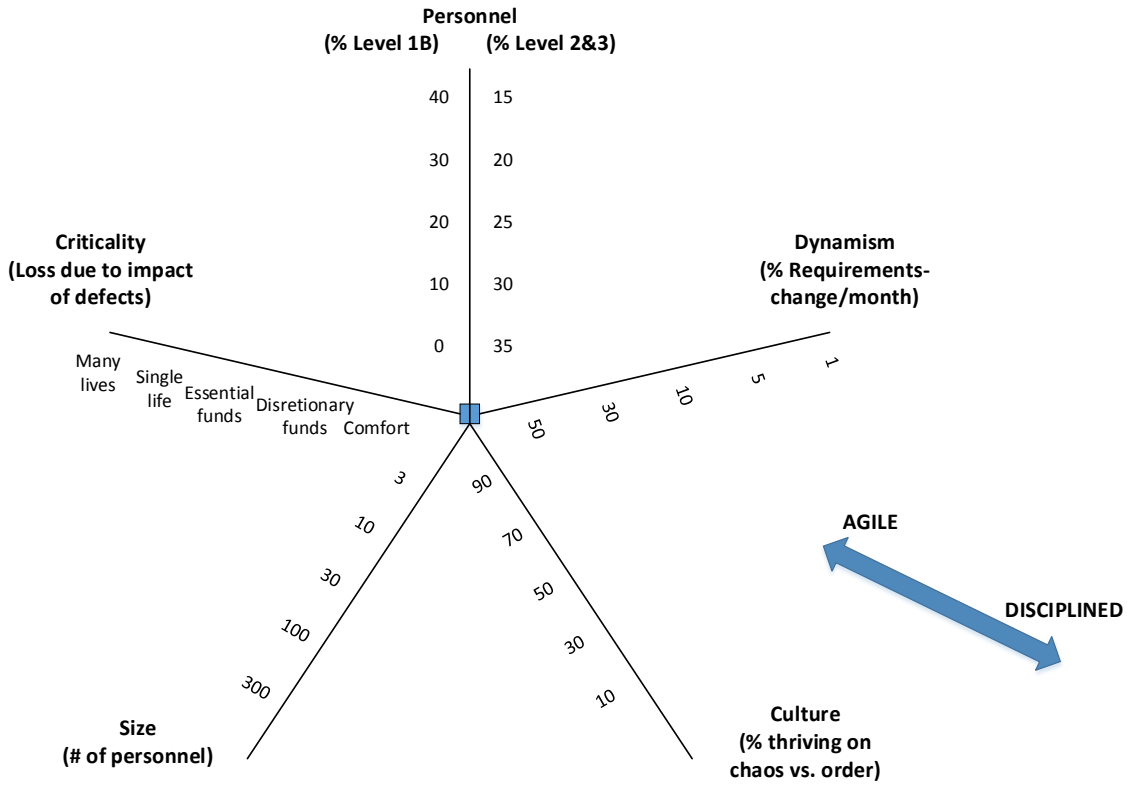


Figure 5. Differences between agile and traditional methods (Boehm & Turner, 2003)

Table 2. The five critical agility and plan-driven factors (Boehm & Turner, 2003)

| Factor | Traditional discriminators | Agility discriminators |
|---|---|---|
| **Size** | Methods evolved to handle large products and teams; hard to tailor down to small projects. | Well matched to small products and teams; reliance on tacit knowledge limits scalability. |
| **Criticality** | Methods evolved to handle highly critical products; hard to tailor down efficiently to low-criticality products. | Untested on safety-critical products; potential difficulties with simple design and lack of documentation. |
| **Dynamism** | Detailed plans and "big design up front" excellent for highly stable environment, but a source of expensive rework for highly dynamic environments. | Simple design and continuous refactoring are excellent for highly dynamic environments, but present a source of potentially expensive rework for highly stable environments. |
| **Personnel** | Need a critical mass of scarce Cockburn Level 2 and 3 experts during project definition, but can work with fewer later in the project—unless the environment is highly dynamic. Can usually accommodate some Level 1B people. | Require continuous presence of a critical mass of scarce Cockburn Level 2 or 3 experts; risky to use non agile Level 1B people. |
| **Culture** | Thrive in a culture where people feel comfortable and empowered by having their roles defined by clear policies | Thrive in a culture where people feel comfortable and empowered by having many degrees of freedom; thrive on chaos. |

## 2.4 Software quality assurance

According to Pressman (2005) software quality does not just appear. Software quality is the result of good project management and software engineering. With the use of quality assurance it is possible to make the infrastructure to support software engineering methods, project management, and quality control actions. The purpose of quality assurance is to provide management and technical team with the data necessary to achieve product quality. Owens & Khazanchi (2009) define software quality assurance (SQA) as well defined and repeatable process that is part of the project management and the software development lifecycle. The objective of SQA is to assure conformance to requirements, reduce risk and improve quality.

ISO (the International Organisation for Standardisation) and IEC (the International Electro technical Commission) form the specialised system for world-wide standardisation. The International Organization for Standardization is the most widely recognized standard-setting body in the world (ISO 9126-1, 1999). Standards that are reviewed in this study are ISO/IEC 9126 for the evaluation of software quality, ISO/IEC 14598 for the methods that are used to measure and evaluate software product quality, and ISO/IEC 25000 that consists of series 250nn standards which describe the quality model.

### 2.4.1 ISO 9126

ISO/IEC 9126 is currently one of the most widespread quality standards. In its actual form it embraces both quality models and metrics (Botella et al., 2004). ISO 9126 is a standard for software product evaluation. The goal of this standard is to focus on some well-know human biases that can affect to the quality in software development process. In Figure 6 a quality model framework is described. It explains the relationship between different approaches to quality (ISO 9126-1, 1999). Model includes internal quality attributes, external quality attributes, and quality in use attributes.

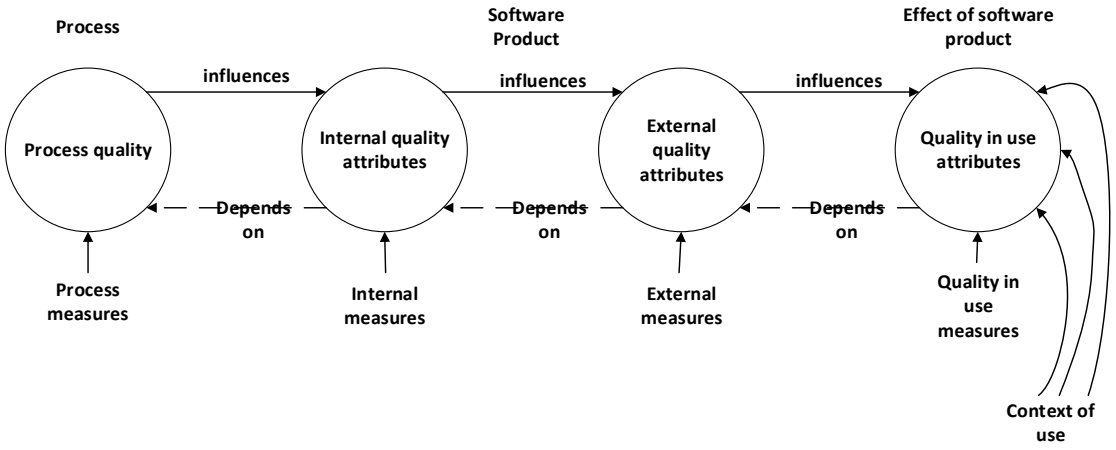


Figure 6. Quality in the lifecycle (ISO 9126-1, 1999)

Quality model for external and internal attributes is shown in Figure 7. The model is divided into six characteristics (functionality, reliability, usability, efficiency, maintainability and portability), which are further divided into sub characteristics.

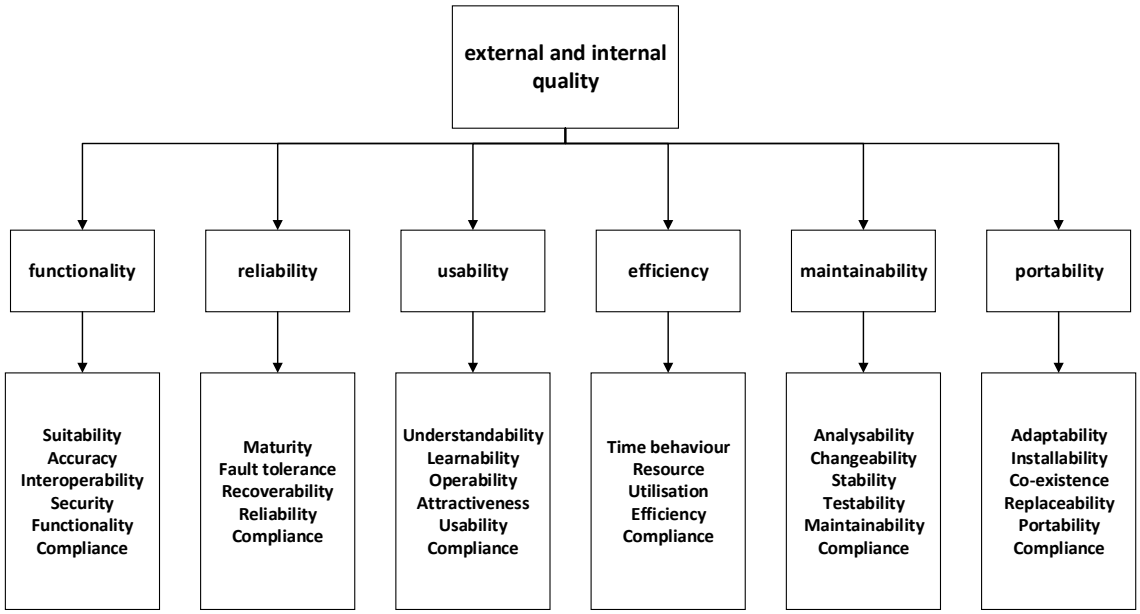


Figure 7. Quality model for external and internal quality (ISO 9126-1, 1999)

Table 3. External and Internal Characteristics of ISO 9126 (ISO 9126-1, 1999)

| Characteristics | Definition |
|---|---|
| **Functionality** | The capability of the software product to provide functions which meet stated and implied needs when the software is used under specified conditions. |
| **Reliability** | The capability of the software product to maintain a specified level of performance when used under specified. |
| **Usability** | The capability of the software product to be understood learned, used and attractive to the user, when used under specified conditions. |
| **Efficiency** | The capability of the software product to provide appropriate performance, relative to the amount of resources used, under stated conditions. |
| **Maintainability** | The capability of the software product to be modified. Modifications may include corrections, improvements or adaptation of the software to changes in environment, and in requirements and functional specifications. |
| **Portability** | The capability of the software product to be transferred from one environment to another. |

Table 4. External and Internal Sub Characteristics of ISO 9126 (ISO 9126-1. 1999)

| Characteristics | Sub Characteristics | Definition |
|---|---|---|
| **Functionality** | Suitability | The capability to provide functions |
| | Accuracy | The capability to provide results |
| | Interoperability | The capability to interact with other systems |
| | Security | The capability to protect information |
| | Functionality compliance | The capability to adhere to standards |
| **Reliability** | Maturity | The capability to avoid failure |
| | Fault tolerance | The capability to function in errors |
| | Recoverability | The capability to recover from errors |
| | Reliability compliance | The capability to adhere to standards |
| **Usability** | Understandability | The capability to make user informed |
| | Learnability | The capability to make user learn its application |
| | Operability | The capability to make user to operative |
| | Attractiveness | The capability to be attractive to user |
| | Usability compliance | The capability to adhere to standards |
| **Efficiency** | Time behaviour | The capability to provide fast processing times |
| | Resource utilisation | The capability to use appropriate resources |
| | Efficiency compliance | The capability to adhere to standards |
| **Maintainability** | Analysability | The capability to be diagnosed |
| | Changeability | The capability to be modified |
| | Stability | The capability to avoid unexpected effects |
| | Testability | The capability to be validated |
| | Maintainability compliance | The capability to adhere to standards |
| **Portability** | Adaptability | The capability to use in different environments |
| | Installability | Installation in different environments |
| | Co-existence | The capability to co-exist with other software |
| | Replaceability | The capability to be replace to another software |
| | Portability compliance | The capability to adhere to standards |

ISO 9126-1 (1999) defines the quality in use attributes as the user's view of quality. The attributes of quality in use are categorised into four characteristics: effectiveness, productivity, safety and satisfaction (Figure 8). Quality in use determines the capability of the software product to achieve these specific characteristics. Characteristics of

quality in use measure results of using software, rather than properties of the software itself.

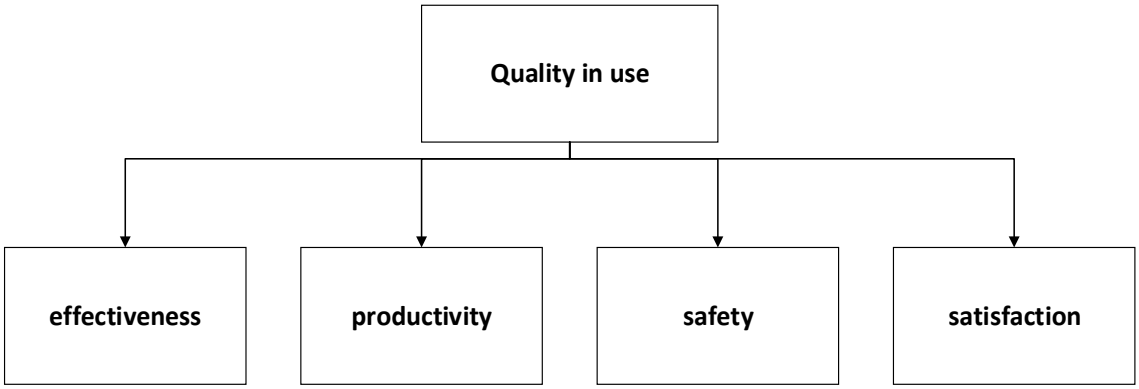


Figure 8. Quality in use (ISO 9126-1, 1999)

ISO 9126-4 (2002) defines the quality in use attributes. Effectiveness metric evaluates the tasks performed by users achieving goals with accuracy and completeness. Effectiveness metric does not evaluate how the goals were achieved, only the extent to which they were achieved. Productivity metric evaluates the resources that users consume in relation to the effectiveness achieved. Safety metrics evaluate the level of risk to people, business, software, property or the environment. Satisfaction metrics evaluate the user's attitude towards the use of the product.

### 2.4.2 ISO 14598

The ISO/IEC 14598 series of standards give methods for measurement, assessment and evaluation of software product quality. ISO/IEC 14598 is intended for use of developers, acquirers and independent evaluators, particularly who are those responsible for software product evaluation. The evaluation results produced by the application of ISO/IEC 14598 can be used by managers and developers/maintainers to measure compliance to requirements and to make improvements where necessary. ISO/IEC 14598 consist six parts: Three processes for evaluation, two for support of evaluation

and a general overview (Figure 9). Each evaluation process can be used in conjunction with parts for evaluation support (ISO 14598-1, 2012).

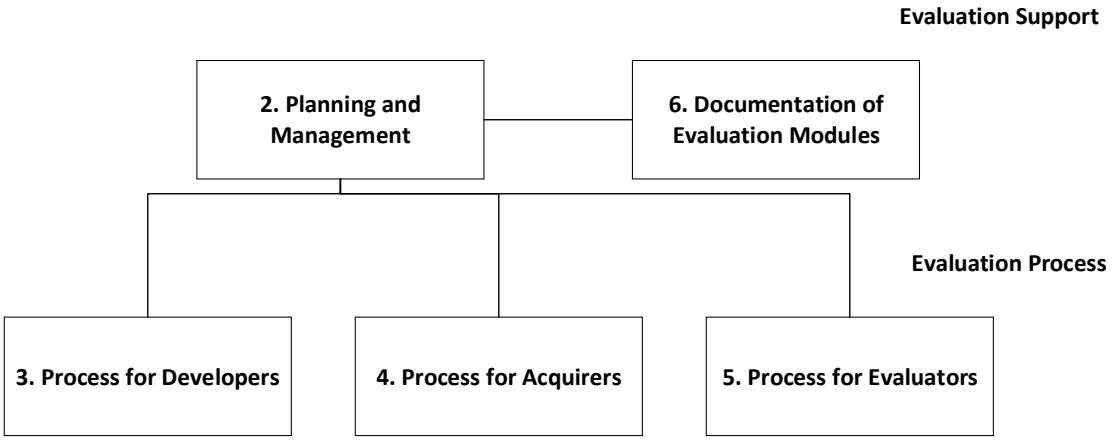


Figure 9. Relationship between evaluation process standards and evaluation support standards (ISO 14598-1, 2012)

**Process for developers**: ISO/IEC 14598-3 should be used by organizations that are planning to develop a new product or upgrading an existing product. Process focuses on indicators that measure intermediate products during the lifecycle (ISO 14598-1, 2012).

**Process for acquirers**: ISO/IEC 14598-4 should be used by organizations that are planning to acquire a new product or upgrading and existing product. Process is used for acceptance or selection of the product (ISO 14598-1, 2012).

**Process for evaluators**: ISO/IEC 14598-5 should be used by evaluators carrying out an independent evaluation of the product. This process is usually performed from the request of a developer or some other party involved (ISO 14598-1, 2012).

**Planning and management**: ISO/IEC 14598-2 Planning and Management contains requirements and guidance for supporting functions for software product evaluation. The support is for planning and managing a software evaluation process and activities included in it. This part of ISO/IEC 14598 is usually used by manager to produce evaluation plan (ISO 14598-1, 2012).

**Evaluation modules**: ISO/IEC 14598-6 provides guidance for documenting evaluation modules. These modules include quality model, the associated data and information about the application. For each evaluation the appropriate module is selected (ISO 14598-1, 2012).

Evaluation process overview of the ISO/IEC 14598 is shown in Figure 10. Processes starts with establishment of the evaluation requirements, then specify, design and finally execute the evaluation (ISO 14598-1, 2012).

