# Contents

1 Introduction 1

2 Overview of Agile Software Development and Testing 2
   2.1 Agile Testing 2
   2.2 Agile Software Development 3

3 Analysis of Agile Methodologies 7
   3.1 Comparison of Waterfall Approach and iterative Approach 7
   3.2 Iterative Methodologies 8
      3.2.1 Strengths of Iterative Methodology 9
      3.2.2 Weaknesses of Iterative Methodology 10
      3.2.3 Strengths of Waterfall Methodology 11
      3.2.4 Weaknesses of Waterfall Methodology 12
   3.3 Scrum Methodology 13
      3.3.1 Scrum Lifecycle 13
      3.3.2 Members and Roles in the Scrum Team 14

4 Software Testing 16
   4.1 Types of Software Testing 17
      4.1.1 Unit Testing 17
      4.1.2 Acceptance Testing 17
      4.1.3 Performance and Manual Testing 18
   4.2 Improving Software Testing 19
      4.2.1 Automating Software Testing 21
      4.2.2 Software Test Activities 23
   4.3 Benefits of Automated Software Testing 24
      4.3.1 Software Quality 24
      4.3.2 Productivity in Software Development 25
   4.4 Limitations of Automated Software Testing 25
      4.4.1 Lack of Skilled Personnel 26
      4.4.2 Significant Maintenance Costs 26
      4.4.3 Inadequacy to Replace Manual Testing 26
   4.5 Decision Making in Test Automation 27
      4.5.1 Appropriate Task to Automate 27
1 Introduction

Software quality is an essential feature that can be said to enable long-term customer relations because products with fewer bugs increase customer satisfaction. Testing itself is about half of the total work amount that is being used in the software development process. Due to this, it is important that there are good testing processes and tools are used to ensure that the product meets the stakeholders’ quality assurance expectation. User Interface (UI) testing is crucial since most laymen are not too concerned about the hundreds of codes written to make one function in an application. Rather they are interested in the UI and functionalities of the application.

Many software development companies are striving to become more agile. This is because many successful agile teams are able to produce higher-quality software that better meets users’ needs at lower cost and faster than traditional teams [1, 3]. Some use more recent agile methods or other variations of iterative development which are in the main stream of software development [2, 31]. Software developers still search for the best methods and continue to question what benefits agile testing would bring to software development as many researchers and consultants are claiming.

This study’s objective was evaluating the most benefit gained and the limitations of test automation and augmenting manual testing with agile automation testing to enhance testing. The study critically analyzes the current state of UI testing in Nokia Differentiating Software (DSW) and proposes recommendations on to improve and implement an efficient and effective testing process.

Many companies have decided to apply agile methodologies to assure software quality. This has required companies to have more methods and processes of measuring success and benefits of development methods [3].Software testers and quality assurance (QA) experts always need proof that the new methodology is in a positive framework and would yield good results, before they advance and organize it on a larger scale. Big organizations are therefore hesitant to implement changes and adopt new methodologies because of their complexity and the need to incorporate new technologies and processes with the existing ones [4, 2-21].
2 Overview of Agile Software Development and Testing

In software product development, agile methodology is one strategy adopted while addressing the challenges of market dynamics. Testing services have a critical bearing on the success of any agile process and methods [4, 4] as customer satisfaction and return on investment are linked to product quality and cost of quality. Agile methodologies are planned to break the software down into convenient fractions that can be delivered earlier to the customer. The aim of any agile process is to deliver an essential functioning product as fast as possible and then to go through a process of continual improvement. [4, 19-31] An agile project is characterized by having a large number of short delivery cycles (sprints), and priority is given to the feedback-loops from one cycle to the next. [4, 4-6]

The feedback-loops drive continuous improvement and allow the issues that inevitably occur (including the way requirements have been defined or the quality of the solution provided) to be dealt with much earlier in the development life cycle. To accomplish this, companies need to re-think their approach to delivery and have their previously independent contributors like business analysts, developers, testers, and end users working together in teams. The testing agenda during the development phase is very different to the traditional approaches that support the Waterfall or V-Model development methodologies because agile uses a Test Driven Development (TDD) model. [5] However, many of the requirements of the extended testing model will still apply [6, 5-12].

2.1 Agile Testing

Agile testing is a software testing practice that follows the principles of the agile manifesto giving emphasis to the perspective of customers who will make use of the system. Agile testing does not emphasize rigidly defined testing procedures, but rather focuses on testing iteratively against newly developed code until quality is achieved from an end customer’s point of view [6].
In other words, the emphasis is shifted from testers known as quality police to something more like the entire project team working toward demonstrable quality. Agile testing involves testing from the customer perspective as early as possible, testing early and often, as code becomes available and stable enough from module/unit level testing [7]. There are key fundamentals of the agile guiding principles such as individuals are more important than processes. Software that works is more important than documentation and customer collaboration is more important than contracts. The highest priority is to satisfy the customer through early and continuous delivery of valuable software, and accepting changes throughout development.

Agile processes harness change for the customer’s competitive advantage to deliver working software regularly, from a couple of weeks to a couple of months, with a preference to the shorter timescale. Business people and developers must work together daily throughout the project, build projects around motivated individuals and give them the environment and support they need, and trust them to get the job done. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Working software is the primary measure of progress and agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely. Continuous attention to technical excellence and good design enhances agility. Simplicity, the art of maximizing the amount of work not done is essential and the best architectures, requirements, and designs emerge from self-organizing teams. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly and responding to change is more important than following a plan. [8, 315]

2.2 Agile Software Development

Agile software development refers to a group of software development methodologies based on iterative development, where requirements and solutions evolve through collaboration between self-organizing cross functional teams [8, 113]. It is a conceptual framework for software engineering that promotes development iterations throughout the life-cycle of the project.
Agile methods promote a disciplined project management process that encourages frequent inspection and adaptation, a leadership philosophy that encourages teamwork, self-organization and accountability, a set of engineering best practices intended to allow for rapid delivery of high-quality software, and a business approach that aligns development with customer needs and company goals. [8, 213] Agile development starts with the most important and a valuable requirement also called as features or functionalities and continue to the less important ones as time goes by. This ensures that customer gets to see the most desirable content soon after the development has been started [9].

Figure 1. Agile Development [8]

In this manner changes to the original software requirements can still be changed with relatively lower cost as illustrated in figure 1. Whereas the customers in the traditional model of development get only the end product which might as well be something that customer is not satisfied with [10].

Agile development practices increase the velocity at which software teams deliver customer value by improving everyone's visibility into project features, quality and status by doing this the main objective of agile development are to respond faster to changing customer requirements and building your product in short, rapid, fully tested iterations and effectively plan and execute two, two-week iterations for your critical path project which will continually adapt agile principles and practices to your unique environment and to help scale iterative development practices to larger or distributed organizations [11].
According to Scott Ambler 2005, [12] in the beginning of a project there is no need to know details of the requirements and more requirements can come during the evolution of the development. The thing that is important is that the stakeholders can agree which requirements are the most important ones so that team or teams can start working on those. This kind of mentality in a project brings flexibility and improves collaboration in a project. Each starting also required workshop where the stakeholders prioritize all the requirements. Then the team or teams have another sprint planning workshop which is one or two days long [12].

![Iterative development model (Sep 2006)](image)

As illustrated in figure 2, during the planning workshop, team selects the most important features to be developed, tested and documented during beginning an example is Iterative development model.

According to Craig Larman 2004 [14, 9-14] the normal time boxed duration of iteration is between one to six weeks depending on how often resources can finish chosen requirements and how often they need feedback from the customer. The work for a requirement included various tasks such as plan, estimate, construction, testing and documenting, before one can say that the requirement is totally done. [14, 9-14] Usually an agile project has a Definition of Done principles which define when a feature is done and when a release is totally ready for the customers. Definition of Done is an agreement that the project members sets before a sprint can start. [2] The agreement defines what tasks are needed to be undertaken before functionality can be called as done or completed. When the functionality is done, a release is made.
As illustrated in Figure 3, the task usually contains tasks that are needed to be completed before a functionality is partially done. These are the rules of sprints which tell if a goal has been successfully met or not.

To avoid confusion, the Definition of Done needs to be reviewed every time when a sprint starts and if some functionality needs some additional tasks, they are added during the sprint meeting (Lasse koskela). Everybody should agree with the Definition of Done and features that are not done should not be presented in the Sprint Demo meeting [15, 141].