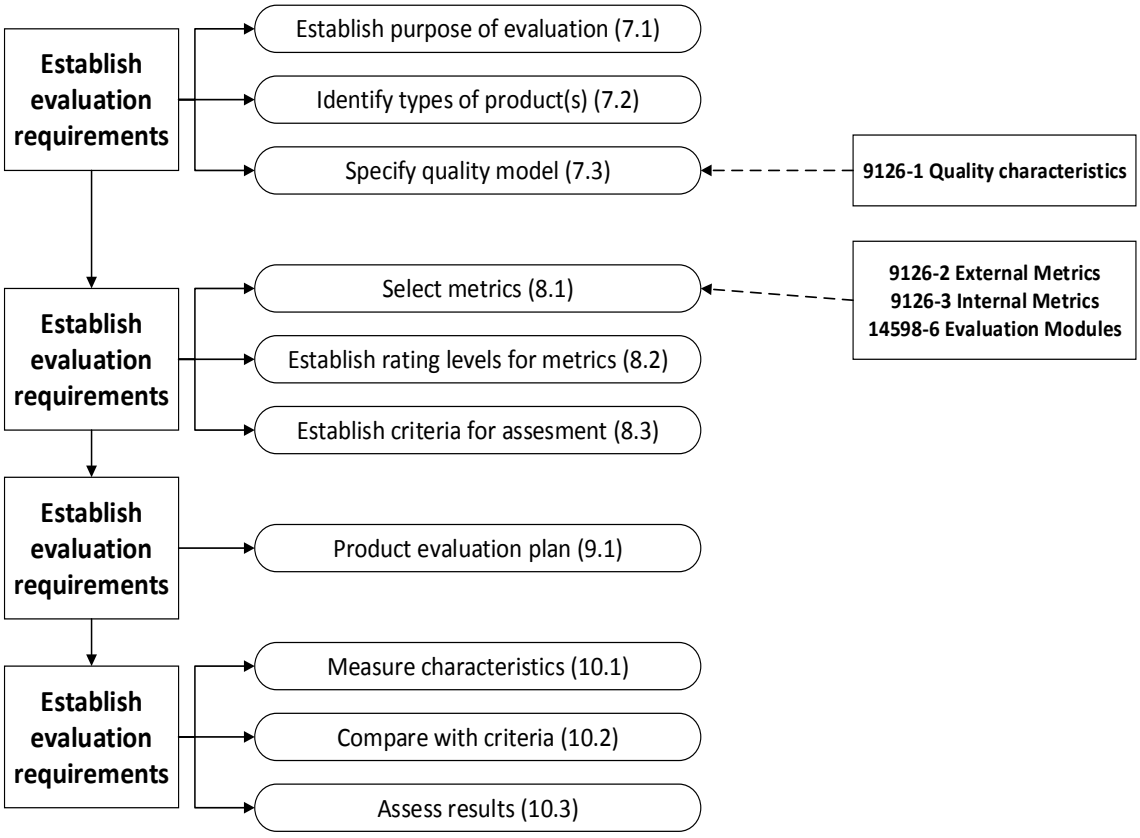


Figure 10. ISO/IEC 14598 Evaluation process (ISO 14598-1, 2012)

### 2.4.3 ISO 25000

ISO 25000 International Standards provides guidance for the use of the new series of International Standards named Software product Quality Requirements and Evaluation (SQuaRE). SQuaRE replaces two related multipart International Standards: ISO/IEC

9126 (Software product quality) and ISO/IEC 14598 (Software product evaluation). SQuaRE is intended for developers, acquirers and evaluators of software (ISO 25000, 2005).

Relationship between ISO/IEC 9126 and ISO/IEC 14598 is shown in Figure 11. ISO/IEC 9126 defines general-purpose quality model, quality characteristics and examples of metrics. ISO/IEC 14598 gives an overview of software product evaluation process and provides guidance and requirements for evaluation. Parts 2 and 6 from ISO/IEC 14598 are for evaluation management and support on corporate level, while other part 3, 4 and 5 provide guidance and requirements for evaluation at the project level (ISO 25000, 2005).

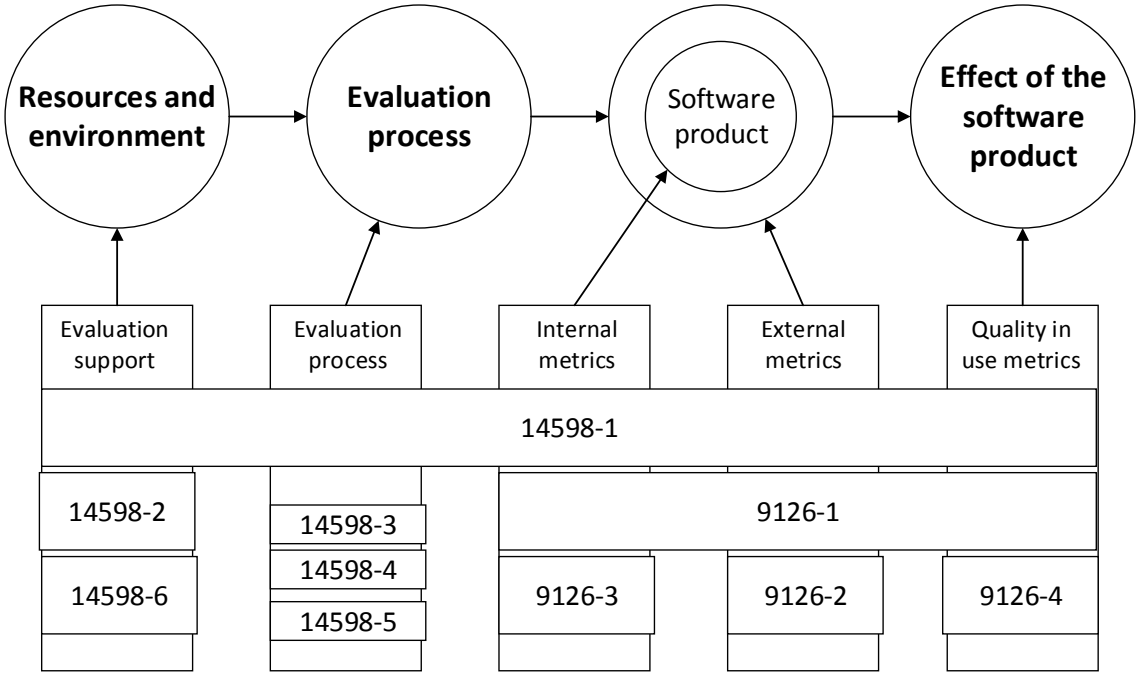


Figure 11. Relationship between ISO/IEC 9126 and ISO/IEC 14598 standards (ISO 25000, 2005)

SQuaRE includes five different families of standards called divisions that are taken from ISO/IEC 9126 and ISO/IEC 14598. In Figure 12 the column titled “Current” lists the standards that are used in the SQuaRE and arrows illustrate how the standards have evolved (ISO 25000, 2005).

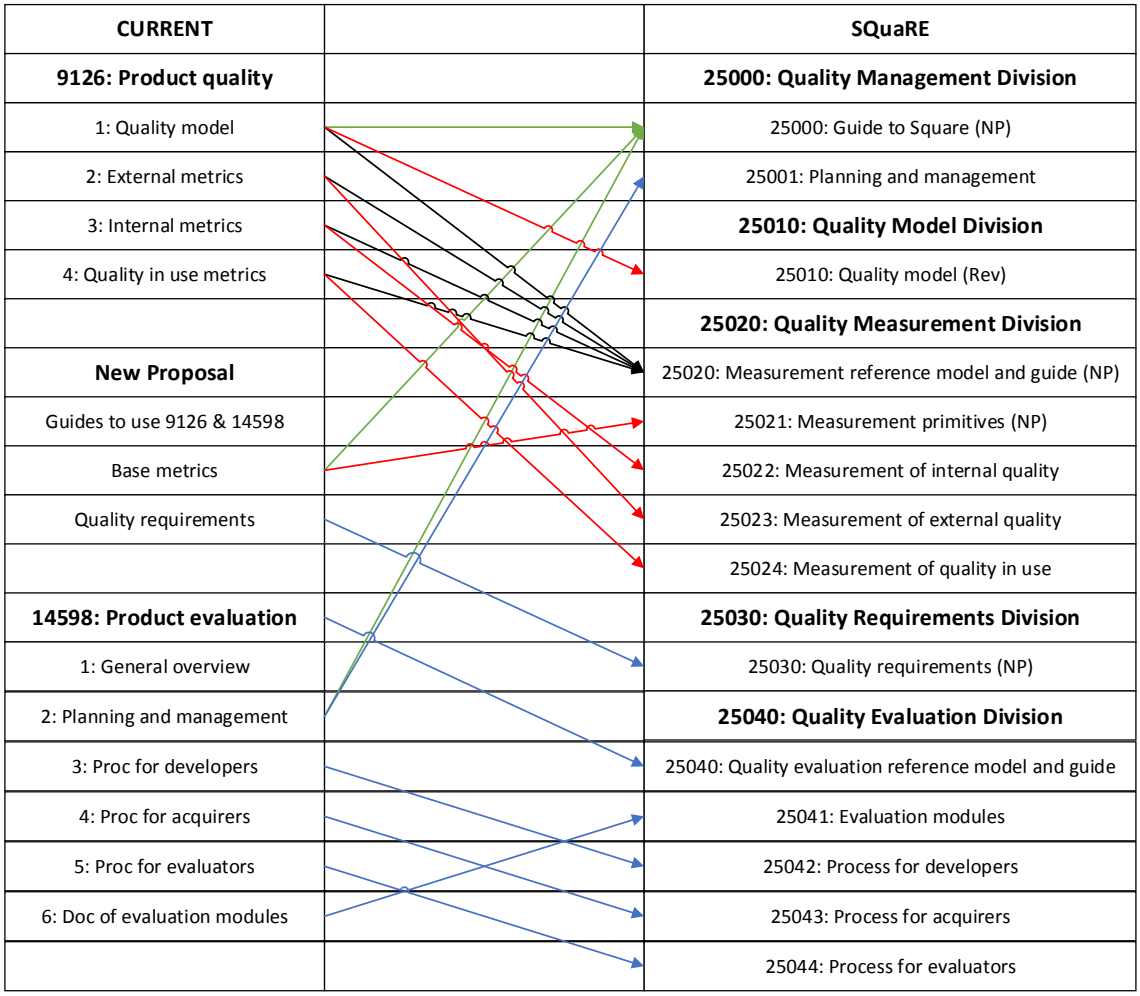


Figure 12. Relationship and transition process between ISO/IEC 9126, ISO/IEC 14598 and SQuaRE series of standards (ISO 25000, 2005).

**ISO/IEC 2500n - Quality Management Division** includes the standards that define common models and terms that are referred further in other standards of the SQuaRE divisions. The division gives high level suggestions how to apply proper standards to specific application cases. The division also provides requirements and guidance for supporting function (ISO 25000, 2005).

**ISO/IEC 2501n - Quality Model Division** includes the standards that present a detailed quality model (internal, external and quality in use). Division also has a practical guidance on the use of the quality model (ISO 25000, 2005).

**ISO/IEC 2502n - Quality Measurement Division** includes the standards with a software quality measurement reference model, mathematical definition of quality

measures, and practical guidance for their application. Measures apply to internal software quality, external software quality and quality in use (ISO 25000, 2005).

**ISO/IEC 2503n - Quality Requirements Division** includes the standards that help specifying quality requirements. Requirements are used in the process of quality requirements elicitation for a software product to be developed or as input for an evaluation process (ISO 25000, 2005).

**ISO/IEC 2504n - Quality Evaluation Division** includes the standards that provide requirements, recommendations and guidelines for software product evaluation. The division also includes support for documentation (ISO 25000, 2005).

Comparable standards for service quality are under development in the ISO.

#### 2.4.4 IEEE 730

The Institute of Electrical and Electronics Engineers Standards Association (IEEE-SA) is an organization within IEEE that develops global standards. IEEE 730 standard helps in determining the content and preparation of software quality assurance plans and also provides a standard how such plans can be prepared and evaluated. This standard is directed toward the development and maintenance of software (IEEE 730, 2002).

IEEE 730 standard for software quality assurance plan (SQAP) consists of 16 sections (Table 6) (IEEE 730, 2002).

Table 6. SQAP sections of IEEE 730 standard (IEEE 730, 2002)

| Section | Name | Definition |
|---|---|---|
| #1 | Purpose | Defines the specific purpose and scope of the particular SQAP. It list the names of software items and intended use of them. |
| #2 | Reference documents | Provides a complete list of documents referenced elsewhere in the text of SQAP. List includes documents that were used in developing the SQAP. |
| #3 | Management | Describes projects organization structure, tasks, roles, and responsibilities. |
| #4 | Documentation | This section includes software requirements, verification and validation plans, user documentation etc. |
| #5 | Standards, practices, conventions and metrics | Identify the standards, practices, conventions, statistical techniques, quality requirements and metrics that will be applied. |
| #6 | Software reviews | Review of software specifications, architecture design, detailed design, verification and validation plan, functional audit, physical audit, in-process audit, management, software configuration management plan and post-implementation. |
| #7 | Test | Identifies all the tests not included in software verification and validation plan. |
| #8 | Problem reporting and corrective action | Describes the practices and procedures to be followed for reporting. |
| #9 | Tools, techniques, and methodologies | Identifies the software tools, techniques, and methods used to support SQA process. |
| #10 | Media control | Identifies the media for intermediate and deliverable computer work and protects computer program physical media from unauthorized access. |
| #11 | Supplier control | Assures that software provided by suppliers meets established requirements. |
| #12 | Records collection, maintenance, and retention. | Identifies the SQA documentation that needs to be retained. |

| | | |
|---|---|---|
| #13 | Training | Identifies the training activities necessary to SQAP. |
| #14 | Risk management | Specifies the methods and procedures to identify, asses, monitor and control areas of risk. |
| #15 | Glossary | Contains a glossary of terms unique to the SQAP |
| #16 | SQAP change procedure and history | Contains the procedures for modifying the SQAP and maintain a history of the changes. |

### 2.4.5 Software Testing

Software testing is important part of the software lifecycle. Shao et al. (2007) defines software testing as a technique that is used for validating the quality of the software. According to Bentley et al. (2005) software testing has three main purposes: verification, validation, and defect finding. The verification process confirms software technical specifications. The validation process confirms that software meets it business requirements. A defect is a variance between the expected and actual result.

A large part of software testing is based on a V-model that was originally developed from the waterfall model. According to Sommerville (2004), the V-model has four main phases: requirements, specification, design and implementation and every phase has corresponding verification and validation testing phases. Bentley et al. (2005) add that the use of the V-model adds testing into the entire software development life cycle. The process of the V-model is described in Figure 13.

According to Bentley et al. (2005) the V-Model of testing identifies five software testing phases: unit testing, system testing, integration testing, user acceptance testing and production verification testing.

**Unit testing** (also called module testing) is performed during the program development phase. Unit testing is much simpler than testing a program/system as a whole. Unit testing makes debugging tasks easier because in a case where an error is found, it is located within the tested unit instead of other units. The testing person should be one who is familiar with the internal details of that unit (Eldon, 1990).

**Integration testing** phase is for seeing are the modules working together correctly without contradicting the system's internal and external specifications (Eldon, 1990).

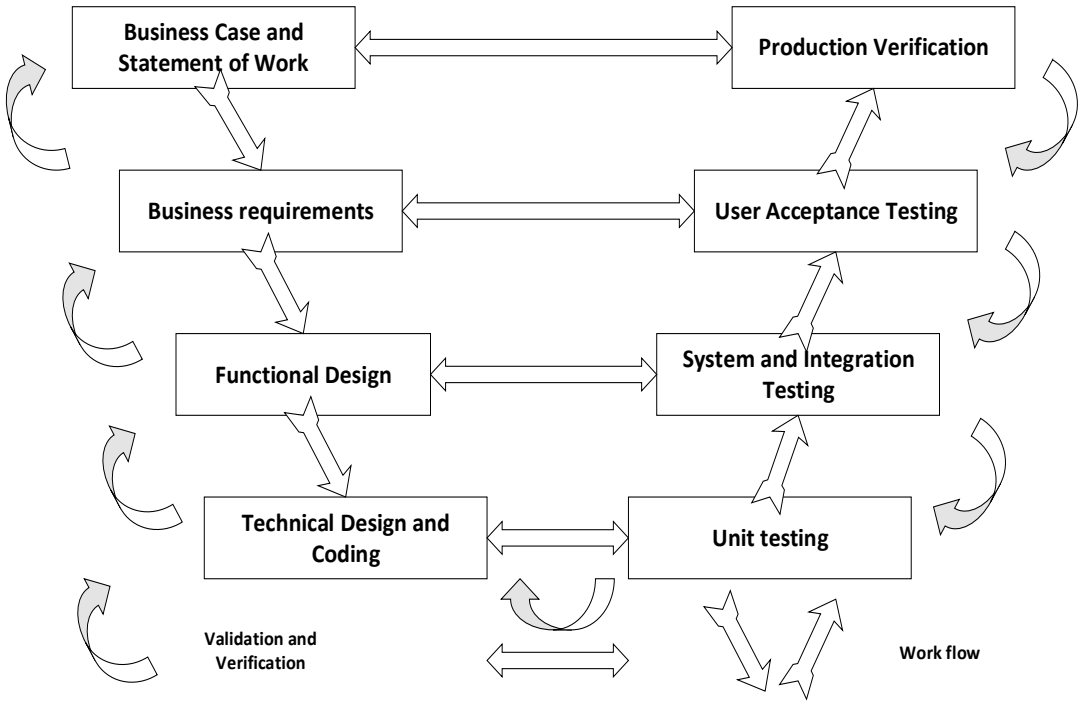


Figure 13. The V-Model of Software Testing (Bentley et al., 2005)

According to Bentley et al. (2005) integration testing examines all the components needed to form a complete system. Integration testing requires involvement with other systems, including systems with outside vendors. Bentley et al. (2005) define **System testing** as phase that tests all the components needed to form the complete application. The system testing tries to minimize interaction with other systems. System testing requires a lot of testing, because it includes feature by feature validation. Eldon (1990) adds that system testing requires the tester to know the functions really well.

**User Acceptance testing (UAT)** is also called beta testing or end-user testing (Bentley et al., 2005). According to Miller & Collins (2001) UAT tests represents the customer's interests. The idea of UAT is to convince customer that features in the application are behaving correctly. Eldon (1990) says that UAT should be performed with system's end-users with assistance of the development group.

Last phase of the software testing is called **Product Verification testing.** Product Verification testing is the final opportunity to test if the software is ready for launch. The application should be removed from the test environment and reinstalled to production environment **(**Bentley et al. 2005).

# 3 METHODOLOGY OF THE RESEARCH

This chapter presents the methodology selected for this research. The selected research method for this study is quantitative research and survey method. According to Rajasekar et al. (2006) research is a logical and systematic search for new and useful information on a particular topic. It is an investigation of finding solutions to scientific and social problems through objective and systematic analysis. It is a search for knowledge, that is, a discovery of hidden truths. The basic and applied researches can be quantitative or qualitative or even both. Sellers (1998) describes qualitative method as an in-depth exploration of what makes people tick on a particular subject: their feelings, perceptions, decision-making processes, etc. and quantitative as methodology that should employ a larger sample which is representative of the entire population being researched.

According to Cooper and Schindler (2003) a research process has the following phases:

- Identification of the problem.
- Defining the research question(s).
- Carrying out exploratory study, to clarify the problem or to refine the research question.
- Developing a plan to carry out the research.
- Developing a plan for data collection.
- Carrying out the data collection with selected method.
- Analyzing the collected data
- Reporting the results.

## 3.1 Quantitative research

Quantitative research is an empirical investigation of social phenomena via statistical, mathematical or computational techniques. The objective of quantitative research is to develop test theories based on phenomena. Quantitative data is in numerical form, like for example, statistics or percentages (Given, 2008). Creswell (1994) describes that quantitative research is explaining phenomena by collecting statistical data and after

analyzed with mathematically based methods. According to Sukamolson (2007) there are four different types of qualitative research: survey research, correlation research, experimental research, and causal-comparative research.

## 3.2 Survey method

Surveys are probably the most commonly-used research methods world-wide. Pfleeger & Kitchenham (2002) describe the survey as a comprehensive system for collecting information to describe attitudes and behavior. Pfleeger & Kitchenham (2002) also divide survey process into 10 different phases:

1. Setting specific, measurable objectives.
2. Planning and scheduling the survey.
3. Ensuring that appropriate resources are available
4. Designing the survey.
5. Preparing the data collection instrument.
6. Validating the instrument.
7. Selecting participants.
8. Administering and scoring the instrument.
9. Analyzing the data.
10. Reporting the results.

According to Taylor-Powell and Hermann (2000) there are five main survey methods: mail, telephone, face-to-face, handout and electronic. Advantages and disadvantages of them are shown in Table 5.

### 3.2.1 Probability sampling

Probability sampling ensures that each and every subject in the population has the same chance of being included in the sample. Pfleeger & Kitchenham (2002) describe four different probability sampling methods: Simple random sample, stratified random sample, systematic sampling, and cluster-based sampling. Simple random sample

Table 5. Advantages and disadvantages of survey methods (Taylor-Powell & Hermann, 2000) & (Pfleeger & Kitchenham, 2002)

| Survey method | Advantages | Disadvantages |
|---|---|---|
| **Mail** | Able to send survey to large number of respondents. Sending mail is also cheap. | Need complete and accurate mailing list. Fear of low response rate. |
| **Telephone** | You get results quickly because many of the target audience have telephones. Survey questions are also clearer to audience with telephone. | Households without telephone are hard to reach. |
| **Face-to-face** | Validity and reliability of data is better. | Time consuming and costly. |
| **Handout** | Respondents are available easily and you get immediate feedback from them. Handouts are also cheap way of conducting survey. | You may not be able to reach the people again. |
| **Electronic** | Large audience is easy to reach. | Low response rate. |

method is where every member of the target population has the same probability of being included in the sample. Stratified random sample divides the target population to subgroups and different subgroups are expected to answer differently to the questions. Systematic sampling selects every nth member from the population list. Cluster-based sampling is the term given to surveying individuals that belong to defined groups (Pfleeger & Kitchenham, 2002).

### 3.2.2 Reliability and validity of survey

Reliability and validity of the data are important factors when conducting a survey. Reliability is defined as the extent to which a questionnaire, test, observation or any measurement procedure produces the same results on repeated trials. Validity is defined as the extent to which the instrument measures what it purports to measure (Miller, 2013). Pfleeger & Kitchenham (2002) define several aspects that are needed to be taken in consideration when measuring validity of the survey. These aspects are face validity, content validity, criterion validity and construct validity. Face validity is a cursory review of items by untrained judges. Content validity checks how appropriate the instrument seems to a group of reviewers. Criterion validity measures how instrument compares with another similar instrument. Construct validity examines how an instrument behaves when it is being used (Pfleeger & Kitchenham, 2002).

## 3.3 Methods for analysis part

The survey includes both multiple choice questions and open-ended questions that need to be analyzed in different ways.

### 3.3.1 Multiple choice questions

The results from the multiple choice questions are presented in pie or bar chart depending on the analyzed question. Pie and bar charts are a graphical data analysis technique for summarizing distributional information of a variable. Multiple choice questions are also analyzed with Pearson correlation coefficient. According to Callaghan (1996) a correlation coefficient measures the strength and direction of a linear association between two variables. It ranges from -1 to +1. The closer the absolute value is to 1, the stronger the relationship. A correlation of zero indicates that there is no linear relationship between the variables. The coefficient can be either negative or positive.

### 3.3.2 Open-ended questions

The results from open-ended questions are analyzed with six step guide (Cerritos College, 2013).

1. Read carefully the responses.
2. Develop coding categories.
3. Label each response with one or more coding categories.
4. Look at what you have and do sub-coding.
5. Think about what the responses mean and identify the patterns and trends.
6. Write up the analysis.

# 4 EMPIRICAL DATA ANALYSIS

This chapter presents an empirical data analysis of the completed survey. The chapter includes background of the research, which introduces the population of the study, limitations of the research, data collection method, data collection time, respondents of the survey and industry sectors of the companies. The empirical data analysis has been divided into three major parts, which are further divided into smaller detailed sections of related topic. The first part includes software development life cycles and used development methods in South Korean companies. Second part discusses software quality assurance (SQA), with used standards and testing methods in South Korean companies. The last part of the data analysis examines the current and future state of the South Korean software industry.

## 4.1 South Korean software industry

According to Park (2001) digital economy has emerged and South Korea has transitioned to the knowledge –based economy. OSEC report (2011) says that South Korean ICT can be divided into three different sectors. These areas are ICT equipment, telecommunications services and software industries. The biggest sector is ICT equipment in terms of production that is more than two thirds of the ICT industry. The wired and wireless telecommunications are the most important sector of industry in today's knowledge based society of South Korea. About 30 to 40% of South Korea's total GDP growth is contributed by the ICT industry.

According to OSEC report (2011), the market focus started to move from hardware to services and solution and the share of the market for software is expected to rise to 39% by 2015. This makes software the fastest-growing sector of the IT market in South Korea. South Korean software market trends to application packages for other industries such as car, pharmaceutical, financial services and healthcare. South Korea is also putting a lot of effort to the future of software industry with the IT Future Vision 2020 plan. According to eGov Innovation (2012), "South Korea's Ministry of Strategy and Finance said the government is expanding financial assistance for new growth engines

from 3.4 trillion won in 2011 to 3.9 trillion won in 2012, focusing primarily on strengthening the structure of the software industry".

The problem in Korean software development is lack of talented people. According to Korea Times news (2011), "Korea lacks competitiveness in software as it lacks qualified people…as software is applied in diverse industries such as electronics, automobiles, and medicine, software human resources are key to enhance longevity in the industries," said Ryu Ji-Seong, a senior fellow at the Samsung Economic Research Institute, in a report.

## 4.2 Background of the research

The study was started in January 2013 with literature review of the research topic. The literature review included software life cycle methods, software quality assurance and material about the South Korean software industry. Research documents were gathered from scholar.google.com and supervisor D.Sc. Ossi Taipale, who provided reading material about the research topic. The literature review ended in March 2013 and based on the gathered information the research questions were defined. March 2013 was used for preparation of the survey and from April 2013 to June 2013 for data collection period. Data analysis was done from July 2013 to August 2013. End of the year 2013 was used for writing the results and rest of the missing theory part.

The purpose of this study is to explore South Korean ICT industry. Collected data should answer to the research questions and bring results for STX-project. The objective of the literature review was to increase knowledge in the research area and to produce suitable questions to the survey.

### 4.2.1 Sample of the research

The sample of the study consists of 34 South Korean companies that develop software. Interviewed people were asked the amount of employees in their company and the results were divided into four different categories. The results show (Figure 14.) the sizes of companies in the study. The largest segment consists of companies between 10

and 50 people (53%). The second largest segment represents companies between 50 and 500 people (27%). The last categories consists of organizations under 10 people with 13% and organizations over 500 people with 7%. The median of employee size is 25. It shows that the companies that took part in this study, are mostly small and medium sized companies (SME's) (European Commission, 2005).

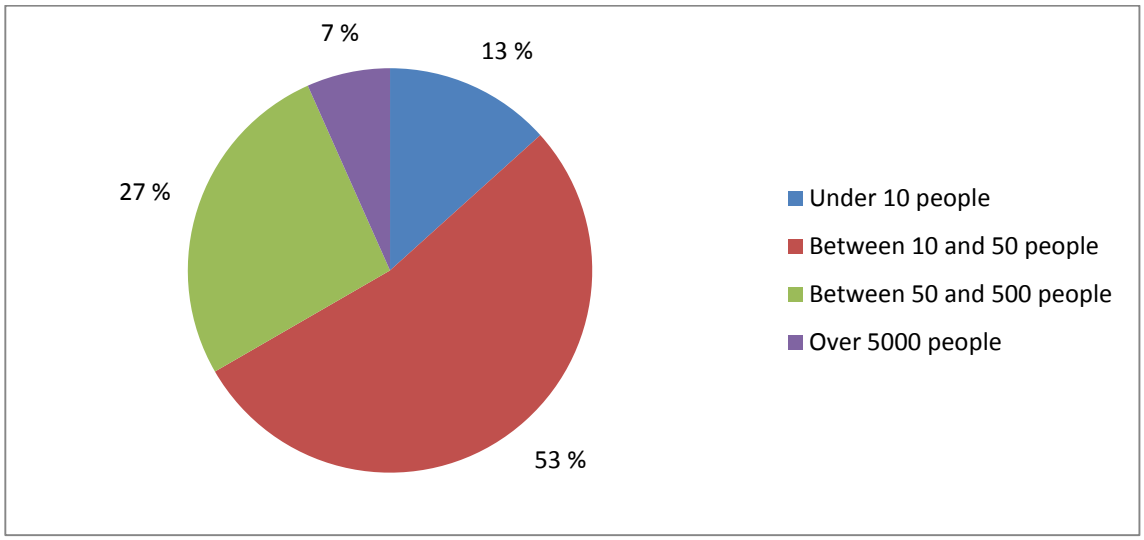


Figure 14. Size categories of the companies

### 4.2.2 Data collection method

The used data collection method in this study was an online survey. Other possibilities included telephone or in-person interviews. Telephone interview was not used because of the language barrier between interviewer and interviewee. The Korean language skill of the interviewer was not on a high level enough to keep interviews and some of the interviewees' English language level was not high enough to understand the questions correctly. Also in-person interviews were abandoned because the above reason and the lack of planned data collection time.

Online survey was implemented with an online software called Webropol (Webropol, 2013) that is an Internet-based application. It allows creating questionnaires, compiling the answers and reporting the results. The survey included 23 questions in total. Questions were divided into: 1 general information, 4 software life cycle questions, 8

agile methods questions, 5 software quality questions, 2 software testing questions, 2 South Korean software industry questions and 1 open ended question at the end. Questions in the online survey were available both in English and Korean languages, which helped respondents to understand the questions more specifically.

Probabilistic sampling was used in selecting the sample from population. Collection of potential and suitable respondents started by using Google as the search engine for South Korean companies websites. Candidates of the sample were selected based on knowledge gained from their webpages. Websites of the companies were mainly in Korean language, which gave little problems at the beginning. After acquiring email addresses from companies websites Webropol was used to send the survey link to their emails. Approximately 300 randomly selected emails were sent to South Korean companies, which resulted only to 8 answers (2.6% answer rate). After disappointment of sending the survey by email, the study continued using telephone calling. A hired person with Korean and English language proficiency called South Korean companies and acquired personal emails. Approximately 200 phone calls were made and it resulted to extra 26 answers (13% answer rate).

#### 4.2.3 Data collection time

The data collection was conducted between 8$^{th}$ April 2013 – 15$^{th}$ June 2013.

#### 4.2.4 Interviewees

People who took part in this study worked in software development units in South Korea. Interviewees are divided into three different groups based on their job description: Executive-level that contains chief executive officers (CEOs), chief technical officers (CTOs) and manager level employees, the second group project leaders contains project managers and the third group developers consists of software developers.

The results show (Figure 15) that the majority of respondents belong to software developers (48%). From the rest of interviewees, 28% belong to project leaders and 24% to executive-level employees.

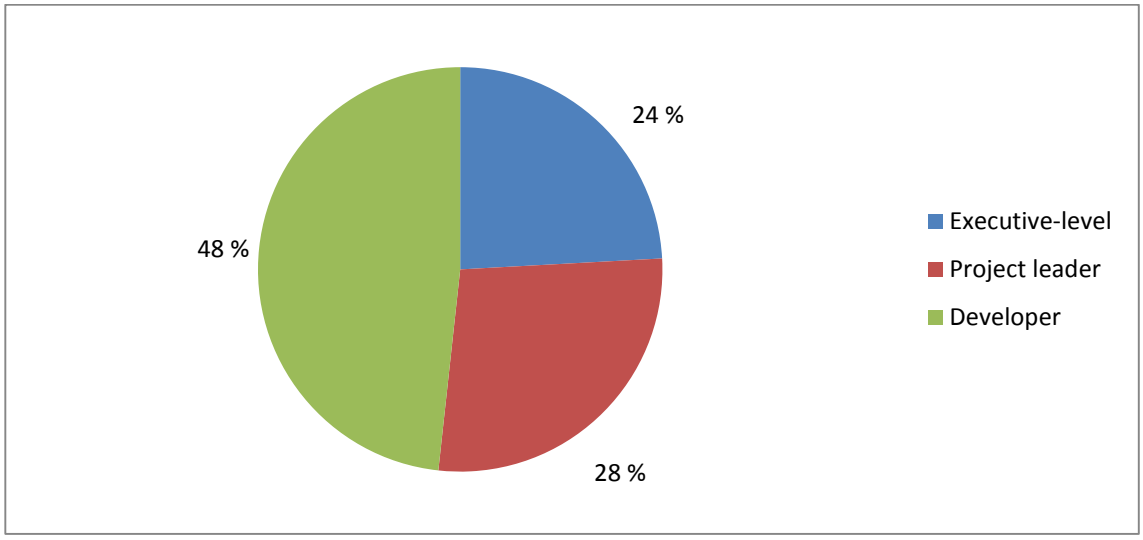


Figure 15. Roles of the interviewees

### 4.2.5 Industry sector

Figure 16. shows the industry sectors of the companies that responded to the survey. Overwhelming majority of the companies (80%) represents the IT industry. Other industries include manufacturing (7%), communication (7%), bioinformatics (3%) and civil engineering (3%). IT being the most dominant industry sector is an optimal result, because it is more likely that companies related to IT sector are familiar with software life cycle models and SQA.

Figure 16. also shows IT industry further divided into more accurate segments. The largest segment inside IT included companies of software development (63%), mobile games (14%), machine vision (7%), information security (3%), system integration (SI) (3%) and internet service (3%).

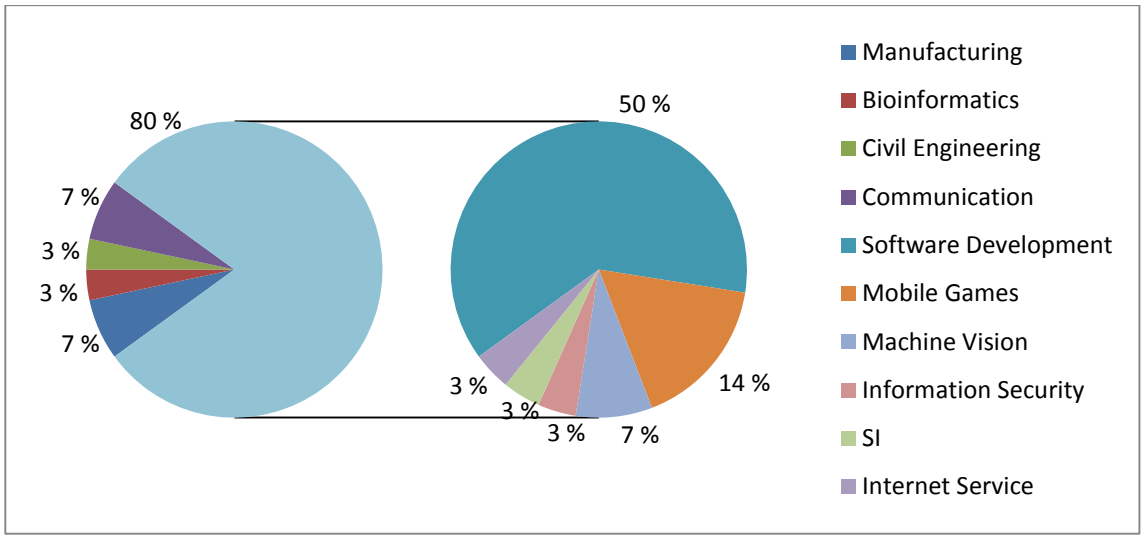


Figure 16. Industry sectors of the companies

## 4.3 Software life cycle models and methods in South Korea

The first part of the research consists of information about the usage of software life cycle models and methods used in South Korean software industry. The respondents were asked what software life cycle methods they are using in projects. The respondents were also asked about customer participation in the software life cycle and do they have their own quality assurance department and how big it is.

### 4.3.1 Software development method

Looking deeper into software life cycle the survey reveals which specific models or methods are most popular amongst South Korean software companies. The respondents were asked what software development methods the company is currently using. In a situation where company was using two or more different methods at the same time, it was possible to give multiple answers.

The results (Figure 17) show that the use of traditional and agile methods is almost equal. Agile methods gathered 46% and traditional methods 42% of respondents' answers. The minority applied homemade methods (12%) that were not recognized as traditional or agile method, but were combinations of two or more known methods.

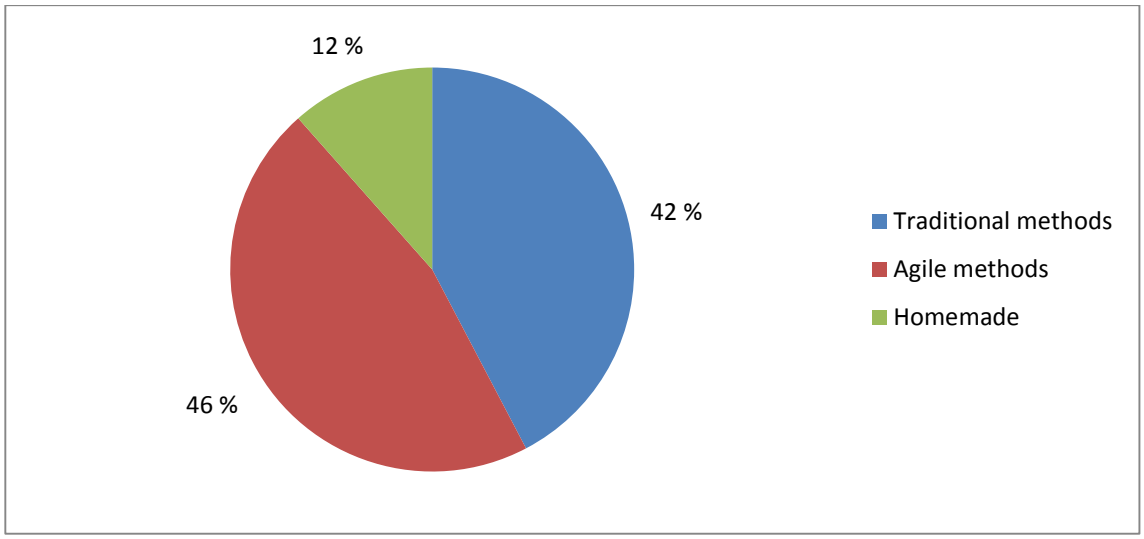


Figure 17. Traditional vs. Agile methods

Traditional and agile methods consist of different known methods, figure 18 shows most popular models and methods. Most popular software life cycle model is Waterfall (31%) and two agile methods Scrum (17%) and Extreme Programming (14%) are the most popular methods. Other used methods and models were Feature-Driven Development (FDD) (12%), homemade methods (12%), Spiral model (6%), Rational Unified Process (RUP) (6%) and Lean software development (LSD) (2%).

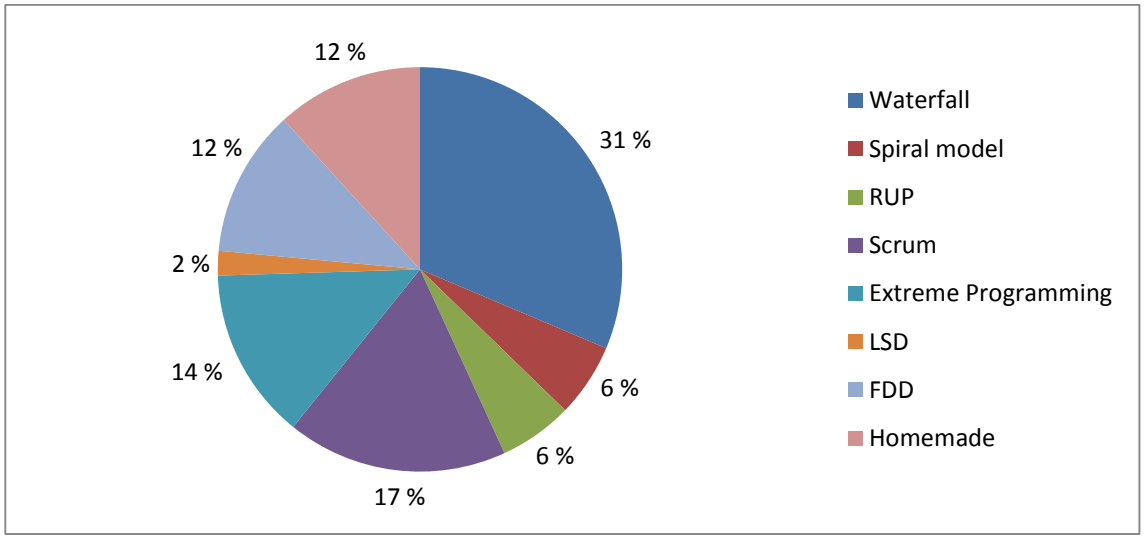


Figure 18. Popular life cycle models and software development methods

Usually traditional methods are known for being used in bigger companies and projects with more employees, while agile methods favor in smaller work groups and projects. A hybrid model is a combination of two or more methods. You can, for example, use Waterfall and use iterative aspect of Scrum in it and make it more suitable for your project. Figure 19 shows what kind of methods companies are using versus their company size. Companies were divided into three different categories, small company under 50 employees, mid-size company from 50 to 100 employees, and big company over 100 employees. The results show that in small companies the use of software development methods varies a little bit, 42% are using hybrid, 32% are using only traditional methods, and 26% are using only agile methods. In medium sized companies only traditional methods are used 71%, hybrids 29%, while use of agile methods is 0%. In large companies 50% are using hybrids and 50% are using agile methods.

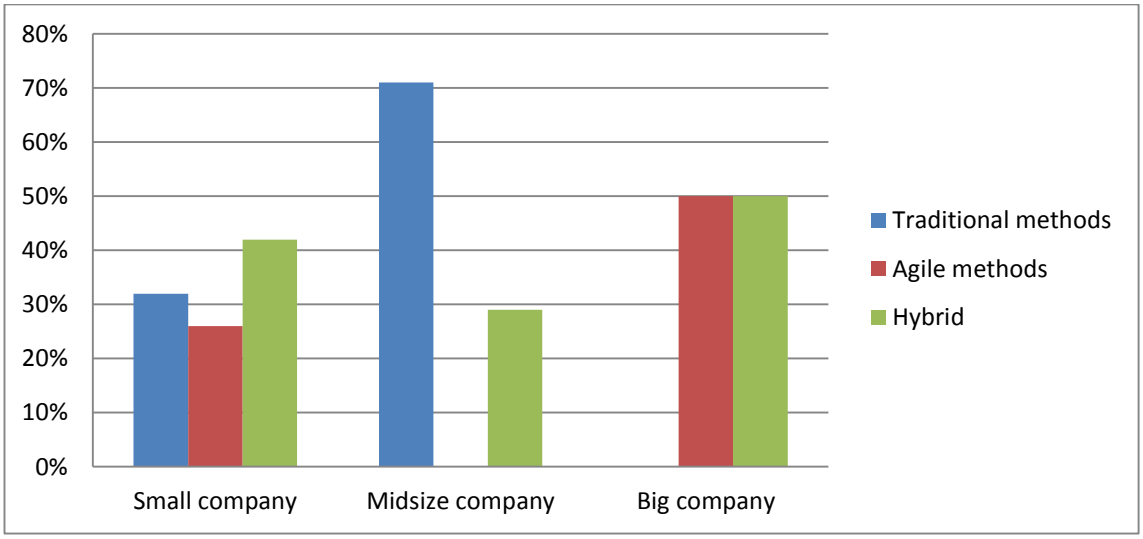


Figure 19. Used software life cycle method compared to company size

The results of the software development methods shows that South Korean software industry is starting to adapt agile methods. Almost half of the companies interviewed are using agile methods in some way and that shows big interest in them. Especially the amount of hybrid models that combines the use of traditional and agile methods is popular amongst South Korean software development. Another interesting result is that the use of traditional methods is still notable. This shows that South Korean software industry still relies on traditional way of leading software life cycle. Especially small

and midsized South Korean companies are using Waterfall method a lot, although use of agile methods could be more suitable to them based on their company's size.

### 4.3.2 Customer participation

Customer participation is important part of the software development. Questions measured how much customers participate in software development life cycle. The results (Figure 20.) show that answers were divided equally. Most answers got once a week (32%), and a couple of meetings during the life cycle (29%). Other categories were meetings once a month (27%) and daily (12%).

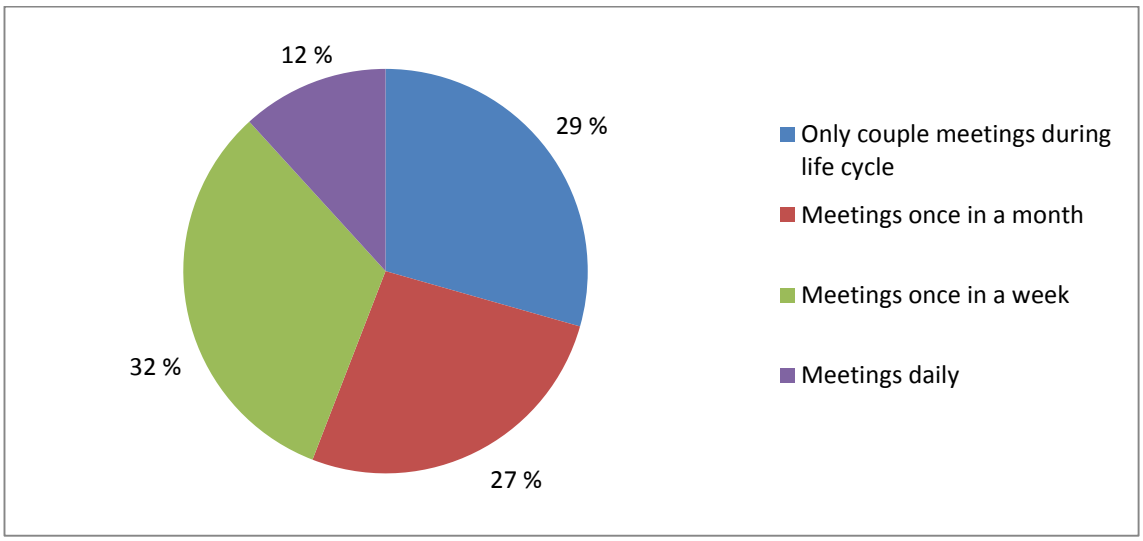


Figure 20. Customer participation

Traditional methods usually rely more on documentation and do not have so many meetings with customers during the life cycle, whereas agile methods encourages user to have meetings on almost weekly basis. Figure 21 shows how often participating companies organize meetings with customers versus the software life cycle method they are using.

The results show that meetings when using agile methods are arranged weekly and monthly that is one of the important features of agile methods. When using traditional

methods meetings are mostly arranged only a couple of times during the life cycle or on weekly basis. These results are interesting because usually when using traditional methods meetings with customer are not that important feature compared when using

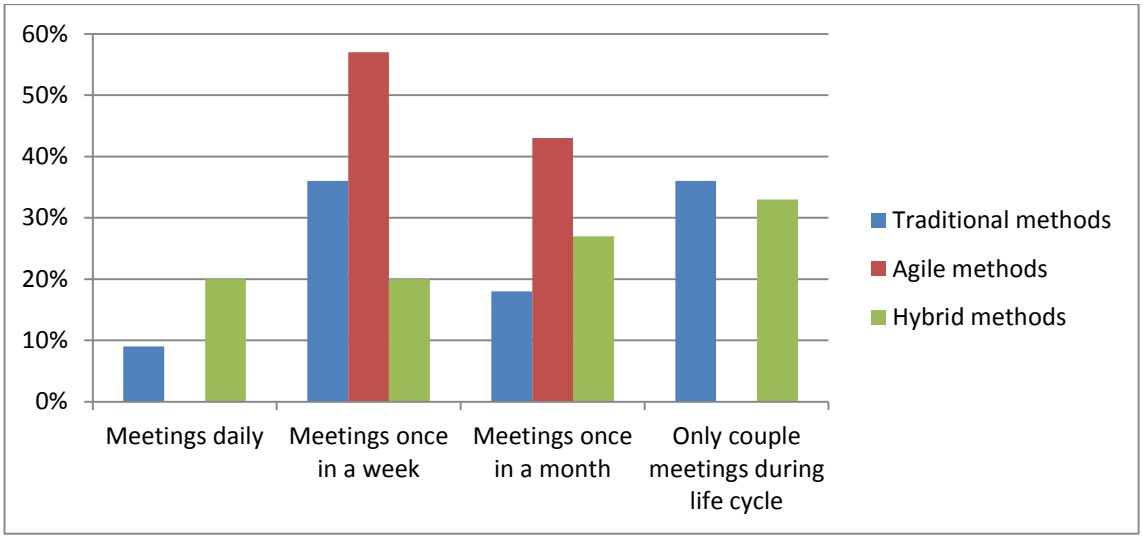


Figure 21. Meeting customers and software development method

agile methods, but still there are a lot of companies that are using only traditional methods and arranging meeting with customer on weekly basis. It seems that some companies adapt different features of agile methods or traditional methods to their software development method. Some companies are having meetings really often, while some are still following the traditional style of having only a couple of meetings with customer during the life cycle.

### 4.3.3 Quality assurance unit

Concerning software quality assurance it was asked if the respondent's company has their own quality assurance department. The results show (Figure 22) that 53% of companies don't have and 47% of companies have their own quality assurance department.

Figure 23 compares company's size and the existence of the quality assurance unit. The results show that the majority of the small and medium sized companies don't have

their own quality assurance department, but all the large companies of this study have their own quality assurance department. This shows that smaller companies are doing their quality assurance inside their development team or they outsource it from other companies.

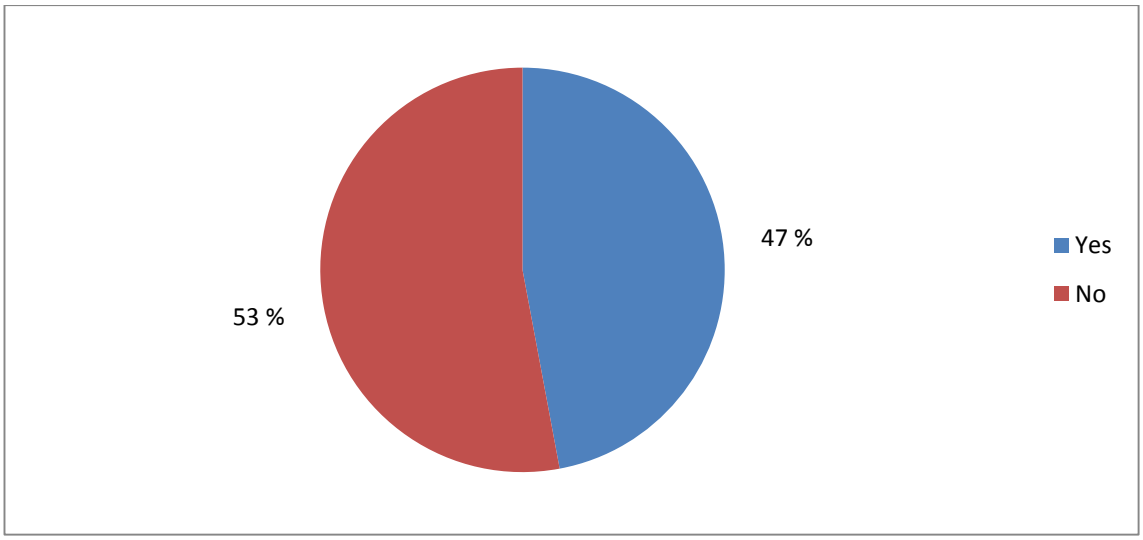


Figure 22. Quality assurance unit

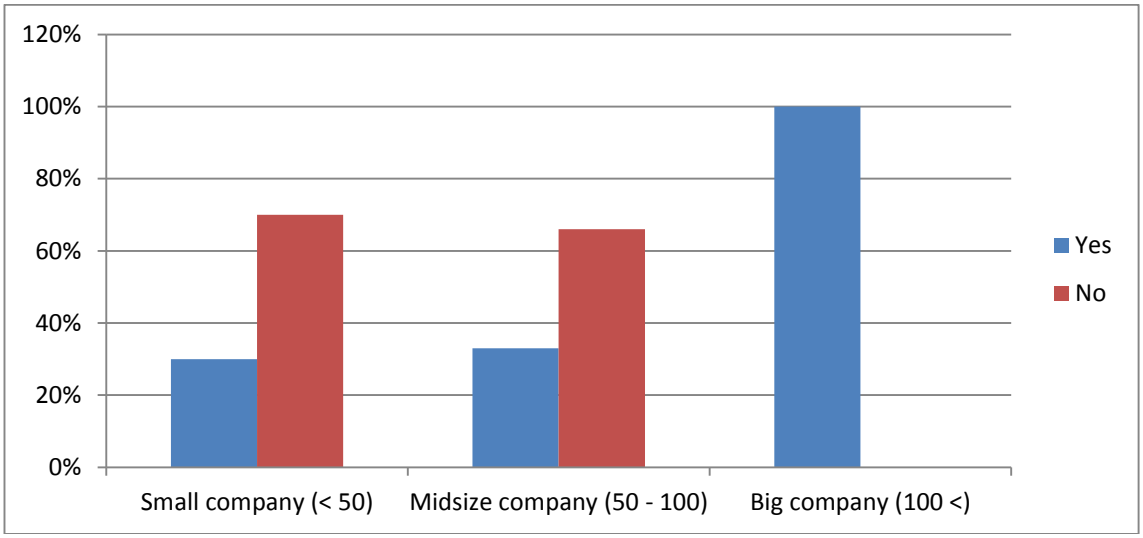


Figure 23. Existence of the quality assurance department compared to company size

The respondents were asked how many percent of their employees are assigned to quality assurance unit (Figure 24). Respondents were asked how many percent is the quality assurance unit compared to the entire software development department. The results were divided into four groups, under 25%, between 25% and 50%, between 50%

and 75%, and over 75%. Most respondents answered that they have from 25% to 50% of their employees assigned to the quality assurance unit. Respondents with fewer than 25% of their employees assigned to quality assurance unit were 43%. Least answers got quality assurance units from 50% to 75% (7%) and over 75% with (0%).

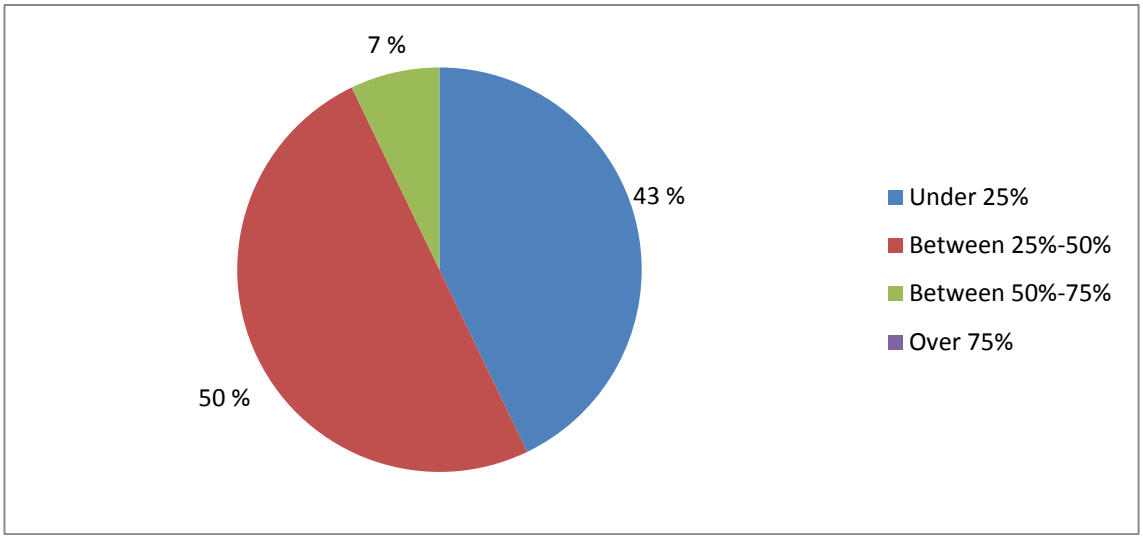


Figure 24. Employees assigned to the quality assurance unit compared to the entire software development department.

## 4.4 Agile methods in South Korea

The first questions explore experience and knowledge of agile methods in South Korean software industry. Second part explains effects of agile methods on productivity, quality, costs, and satisfaction of customers in projects. The purpose of the questions is to find out if agile methods have positive or negative effects on software development.

### 4.4.1 Experience with agile methods

The respondents were asked their experience with agile methods. Out of 34 respondents 16 stated they use agile methods. Figure 25 shows that results are being scattered almost equally; amount of respondents who have more than 2 years of experience with agile methods was 38%, closely following by the respondents who have experience from 6

months to 2 years (37%) and the respondents who have experience less than 6 months (25%).

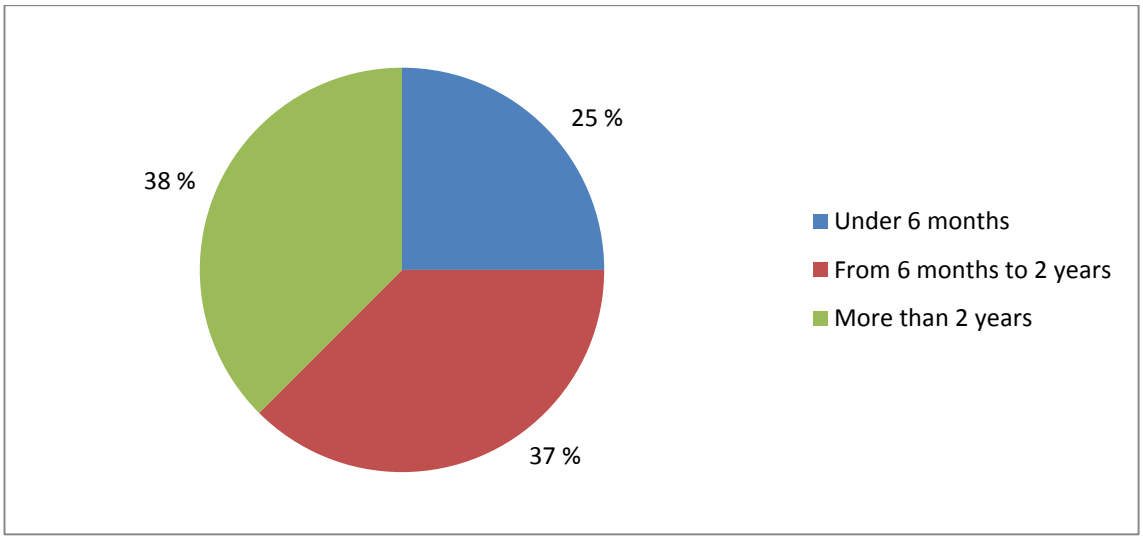


Figure 25. Experience with agile methods

Respondents were also asked about their knowledge of agile methods (Figure 26). Respondents estimated that their knowledge is average (47%) and 23% described it as low. Rest of the answers were high (18%) and very high (12%). Nobody described their knowledge as very low.

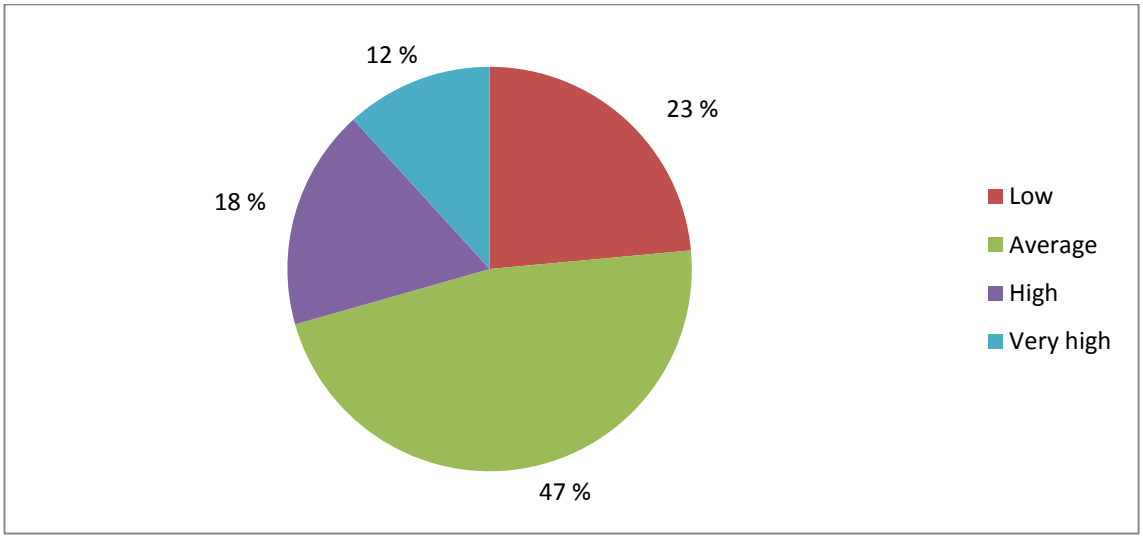


Figure 26. Estimated knowledge of agile methods

The results show that South Korean software industry has a good amount of experience using agile methods. Over 75% of respondents have used agile methods more than 6 months and that indicates lot of interest to use it. On the other hand, almost half (47%) of the respondents described their knowledge of agile methods only average and 23% described it as low. This shows that even though respondents have been using agile methods for a rather long time they still don't feel that they know much about it.

### 4.4.2 Effects of using agile methods

Questions also focused on how the use of agile methods affects the software life cycle. The focus was on four aspects in the software life cycle: productivity, quality, costs and customer satisfaction. The results show (Figure 27) that 62% of respondents answered that productivity was better than before when using agile methods, 19% thought that productivity was unchanged, 13% estimated that productivity was much better and 6% that productivity had went down a little.

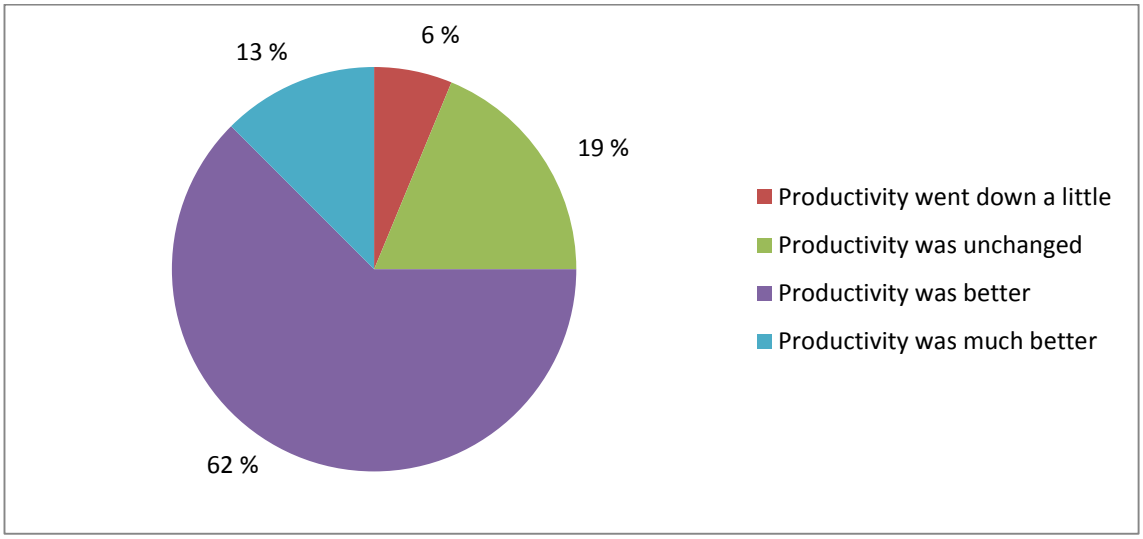


Figure 27. Effects on productivity

Effects on quality followed the same pattern (Figure 28) as productivity. Majority (81%) of the respondents thought that quality was better and 13% that quality was much

better than before, 6% estimated that quality did not change. Notable is that none of the respondents answered that quality went down after using agile methods.

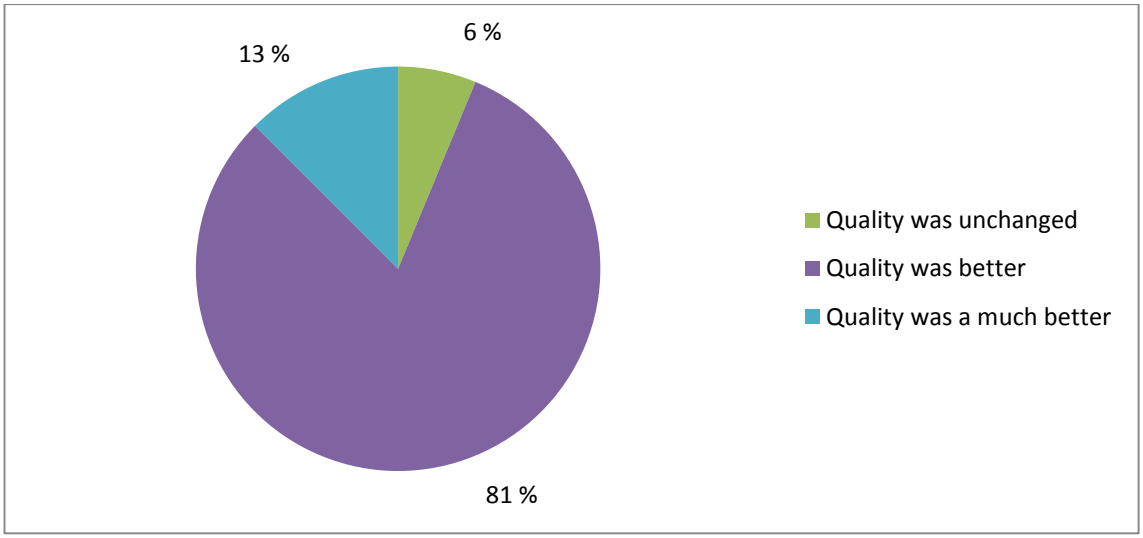


Figure 28. Effects on quality

When asking (Figure 29) about the costs of using agile methods, 50% of the respondents thought that costs were unchanged, 29% that costs were a little lower, 14% that costs were a little higher and 7% that costs were a lot lower.

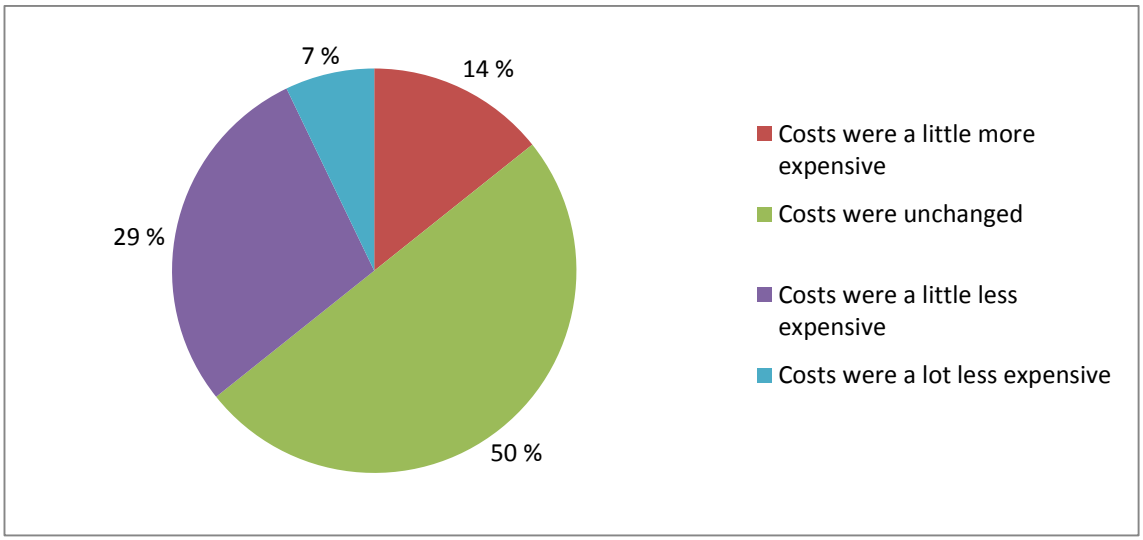


Figure 29. Effects on costs

Satisfaction of customers when using agile methods is described in Figure 30. Majority (75%) of the respondents thought that satisfaction was better, 13% that satisfaction was much better and 12% that satisfaction hasn't changed. Notable is that none of the respondents thought that satisfaction of customers has went down when using agile methods.

Overall the use of agile methods within South Korean software industry has remarkable positive effect. Majority of the respondents thought that productivity, quality and satisfaction of customer were better than before. This shows that agile methods can make the software life cycle better when they are used correctly and in the right projects. Only costs of agile methods were not much lower. Half of the respondents thought that costs of their projects didn't change after adaption of agile methods, but almost 30% thought that costs were a little lower. This indicates that costs may also be reduced with agile methods.

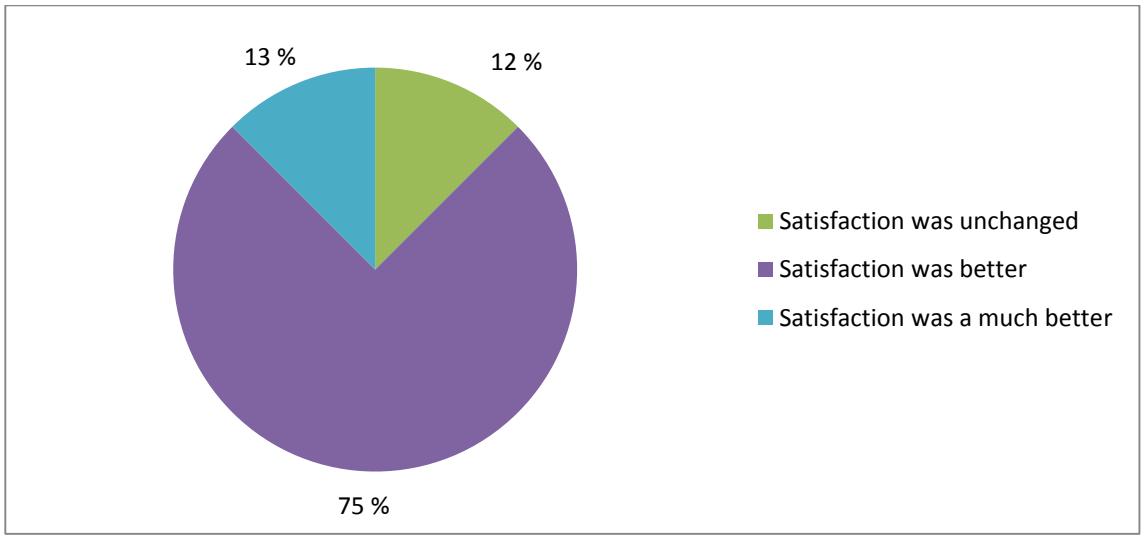


Figure 30. Effects on customer satisfaction

### 4.4.3 Features of agile methods

The questions explored also favorite features of agile methods. Majority of the respondents estimated (Figure 31) that change is more important than following a plan (44%), the second favorite feature was that working software is more important than the

comprehensive documentation (31%). Last two favorite features said that customer collaboration is more important than contract negotiation (13%), and individuals and interactions are more important than processes and tools (12%).

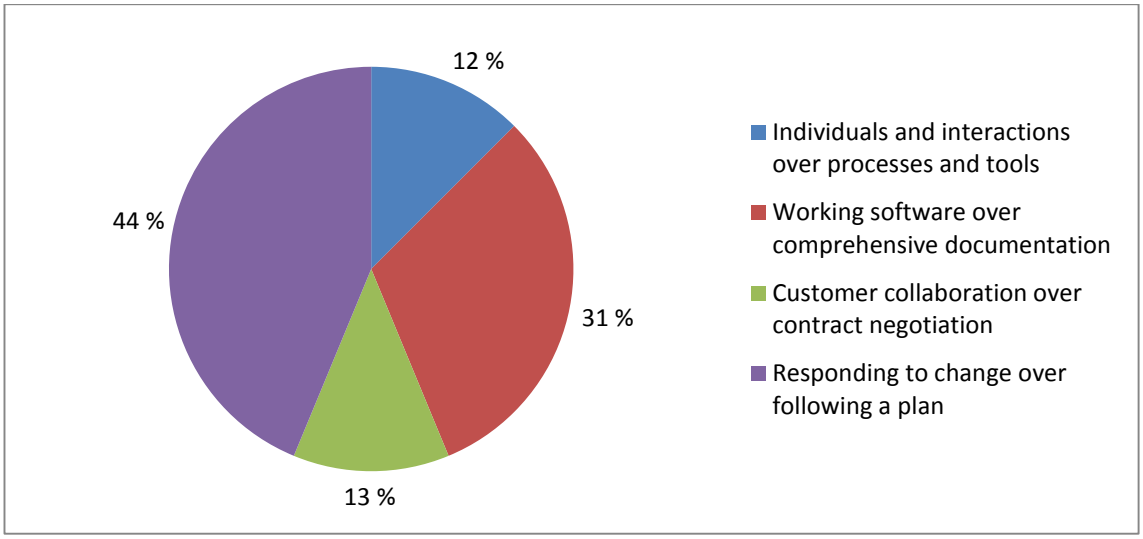


Figure 31. Features of agile methods

### 4.4.4 Advantages and disadvantages of agile methods

The respondents were asked what they think about advantages and disadvantages of agile methods and are they planning to use agile methods in future. Most of the respondents thought that the biggest advantage with agile methods is that requirements are easier to change during product development cycle compared to traditional methods. Another popular advantage was the rapid development style and possibility for quicker prototypes. Other advantages included better quality, better communication and better management of human resources.

The respondents thought that disadvantages of agile methods include experience of users, too rapid development speed and complexity of the method in really simple projects. Agile methods need experienced users and that's why the respondents thought that it is important to have people with knowledge of agile methods before running it. Also rapid development speed is not good for inexperienced workers, quick iterations might get workers exhausted and that will effect immediately to the project.

Respondents also mentioned that agile methods might not be suitable for really simple projects, like for example web-development.

When asking about the future with agile methods, every respondent answered that they will continue their projects with agile methods. This shows that even though running projects with agile methods might be more difficult, respondents still find more advantages than disadvantages in it and they feel comfortable to use them in the future.

The respondents were also asked how the use of agile methods affects testing and quality activities. Most of the respondents thought that testing becomes more stable and it is easier to find bugs. The biggest reason for this is the possibility to collaborate more with customers. The respondents thought that because of this feature, testing and developing becomes faster and easier when communication with customers is more frequent. This results to better quality and prevents misunderstandings between involved companies. Few of the respondents mentioned that using agile methods did not have any effect on their testing or quality activities. One respondent said that there might be problems if same employees are doing developing and testing at the same time, so they should be separated

## 4.5 Software quality assurance in South Korea

This part of the empirical analysis includes the study of software quality assurance in South Korea. The respondents were asked about their software quality assurance plans and software quality standards they are using during their software development projects.

### 4.5.1 Software quality assurance plans

Respondents were asked if their software department has any quality assurance plan. The results (Figure 32) show that 55% of the respondents are using some kind of quality assurance plan during their software development. The rest of the respondents (45%) don’t use any quality assurance plan.

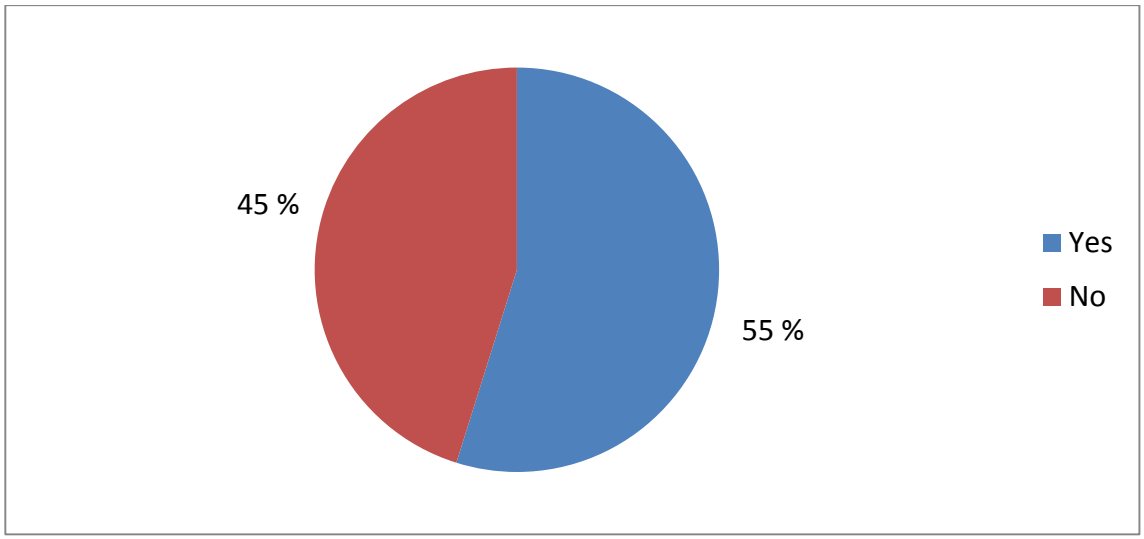


Figure 32. Quality assurance plan

Figure 33 shows which software quality assurance standards the respondents are using during their software life cycle. Most respondents used homemade standards (47%) that are developed within the company and 32% did not use any standards. Known quality assurance standards like the ISO series and the IEEE were not popular amongst South Korean companies. ISO 9126 gathered 14% while ISO 14598, ISO 25000 and IEEE gathered 0%. 7% of the respondents used other standards like defense standards.

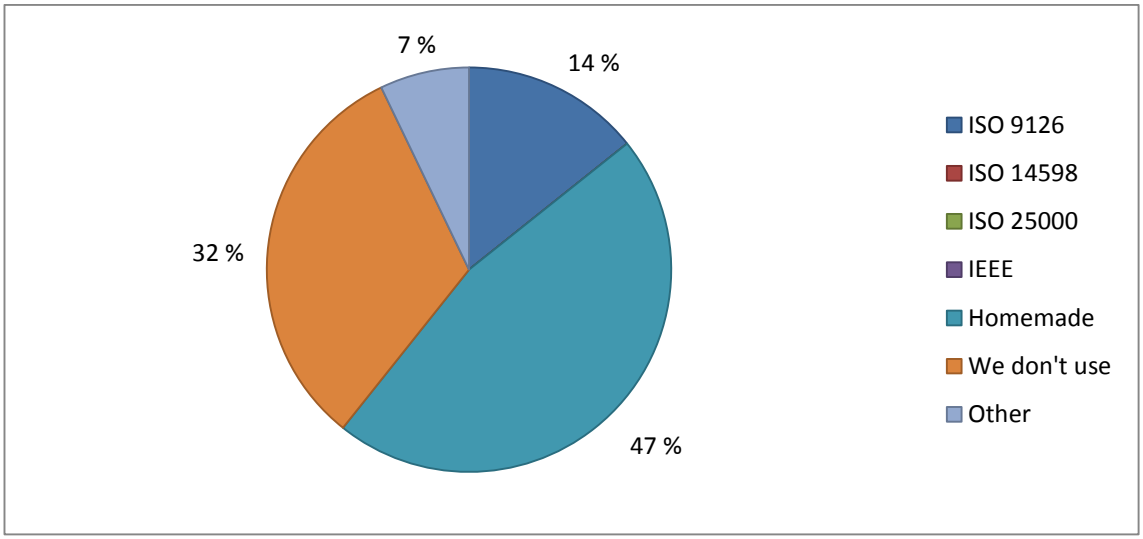


Figure 33. Quality assurance standards

The respondents were also asked which they estimate as the most important software quality characteristics. The possible answer options were taken from the ISO 9126, which included functionality, reliability, usability, efficiency, maintainability and portability. The results show (Figure 34) that the most popular factors in descending order were reliability (35%), functionality (25%), maintainability (19%), efficiency (17%), usability (5%) and portability (0%).

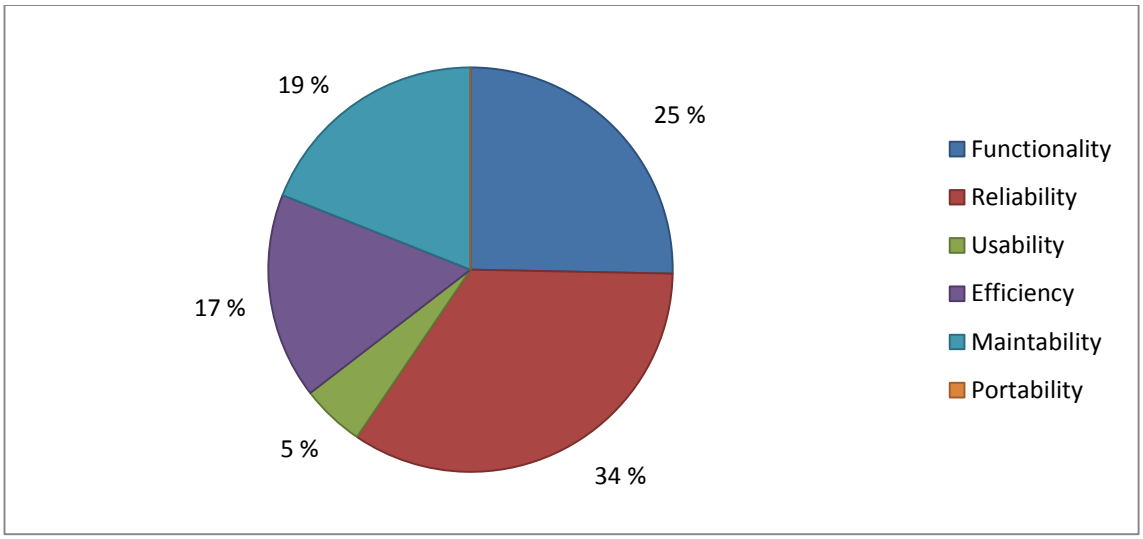


Figure 34. Software quality characteristics

The respondents were also asked what kind of software quality activities they have. Most of the respondents answered that planning QA before starting development is really important activity to do. This includes good communication with customers to find out exact requirements. In the case where the company don't have an own QA department or team and its having outsourced testing, it is important to have a good communication with the other companies and use different testing tools to make processes more effective.

### 4.5.2 Software testing

Software testing is an important part of the quality assurance. The respondents were asked what test phases they are using during their software testing and are the tests done

manually or automatically. Figure 35 shows that all the test phases are being in used almost equally. The most popular phases in descending order include unit testing (27%), user acceptance testing (25%), integration testing (25%), product verification testing (12%) and system testing (11%).

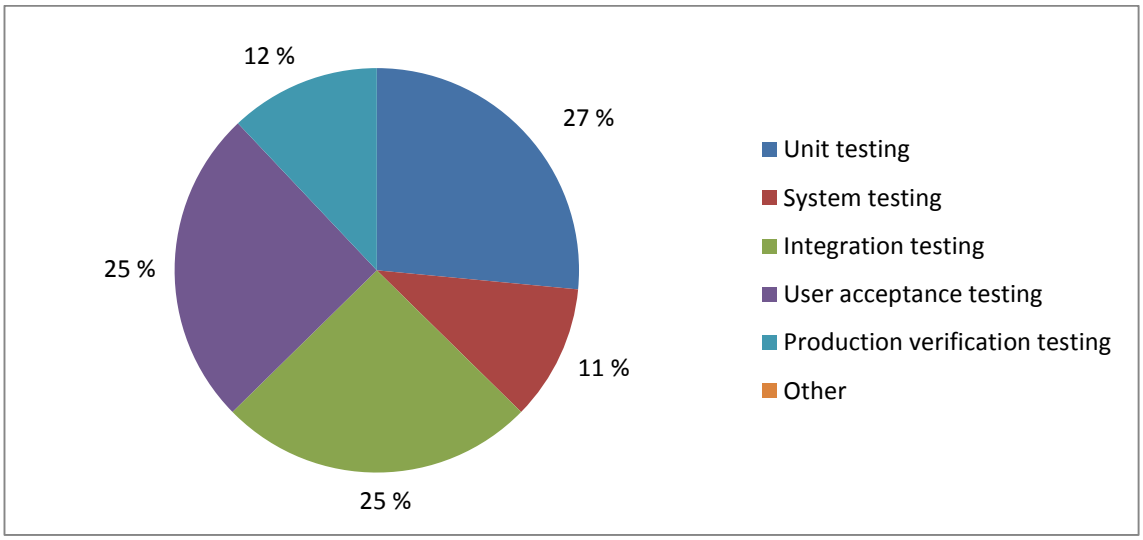


Figure 35. Testing phases

Figure 36. shows what amount of software testing is done automatically and manually. The results show that 45% of companies are using only manual testing and 16% only automatic testing. The remaining companies (39%) are using both manual and automatic testing.

## 4.6 South Korean software industry

IT is one of the biggest export areas of South Korea. It was the last main topic of this empirical study. Questions tried to find out what the respondents think about current and future state of South Korean software industry. The study also tried to find out biggest strengths and weaknesses of South Korean software industry.

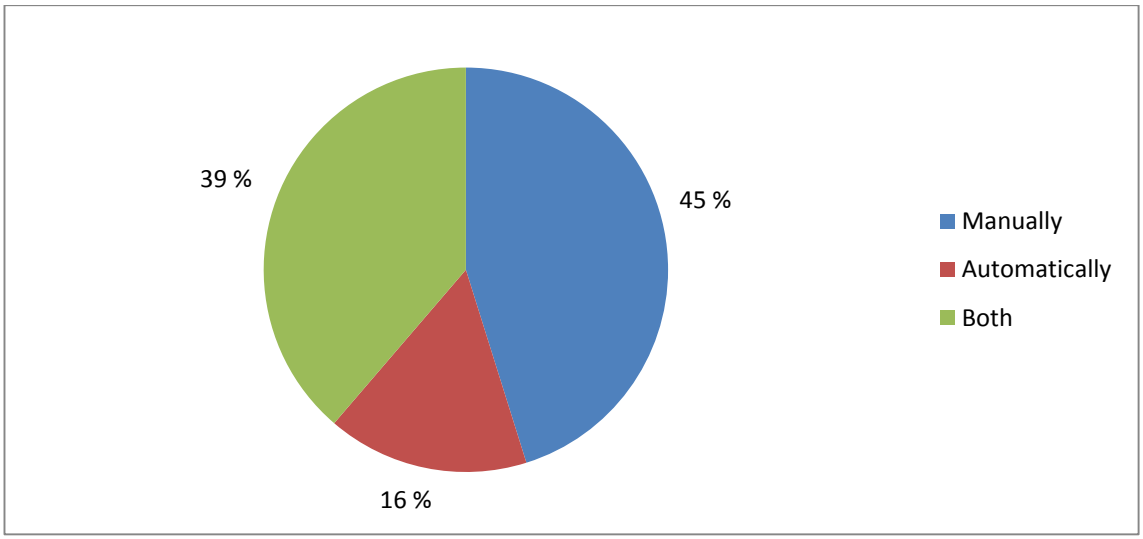


Figure 36. Manual vs. automatic testing

### 4.6.1 Current and future state of South Korean software industry

The respondents were asked their opinion about current and future state of South Korean software industry. Over 80% of the respondents think that the South Korean software industry is not in a good shape. Analysis of the answers revealed three major points which the respondents are most worried about. The three main points include labor force, support of government and problems with start-up companies.

Most of the respondents thought that the biggest problem in the current South Korean software industry is the labor force. Respondents mentioned as a problem the treatment of the labor force in the software industry. Salaries are not high enough and they are lower than in other occupations and it hinders the interest in software development field. Work hours are too long and even in some cases workers are not paid for extra hours. These kinds of problems are the reason why many students are not willing to take IT courses in the university and go to the software industry. This also means that older employees want to move out from South Korea. This leads to the situation where the labor force must come from other countries, while the knowledge of South Korean software industry workers is not developing.

Second major problem in South Korean software industry is the support of government. Most of the respondents felt that South Korean government is not giving enough money and support at the moment, because people are underestimating the importance of

software. Lack of support and awareness will lead to lower competition and will start to show up in quantity of the software.

Last major problem that the respondents mentioned was with start-up companies. South Korean software industry relies more on big companies with system integration business, rather than small companies with solutions. Respondents felt that starting a new company is hard due to policies and restrictions of government. Other problems that were mentioned with lower quantity included the infrastructure of South Korean software industry and difficulties with providing systems. Also one of the respondents mentioned that the use of software life cycle methods is not on a good level.

Although majority of the respondents think that the current state of South Korean software industry is on a weak level, many of the respondents are still thinking that the future of the South Korean software industry is looking good. The respondents think that current government is making good changes and decisions regarding policies in software industry and that will help South Korean software industry to grow and get in a better shape in the future. Mutual vision among the respondents was that if South Korean government start to support software industry it will start to grow and produce better results.

### 4.6.2 Strengths and weaknesses of South Korean software industry

Figure 37 shows what the respondents think to be the biggest strengths of South Korean software industry. Almost half of the respondents (45%) think that South Korean employees are the biggest strength in industry. Many of the respondents say that South Korean workers are diligent, fast workers and have passion for learning software development. Second major strength is the infrastructure of South Korean software industry (26%), that is well organized and has good networking capabilities. Three smaller strengths mentioned include rapid development (8%), culture for technology in South Korea (7%), and interest on new development methods inside developers (7%). The category other (7%) included, for example, growing awareness of software and support for education.

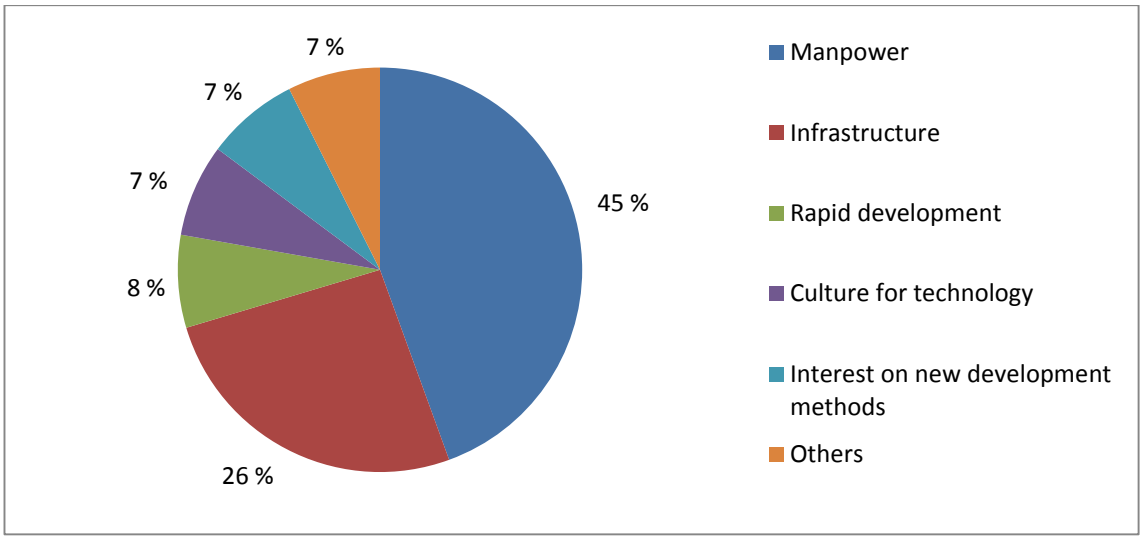


Figure 37. The biggest strengths of South Korean software industry

When asking about the weaknesses of South Korean software industry (Figure 38.), respondents mentioned two major weaknesses. The first major weakness is the treatment of employees that was also mentioned as the reason for the current negative state of the South Korean software industry. The second major weakness was organizational problems that included old organizational cultures, bad systems and lack of long term plans inside companies. Rest of the weaknesses included lack of cooperation between companies (9%), small market (9%), globalization (9%), support of government (6%), foundation of software in South Korea (6%) and others (6%) that included insufficient education and copyright culture.

## 4.7 Statistics of analyzed data

Figure 39 shows Pearson correlation coefficient between multiple choice questions in the survey.

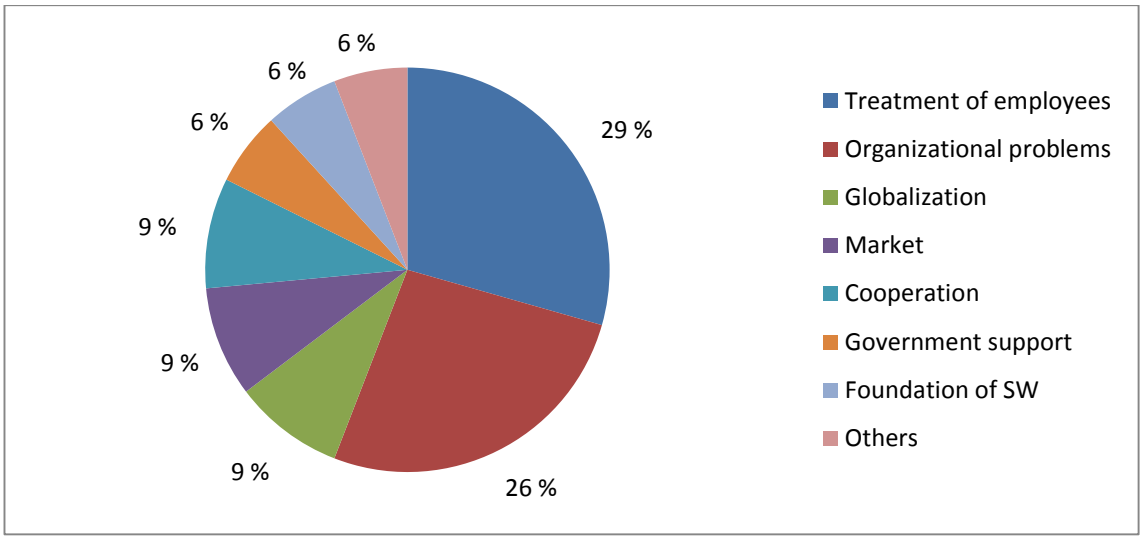


Figure 38. The biggest weaknesses of South Korean software industry

| | 2 | 3 | 4 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 14 | 15 | 16 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **2** | **1** | 0.24 | -0.15 | -0,1 | 0.48 | 0.01 | 0.08 | -0.05 | 0.11 | 0,22 | 0.05 | 0.02 | 0.14 | 0.08 |
| **3** | 0,24 | **1** | -0.34 | 0,38 | 0.43 | -0.38 | -0.05 | **-0,54** | -0.14 | -0,15 | -0.13 | 0,41 | 0.03 | 0.25 |
| **4** | -0,15 | -0.34 | **1** | -0,3 | -0.44 | 0.05 | -0.42 | 0.05 | 0 | 0,1 | 0.23 | -0.01 | 0.05 | -0,4 |
| **6** | -0,1 | 0.38 | -0.3 | **1** | 0.49 | 0.26 | 0.16 | 0.25 | 0.32 | -0,05 | -0.02 | 0.34 | -0.11 | -0.09 |
| **7** | 0,48 | 0.43 | -0.44 | 0,49 | **1** | 0.07 | 0.28 | 0.4 | 0 | -0,22 | 0.16 | 0.17 | **0,57** | -0.04 |
| **8** | 0,01 | -0.38 | 0.05 | 0,26 | 0.07 | **1** | **0,64** | **0,56** | **0,69** | 0,28 | -0.23 | -0.41 | 0.26 | 0.25 |
| **9** | 0,08 | -0.05 | -0.42 | 0,16 | 0.28 | **0,64** | **1** | **0,53** | **0,58** | 0,15 | -0.17 | -0.32 | 0.26 | **0,54** |
| **10** | -0,05 | **-0,54** | 0.05 | 0,25 | 0.4 | **0,56** | **0,53** | **1** | **0.5** | -0,03 | 0.08 | -0.4 | 0.39 | 0.3 |
| **11** | 0,11 | -0.14 | 0 | 0,32 | 0 | **0,69** | **0,58** | **0.5** | **1** | 0,34 | **-0,5** | -0.32 | 0.12 | 0.23 |
| **12** | 0,22 | -0.15 | 0.1 | -0,05 | -0.22 | 0.28 | 0.15 | -0.03 | 0.34 | **1** | -0.33 | -0.18 | -0.31 | -0.04 |
| **14** | 0,05 | -0.13 | 0.23 | -0,02 | 0.16 | -0.23 | -0.17 | 0.08 | **-0,5** | -0,33 | **1** | 0.37 | -0.04 | -0.12 |
| **15** | 0,02 | 0,41 | -0.01 | 0,34 | 0.17 | -0.41 | -0.32 | -0.4 | -0.32 | -0,18 | 0.37 | **1** | -0.23 | -0.12 |
| **16** | 0,14 | 0.03 | 0.05 | -0,11 | **0,57** | 0.26 | 0.26 | 0.39 | 0.12 | -0,31 | -0.04 | -0.23 | **1** | 0.01 |
| **19** | 0,08 | 0.25 | -0,4 | -0,09 | -0.04 | 0.25 | **0,54** | 0.3 | 0.23 | -0,04 | -0.12 | -0.12 | 0.01 | **1** |

Figure 39. Pearson product-moment correlation coefficients

The most notable positive correlation occurs between questions 8, 9, 10 and 11. Questions are about changes in productivity, quality, costs and customer satisfaction when using agile methods. This shows that these four factors that are used to measure software development projects are strongly related to each other. For example better quality in projects leads also to good productivity and customer satisfaction. Significant

positive correlation also appears between questions 7 and 16, where knowledge of agile methods correlates with the ISO 9126 quality factors, and between questions 9 and 19, where the quality of agile methods correlates with used testing methods. These correlations are not necessary important, because in questions 16 & 19 it was possible to give multiple answers.

Significant negative correlation appears in two cases. The first case is between questions 3 and 10, where participation of the customer is compared to the cost of the project when using agile methods. This correlation gives indication that customer participation lowers the cost of the project when using agile methods. Second case is between questions 11 and 14, where satisfaction of the customer is correlating with existence of quality assurance plan and indicates that lack of quality assurance plan lowers customer satisfaction.

# 5 DISCUSSION

This chapter discusses about the results of the completed survey. The research has been limited to companies that are located in the South Korea and that have a software development unit. The purpose is to answer three research questions defined at the beginning of the study. The results are being compared to similar studies about discussed topics. The purpose is to compare how South Korean software development is differentiating from other regions and what similarities do they have. The other studies that are being compared are shown in Table 7.

Table 7. Comparable studies of discussed topics

| Paper reference | Scale/Region | Year | Number of respondents | Focus area |
|---|---|---|---|---|
| Johnson (2002) | Global | 2002 | 131 | Agile methods |
| Schindler (2008) | Austria | 2008 | 61 | Agile methods |
| Rodriguez et al. (2011) | Finland | 2011 | 408 | Agile and lean methods |
| Awad (2005) | Not available | 2005 | 15 | Agile methods and SQA |
| Garousi & Zhi (2012) | Canada | 2010 | 246 | Software testing practices |
| Laanti et al. (2010) | Global | 2010 | +1000 | Agile methods |
| Stelzer et al. (1996) | Europe | 1996 | 36 | Quality assurance standards |
| French Scrum User Group (2009) | France | 2009 | 230 | Agile methods |

## 5.1 Use of software life cycle models and used methods in South Korea

**What software development methods and quality assurance plans South Korean companies are using during software life cycle?**

The results of completed survey show that the use of agile methods is slightly surpassing traditional methods within the companies in South Korean software industry. The most used agile methods include Scrum and XP, while the most popular traditional method is Waterfall. Other studies from similar topic also show that the use of agile methods are surpassing traditional methods or getting really close to it (Schindler (2008), Rodriguez et al. (2011), Garousi & Zhi (2012)). This shows that South Korean software industry follows the trend and is willing to learn and try agile methods as their software development method.

Interesting part about software life cycle models and used methods in South Korea is the big amount of hybrid models that combines the use of agile methods and traditional methods. The large amount of hybrid models indicates that companies find some aspects of agile methods and traditional methods useful to their projects and don't want to try them on their own. Another interesting result from the survey was the observation of the relationship between development method and the company size. Most of the SME's are using traditional methods, instead of agile methods. Agile methods are usually known as a method of SME's.

In the survey it was asked if South Korean companies have any quality assurance plans and what software quality assurance standards they are using. The results were surprising because almost half of the respondents did not have software quality assurance plan and the ones that are using do not use any familiar software quality standards such as the ISO and IEEE series. These findings are really important, because software quality assurance is the major part of the software life cycle.

Another interesting result was the use of quality assurance standards in software life cycle. Overwhelming amount of South Korean companies are not using any familiar software quality standards, like for example ISO and IEEE series. Instead of known software quality standards, companies are using homemade quality standards or no standards at all. Lack of known quality assurance standards use does not automatically

mean that quality of the software is low. Instead it is possible that South Korean companies think that with their current quality assurance planning, they deliver quality that passes their quality measurements. In a case where known quality standards are not used because lack of knowledge, it is really important to start education about them. Study from Stelzer et al., (1996) shows that with software quality assurance standards, it is possible to improve quality in projects.

The survey also revealed that software testing in South Korea is in a good level. All the phases of software testing are being used, and the level of automated testing is on a good level. Comparing to Garousi & Zhi (2012) research, the amount of automated testing is much higher than manual testing.

## 5.2 Effect of agile methods on software life cycle

**How the use of agile methods affect to software life cycle?**

Agile methods being really popular amongst South Korean software industry, it is important to discuss about the effects of agile methods on software development projects. The data gathered from the South Korean software industry shows that agile methods increases quality, productivity and customer satisfaction and in some cases lower the cost when they are used in projects. Awad (2005), Laanti et al. (2010), French Scrum User Group (2009), Garousi & Zhi (2012) and Johnson (2003) also gathered similar results in their research, where productivity, quality and customer satisfaction were significantly better, while costs were mainly unchanged.

We can see that the use of agile methods affects positively to the projects. This means that people behind the agile method must have certain level of skill and knowledge when using them. Experience and knowledge of agile methods within South Korean software industry shows interesting results. South Korean companies seem to have a decent amount of experience with the use of agile methods, but still they rate their level of knowledge mostly average and low. This shows that you don't necessary need a high level of knowledge with agile methods to get better results. On the other hand it is interesting that one of the disadvantages of agile methods mentioned amongst South Korean software companies was that it might need experienced people to run it. These

results also indicate that South Korean software industry may lack educational aspect of agile methods.

Agile methods emphasize that communication, changing requirements and working software are important factors during project development process. The results of survey show that South Korean software industry thinks that the same factors are reason, why agile methods are so good to use in projects.

Based on the gathered information it is safe to say that adaptation of agile methods has positive effect to South Korean software industry. With higher level of experience and knowledge it could have even more effect, when companies have better base knowledge on agile methods. Remarkable fact is that all of the South Korean companies, who were using agile methods, are planning to use them in the future projects.

## 5.3 Strengths and weaknesses of South Korean software industry

**What are the biggest strengths and weaknesses of South Korean software industry?**

Strengths of South Korean software industry are looking positive. Majority of the respondents thought that South Korean software industry has extremely hard working employees. Hard working employees are the basis of a good company, and the most important reason why companies are able to produce software with good quality. Also the infrastructure of South Korea software industry is well build and it includes good networking possibilities. This shows that South Korea has good basis for good software industry that requires high-level people and infrastructure behind it.

The weaknesses of South Korean software industry are treatment of employees and organizational problems. Weaknesses like this have a large impact on companies and industry. Bad treatment of employees, like low salary or long hours, directly impacts negatively. Survey revealed that students in the universities are not taking software related courses, because software industry has bad reputation at the moment. It is obvious that bad reputation is the reason why people are avoiding software industry.

This leads to hiring work force from other countries than South Korea while development of South Korean own software labor is ending.

Another major weakness that got mentioned was lack of government support. Especially IT being one of the biggest export areas of South Korea, the lack of government support on software industry is really surprising. Respondents mentioned that it is really hard to start new companies because of current policies. This also results to bad reputation of software industry.

## 5.4 State of the South Korean software industry

Majority of the respondents thought that South Korean software industry is not in good shape and the results gathered from the survey also indicate this. South Korean software industry has both positive and negative factors in it. Because of this it is really important to draw conclusion about the overall state of South Korean software industry.

Use of software life cycle models and methods seems as a positive factor in South Korean software industry. Use of different models and methods is diverse, which shows that companies are really looking for the right one to fit to their company structure, instead of just using basic waterfall model to every project. The amount of traditional software development methods in SME's indicate that smaller South Korean software companies are still counting on them, instead of using just agile methods. Every project has different factors that decide which model or method suits best and if companies are able to learn about different models and methods, it might result to better results in the future projects.

The results also revealed that the use of agile methods increases productivity, quality, customer satisfaction and in some cases the costs of the project. Also the level of knowledge was mentioned to be mostly average and low. This shows that South Korean software industry should put more effort especially in SME's on learning and using of agile methods. This step might lead to even better results on projects, which automatically increases the overall image of South Korean software industry.

Software quality assurance in South Korean software industry is factor that needs a lot of investigation in the future. The results showed that quality assurance is not a high priority, because almost half of the companies don't have any software quality assurance plans linked with their product life cycle. Also the lack of known quality assurance standards in use is really low, which tells that knowledge of standards is really low or companies do not find the use of them efficient. Raising the knowledge of quality standards like ISO/IEC or IEEE has possibility for increasing the level of quality in the South Korean software development.

The most positive and at the same time most negative factor that is affecting the state of the South Korean software industry is work force in it. Employees are hard-working and ready to produce software, but at the same time they don't have necessary the best environment for it. Negative factors affecting employees have a large impact on the whole software industry and its future. This is the most important aspect that South Korean software industry has to take care off to ensure the productivity and quality of software coming out from South Korea. Although many of the respondents thought that current state is not looking good, many think that future is still looking good. Government is starting to make good changes to software industry that has possibility to boost companies to better results.

# 6 CONCLUSIONS

The purpose of this study was to find out information about the state of South Korean software industry. The study was executed with survey method and collected data was used to answer the research questions and bring results for STX-project. The total number of respondents was 34. The three research questions that this study is answering are:

1. What software development methods and quality assurance plans are South Korean companies using during the software development life cycle?

2. How the use of agile methods affect software development life cycle in South Korea?

3. What are the biggest strengths and weaknesses of the South Korean software industry?

For the first research question the survey asked what kind of software life cycle models and methods are respondents companies using. The results of the completed survey showed that the use of agile methods is slightly surpassing the use of traditional software development methods. Also the use of so called hybrid methods that include aspects from both development methods is really popular. This shows that South Korean companies are adapting the use of agile software development methods. Another part of the first research question was the use of quality assurance. The survey revealed interesting result because almost half of the South Korean companies do not use any software quality assurance plan in their projects. Companies were also asked if they use any software quality standards in their plans. Most of the companies are using homemade standards or no standards at all; the use of quality assurance standards such as ISO/IEC and IEEE was really low.

For the second research question the survey asked respondents how the use of agile methods affects to productivity, quality, customer satisfaction and costs of a software project. It was also asked if respondents are going to use agile methods in the future and are the biggest strengths and weaknesses in using them. The results revealed that with the use of agile methods productivity, quality, and customer satisfaction are higher than

with traditional methods. Some respondents thought that costs were lower, but overall it seems that costs don't have an effect when using agile methods. These are interesting results considering that respondents did not described their knowledge of agile methods that high. This shows that with average knowledge, agile methods can make project more efficient. Respondents thought that biggest strengths in agile methods are ability to respond to changing requirements and better communication. Weaknesses are that it requires people with knowledge of agile methods to work. The results also revealed that every respondent who was currently using agile methods is also going to use it in the future projects.

For the last research question it was asked what the respondents think about the current and future state of the South Korean software industry, and what are the biggest strengths and weaknesses in it. Overwhelming amount of the respondents thought that current the state of the South Korean software industry is not good. The respondents thought that the biggest reason for this is the bad treatment of the labor force in South Korean software industry. Work hours are too long and even in some cases workers are not paid for extra hours. Salaries are not high enough and they are lower than other in occupations and that hinders the interest in software development field, which results to lack of talented people in software industry. Another big reason for the bad state of industry was the lack of government support and difficulties in starting new companies because of policies. Although respondents think that the current state is bad, many feel that the future is looking bright. South Korean software industry has good labor force that is talented and diligent and the infrastructure behind it has good qualities. Also the South Korean government is starting to make changes on supporting software industry and that will hopefully have a large impact on it.

Overall conclusion is that South Korean software industry has clear strengths and weaknesses that will affect the current and future state. Major strengths include labor force, infrastructure and diversity in software life cycle models and methods. Companies are willing to try different styles of development to find the best one for their projects. Weaknesses include treatment of labor force, government support and lack of software quality assurance planning. South Korean government started IT future visions 2020 plan with the purpose of strengthen the current software industry for the future. It is interesting to see what kind of effect this plan has on the South Korean software industry.

# REFERENCES

Abrahamsson, P., Salo, O., Ronkainen, J., Warsta, J. (2002). Agile software development methods: Review and Analysis. Espoo, Finland: Technical Research Centre of Finland, VTT Publications 478.

Awad, M. (2005). A comparison between agile and traditional software development methodologies. [online] cited 4.10.2013. Available at: http://pds10.egloos.com/pds/200808/13/85/A_comparision_between_Agile_and_Traditional_SW_development_methodologies.pdf

Beck, K., (1999). Embracing change with Extreme Programming, IEEE Computer, Vol. 32, Issue 10, October 1999.

Bentley, J.E. (2005) Software Testing Fundamentals – Concepts, Roles, and Terminology. Proceedings of SAS Conference, Philadelphia. Pennsylvania, pp. 1-12.

Boehm, B. (1986). A Spiral Model of Software Development and Enhancement, ACM SIGSOFT Software Engineering Notes.

Boehm, B., Turner, R. (2003). Using Risk to Balance Agile and Plan-Driven Methods. IEEE Computer, Vol. 36(6), IEEE Computer Society.

Botella, P., Burgués, X., Carvallo, J. P., Franch, X., Grau, X., Marco, J., Quer, C. (2004). ISO/IEC 9126 in Practice: What Do We Need to Know. In Software Measurement European Forum 2004, pp. 297-306, January 2004.

Brooks, F. (1987). No Silver Bullet: Essence and Accidents of Software Engineering, Computer, Vol. 20, No. 4, pp.10-19.

Callaghan, K. (1996). The Correlation Coefficient. In Linda Kime & Judy Clark (Eds.) Explorations in College Algebra: Discovering Databased Application. NY: John Wiley. 553-557.

Cerritos College (2013). A Brief Guide to the Analysis of Open-Ended Survey Questions. [online] cited 4.10.2013. Available at: http://cms.cerritos.edu/uploads/ResearchandPlanning/Brief_Guide_to_Open-Ended_Survey_Questions.pdf

Cooper, D.R., Schindler, P.S. (2003). *Business Research Methods*, 8th Edition, Irwin/ McGraw-Hill.

Creswell, J.W. (1994) Research Design: Qualitative and Quantitative Approaches. Thousand Oaks; London: SAGE.

Deemer, P., Benefield, G., Larman, C., Vodde, B. (2010). The Scrum Primer (Version 1.2).

eGov Innovation (2012). [online] cited 15.3.2013. Available at: http://enterpriseinnovation.net/article/south-korea-strengthen-software-industry-launch-it-future-vision-2020-plan.

Eldon, L. (1990). Software testing in system development process: A life cycle perspective. Journal of Systems Management. August.

European Commission (2005). The new SME definition. User guide and model declaration. [online] cited 11.8.2013. Available at: http://ec.europa.eu/enterprise/policies/sme/files/sme_definition/sme_user_guide_en.pdf.

Extreme Programming (2013). [online] cited 20.3.2013. Available at: www.extremeprogramming.org

Fowler, M., Highsmith, J. (2001). The Agile Manifesto. Software Development 9(8): 28-32.

French Scrum User Group (2009). "A National Survey on Agile Methods in France", French Scrum User Group, June 2009. www.frenchsug.org

Garousi, V. Zhi, J. (2012). A survey of software testing practices in Canada. The Journal of Systems and Software 86 (2013) 1354– 1376.

Given, L. M. (2008). The Sage Encyclopedia of Qualitative Research Methods. Sage Publications: Los Angeles, California.

Gornik, D. (2001). IBM Rational Unified Process: Best Practices for Software Development Teams. IBM Corporation Software Group, New York.

Highsmith, J. (2002). Agile Software Development Ecosystems. Boston, MA: Addison-Wesley.

IEEE 730 (2002). IEEE Standard for Software Quality Assurance Plans. The Institute of Electrical and Electronics Engineers.

International Organization for Standarization, ISO Standard 9126: Information Technology - Software product quality, parts 1 and 4, International Organization for Standarization, Geneve, 1999 (part 1), 2002 (part 4).

International Organization for Standarization, ISO Standard 14598: Information Technology - Software Product Evaluation, part 1, International Organization for Standarization, Geneve, 2012.

International Organization for Standarization, ISO Standard 25000: Software engineering — Software product Quality Requirements and Evaluation (SQuaRE) — Guide to SQuaRE , part 1, International Organization for Standarization, Geneve, 2005.

Johnson, M. (2003). Agile methodologies: Survey results. Victoria, Australia: Shine Technologies.

Kane M., Schwaber , K. (2002) . Scrum with XP. Prentice Hall.

Korea Times (2011). [online] cited 15.3.2013. Available at: http://www.koreatimes.co.kr/www/news/biz/2012/11/123_98575.html

Laanti, M., Salo. O., Abrahamsson, P. (2010). Agile methods rapidly replacing traditional methods at Nokia: A survey of opinions on agile transformation. Information and Software Technology 53 (2011) 276–290.

Luciano, R. G., Plínio, R. S. V. (2005). Comparing Software Development Models Using CDM, ACM.

Mahnič, V., Drnovšček, S. (2005). Agile Software Project Management with Scrum, EUNIS 2005 Conference – Session papers and tutorial abstracts, University of Manchester, June.

Miller, J. (2013). Reliability and validity. [online] cited 4.10.2013. Available at: http://michaeljmillerphd.com/res500_lecturenotes/reliability_and_validity.pdf

Miller, R. W., Collins, C. T. (2001). Acceptance Testing, Procs. XPUniverse, July 2001.

Misic, V. (2006). Perceptions of Extreme Programming: An Exploratory Study. AACM SIGSOFT Software Engineering Notes March 2006 Volume 31 Number 2.

Munassar, N., Govardhan, A. (2011). Comparison between Traditional Approach and Object-Oriented Approach in Software Engineering Development, International Journal of Advanced Computer Science and Applications, Vol. 2, No. 6, pp. 70-76.

Munassar, N., Govardhan, A. (2010). A Comparison Between Five Models Of Software Engineering. IJCSI International Journal of Computer Science Issues, Vol. 7, Issue 5, September.

Nerur, S., Mahapatra, R. and Mangalaraj, G, "Challenges of Migrating to Agile Methodologies", Communications of the ACM, Vol. 48, No. 5, May 2005, pp. 72-78.

Nikiforova, O., Nikulsins, V., Sukovskis, U. (2009). Integration of MDA framework into the model of traditional software development, in Selected Papers from the Eighth International Baltic Conference Baltic DB&IS 2008, H.-M. Haav and A. Kalja, Eds., Series "Frontiers in Artificial Intelligence and Applications", Databases and Information Systems V. IOS Press, 2009, pp. 229-242.

O'Docherty, M. (2005). Object-Oriented Analysis & design – understanding system development with UML 2.0, John Wiley.

OSEC (2011). South KoreaInformation and Communication Industry. Swiss Business Hub Korea Seoul, August 2011 [online] cited 15.3.2013. Available at: http://www.osec.ch/sites/default/files/BB_1108_E_Branchenbericht-S%C3%BCdkoreaICT.pdf

Owens, D., Khazanchi, D. (2009). Software Quality Assurance. In: Kidd, T.T., Editor (2009), Handbook of Research on Technology Project Management, Planning and Operations, Hershey, PA: Information Science Reference (an imprint of IGI Global), Chapter XVI, pp. 242-260.

Park, K. (2001). Development of ICT Indicators in Korea. IAOS Satellite Meeting on Statistics forthe Information Society August 30 and 31, 2001, Tokyo, Japan

Petersen, K., Wohlin, C., Baca D. (2009). The Waterfall Model in Large-Scale Development; In: Proceedings of the 10h International Conference on Product Focused

Software Development and Process Improvement (PROFES 2009); Oulu, Finland; Springer LNBI Vol. 32.

Pfleeger, S.L., Kitchenham, B. (2002) Principles of Survey Research (parts 4, 5). Software Engineering Notes vol. 27 no. 5. ACM SIGSOFT.

Pressman, R. S. (2005) Software Engineering: A Practitioner's Approach, 6th Edition, McGraw-Hill.

Punkka, T. (2005). Agile Methods and Firmware Development. HUT / SoberIT.

Rajasekar, S., Philominathan, P., Chinnathambi, V. (2006). Research methodology. [online] cited: 5.8.2013. Available at: http://arxiv.org/pdf/physics/0601009.pdf

Rodríguez, P., Markkula, J., Oivo, M., Turula, K. (2011). Survey on agile and lean usage in Finnish software industry. In Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement (ESEM '12). ACM, New York, NY, USA, 139-148.

Schindler, C. (2008). Agile Software Development Methods and Practices in Austrian IT-Industry : Results of an Empirical Study. Technology. 2008:321-326.

Schwaber, K. (1995). Scrum Development Process. OOPSLA'95 Workshop on Business Object Design and Implementation. Springer-Verlag.

Sellers, R. (1998). Qualitative versus quantitative research - choosing the right approach. Originally published in The NonProfit Times, March 15, 1998.

Shao, D., Khurshid, S., Perry, D. E. (2007). A Case for White-box Testing Using Declarative Specifications Poster Abstract, in Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION, 2007. TAICPART-MUTATION 2007, 2007, p. 137.

Sommerville, I. (2004). Software Engineering, 7th Edition, Addison Wesley.

Stelzer, D., Mellis, W., Herzwurm, G. (1996). Software Process Improvement via IS0 9000? Results of two surveys among European software houses. Proceedings of the 29th Annual Hawaii International Conference on System Sciences – 1996.

Sukamolson, S. (2007). Fundamentals of quantitative research. [online] cited 6.8.2013. Available at: http://www.culi.chula.ac.th/e-journal/bod/suphat%20sukamolson.pdf

Sutherland, J. (2010). Scrum Handbook. Scrum Foundation.

Taylor-Powell, E., Hermann C. (2002). Collecting Evaluation Data: Surveys. University of Wisconsin Cooperative Extension. [online] cited 10.8.2013. Available at: http://learningstore.uwex.edu/assets/pdfs/G3658-10.PDF.

Webropol (2013). [online]. Available at: www.webropolsurveys.com

Yu, B., WooiKhong, L., Wai, Y. T., Soo, F. T. (2012). Software Development Life Cycle AGILE vs. Traditional Approaches. International Conference on Information and Network Technology IPCSIT vol. 37, IACSIT Press, Singapore.

# APPENDIX 1. Survey of South Korean software industry

## Industry survey: South Korean software development
## 산업조사: 한국의 소프트웨어 개발

### General information (일반정보)

**1.** Respondent & Company
(응답자와 회사정보)

Respondent name (이름)

_____

E-mail (이메일)

_____

Role in company (직책)

_____

Company name (회사이름)

_____

Organizational unit (조직단위/부서)

_____

Industry sector (산업분야)

_____

Number of employees (직원수)

_____

## Software life cycle (소프트웨어 개발 프로세스)

**2.** What software life cycle are you using at the moment?
(현재 어떤 소프트웨어 개발 방법론 (SDLC) 을 사용하고 계십니까?)

**Traditional software life cycle methods (전통적인 소프트웨어 개발 방법론)**

- ☐ Waterfall (폭포수 모델)
- ☐ Spiral model (나선형 모델)
- ☐ Rational Unified Process (RUP)
- ☐ Other, what? (다른 것을 사용한다면 무엇을 사용 하십니까?)

**Agile software life cycle methods (애자일 소프트웨어 개발 방법론)**

- ☐ Scrum (스크럼)
- ☐ Extreme Programming (XP)
- ☐ Lean Software Development (LSD, 린 소프트웨어 개발)
- ☐ Feature Driven Development (FDD, 기능 중심 개발)
- ☐ Dynamic Systems Development Method (DSDM)
- ☐ Other, what? (다른 것을 사용한다면 무엇을 사용 하십니까?)

**Others (그 외의 방법)**

- ☐ Homemade (직접 개발한 모델)
- ☐ Other, what? (이외에 다른 것을 사용한다면 무엇을 사용 하십니까?)

**3.** How much are customers participating on your software life cycle?
(소프트웨어 수명에 있어서 고객들과 얼마나 자주 소통하고 계십니까?)

- ○ Only couple meetings during life cycle (전체 개발 기간 동안 두세 번)
- ○ Meetings once in a month (한 달에 한번)
- ○ Meetings once in a week (일주일에 한번)
- ○ Meetings daily (매일)

**4.** Does your organization have its own independent quality assurance group or department?
(귀사에는 품질보증(QA)을 위한 자체적인 팀이나 부서가 따로 있습니까?)

- ○ Yes(있다)
- ○ No(없다)

**5.** If your organization has its own quality assurance group or department, what percent does it consists of your organizational unit and how much are they communicating with other groups during the life cycle? (귀사에 품질보증을 위한 자체적인 팀이나 부서가 있다면, 당신이 속한 조직단위(부서/팀)에서 차지하는 비율(%)은 어느 정도이며, 소프트웨어 개발 프로세스 동안 조직의 다른 그룹들과의 소통은 얼마나 이루어지고 있습니까?)

_____

_____

_____

_____

## Agile software development methods (Answer only if you are using them) 애자일 소프트웨어 개발 방법론 (사용하고 계신 경우에만 답해주세요)

**6.** How long have you been using agile methods? (사용기간이 어떻게 되십니까?)

- Under 6 months (6 개월 이내)
- From 6 months to 2 years (6 개월에서 2 년사이)
- More than 2 years (2 년 이상)

**7.** How would you describe your knowledge of agile methods?
(애자일 방법론에 대해 귀하는 어느 정도 숙지하고 있다고 생각하십니까?)

- Very Low (아주조금)
- Low (조금)
- Average (평균)
- High (많이)
- Very High (아주많이)

**8.** Has adoption of Agile Methods affected to your teams productivity?
(애자일 방법론의 도입이 팀의 생산성에 영향을 주었습니까?)

- Productivity went down a lot (생산성이 많이 떨어졌다)
- Productivity went down a little (생산성이 조금 떨어졌다)
- Productivity was unchanged (생산성에 변함 없다)
- Productivity was better (생산성이 조금 좋아졌다)
- Productivity was much better (생산성이아주좋아졌다)

**9.** Has adoption of Agile Methods affected the quality of applications?
(애자일 방법론의 도입이 어플리케이션의 질에 영향을 주었습니까?)

- Quality went down a lot (질이 많이 떨어졌다)
- Quality went down a little (질이 조금 떨어졌다)
- Quality was unchanged (질에 변함이 없다)
- Quality was better (질이 좋아졌다)
- Quality was a much better (질이 많이 좋아졌다)

**10.** Has adoption of Agile Methods affected the cost of development?
(애자일 방법론의 도입이 개발비용에 영향을 주었습니까?)

- Costs were a lot more expensive (비용이 훨씬 많이 들게 되었다)
- Costs were a little more expensive (비용이 좀 더 들게 되었다)
- Costs were unchanged (비용에 변함이 없다)
- Costs were a little less expensive (비용이 줄어 들었다)
- Costs were a lot less expensive (비용이 훨씬 줄어 들었다)

**11.** Has adoption of Agile Methods affected the satisfaction of customers?
(애자일 방법론의 도입이 고객만족에 영향을 주었습니까?)

- Satisfaction went down a lot (만족도가 많이 떨어졌다)
- Satisfaction went down a little (만족도가 조금 떨어졌다)
- Satisfaction was unchanged (만족도에 변함이 없다)
- Satisfaction was better (만족도가 조금 좋아졌다)
- Satisfaction was a much better (만족도가 많이 좋아졌다)

**12.** What feature of Agile Methods do you like the most?
(애자일 방법론의 어떤 부분을 가장 좋아하십니까?)

- Individuals and interactions over processes and tools. (프로세스와 도구보다는 개인과 상호작용)
- Working software over comprehensive documentation. (철저한 문서보다는 동작하는 소프트웨어)
- Customer collaboration over contract negotiation. (계약 협상보다는 고객과의 협동)
- Responding to change over following a plan. (계획을 따르는 것보다는 변화에 대응하는 것)

**13.** Was there any particular reasons or advantages why you chose to use agile methods and are you planning to use them in future projects, if not, what are the disadvantages you find in them? (애자일 방법을 사용하기로 한 특별한 이유나 이점이 있었습니까? 그리고 앞으로의 프로젝트에도 애자일 방법론을 사용하시겠습니까? 그렇지 않다면 애자일 방법론의 좋지 않은 점은 무엇입니까?)

_____
_____
_____
_____

**Software quality assurance (소프트웨어 품질)**

**14.** Do you have software quality assurance plans?
(소프트웨어 품질보증 계획이 있습니까?)

- Yes(있다)
- No(없다)

**15.** Do you use any standards in software quality assurance?
(소프트웨어 품질보증에 표준규격을 사용하십니까?)

- ISO 9126
- ISO 14598
- ISO 25000
- IEEE
- Homemade (자체 규격)
- We don't use (사용하지 않는다)
- Other, which? (이외에 다른 것을 사용한다면 무엇을 사용하십니까?)

**16.** What are your most important factors in software quality assurance (according to ISO 9126)? (3 most important)
(소프트웨어 품질보증에 있어 귀하께 가장 중요한 3 가지 요소는 무엇입니까? ISO 9126 기준)

- Functionality (기능성)
- Reliability (신뢰성)
- Usability (유용성)
- Efficiency (효율성)
- Maintainability(유지보수성)
- Portability (이식가능성)

**17.** What kind of software quality activities you have for these factors?
(위의 6 가지 요소들을 위해 어떠한 소프트웨어 품질보증 활동을 하십니까?)

_____

_____

_____

_____

**18.** What influence do agile methodologies have on testing and quality activities?

(테스팅과 품질보증 활동에 있어 애자일 방법론이 어떤 영향을 줍니까?)

_____

_____

_____

_____

**Software testing (소프트웨어 테스팅)**

**19.** Which kind of testing does your organization perform?
(귀사는 어떠한 테스팅을 하십니까?)

- ☐ Unit testing (단위 테스팅)
- ☐ System testing (계통 테스팅)
- ☐ Integration testing (통합 테스팅)
- ☐ User acceptance testing (사용자 품질 확인 테스팅)
- ☐ Production verification testing (생산 검증 테스팅)
- ☐ Other? (이외에 다른 것을 사용하시면 무엇을 사용하십니까?)

**20.** Is your testing conducted manually or automatically?

(귀사의 테스팅은 수동적으로 수행됩니까, 자동적으로 수행됩니까?)

_____

_____

_____

_____

**South Korean software industry(한국의 소프트웨어 개발 산업)**

**21.** How do you see the Korean software industry today and in the future?

(귀하는 한국 소프트웨어 산업의 현재와 미래를 어떻게 생각하고 계십니까?)

_____
_____
_____
_____

**22.** What do you think are the biggest strengths and weaknesses in Korean software industry?

(한국 소프트웨어 산업의 가장 큰 강점과 약점은 무엇이라고 생각하십니까?)

_____
_____
_____
_____

**23.** Do you have anything else to say?

(하시고 싶은 말씀이 있다면 남겨 주세요)

_____
_____
_____
_____