As Figure 4 shows, a project has a product backlog which includes requirements in the order of priority. A team takes a set of requirements to their sprint backlog. The sprint backlog contains the amount of hours the team can use for active work during the time boxed period which is often 30 days, but can be something else as well [8].
The amount of requirements taken into a sprint is reflected to the amount of hours the team has for the time period. Every requirement is time estimated and only the selected numbers of requirements are taken to a sprint as the team has time to finish. Once the team has finished the work they deliver working increment of the software for the customer to evaluate. [17]

3 Analysis of Agile Methodologies

3.1 Comparison of Waterfall Approach and iterative Approach

Software development organizations are often asked what methodology they use. The implication is that one particular methodology must be superior to all others [12]. This is particularly common among consulting companies, who usually claim some sort of competitive advantage based on their unique methodology. Other software tool companies or academics promote their own methodology as the best choice for achieving the highest quality, lowest cost, and fastest development time [14, 3]. Sometimes, proponents will argue for the benefits of their own particular choice with almost religious fervor. However, most development methodologies can be basically categorized as either a waterfall or iterative.

The waterfall approach emphasizes a structured progression between defined phases. Each phase consists of a definite set of activities and deliverables that must be accomplished before the following phase can begin [14, 3-5]. In addition, different people are usually involved during each phase. Strengths of the waterfall approach include the ease of analyzing potential changes, the ability to coordinate large distributed teams, predictable dollar budgets, and the relatively small amount of time required from subject matter experts. On the other hand, the weaknesses of the waterfall methodologies include: lack of flexibility, difficulty in predicting actual needs for the software, the loss of intangible knowledge between phases, discouragement of team cohesion, and the tendency to not discover design flaws until the testing phase.

Iterative methodologies include Extreme Programming and RAD approaches [14, 137]. The emphasis is on building a highly skilled and tightly knit team that stays with the project from beginning to end.
The only formal project deliverables is the actual working software and the essential system documentation that is completed at the end of the project. This approach delivers the following benefits: rapid feedback from users increases the usability and quality of the application, early discovery of design flaws, and an ability to easily roll-out functionality in stages, a more motivated and productive team, and finally knowledge retention for the duration of the project. Offsetting the strengths are the following drawbacks to iterative methodologies: difficulty in coordinating large projects, the possibility for the project to never end, a tendency to not thoroughly document the system after completion, and the difficulty in predicting exactly what features will be possible within a fixed time.

The two fundamental methodologies have advantages and it is recommendable when choosing a methodology, to consider the particular goals of the project and restraints of the methodology. Most often, a combination of aspects of both methodologies are used in developments.

3.2 Iterative Methodologies

The iterative family of methodologies shares a common emphasis on highly intense teams with least structure and access to steady feedback from the Subject Matter Experts (SME’s). While it may appear that they lack design or testing, these types of methodologies actually place a great deal of emphasis on them. They just do it in a different way usually, a project will begin with the integrated project team being pull together and briefed on the project objectives [18]. The team will consist of all of the essential roles from the very beginning and each member may actually play multiple roles. Rather than a distinct sequence through segment, the iterative methodology emphasizes creating a series of working prototypes for evaluation by the SMEs until the objectives are accomplished and the system is ready for final release [18, 47].

During the process, it is critical for the actual project lead as well as the senior members of the team to balance the SME requests against the overall system constraints and platform limitations to ensure that quality and performance objectives can be met. It is the development team’s responsibility to offer constructive feedback to the SMEs in order to suggest alternatives and work together to the best mutual solution.
Where possible, individual team members will be given complete ownership of a particular component and charged with ensuring its usability, quality, and performance. The senior team members are responsible for enforcing quality and consistency standards and this no different with the role of the tester \[18, 49\]. Even on large projects, the initial planning will consist of only a broad outline of the business objectives and creating a framework for the overall project components. In some cases, a set of core features for the entire system will be built initially and subsequent features added as the project progresses \[18, 44\]. In other cases, just certain modules will be built entirely during the early part of the project and other components added over time.

3.2.1 Strengths of Iterative Methodology

Feedback from the subject matter experts, testers or users can be based on an actual working prototype of the system relatively early in the project life cycle. This enables the tester to base his or her feedback on actually working with a limited version of final product. Since the development team receives feedback at early stages in the overall development process, changes in requirements can be more easily incorporated into the finished product \[19, 132\]. More importantly, if the SME determines that a feature would not be as valuable, it can be omitted before too much development time has been spent or integrating the particular component into the overall system.

In a similar way, since the team is deploying actual working prototype versions of the application along the way, a flaw in the design should become more apparent earlier in the project schedule \[19, 70\]. Instead of discovering a potential problem only after the system goes to full-scale testing, more design flaws can be addressed before they impact other features and require significant effort to correct. Because each iteration actually functions (sometimes to a limited degree), deploying parts of the system in a staged roll-out becomes much easier. Using an iterative methodology, the team simply stabilizes an earlier iteration of the component, collaborates with the SME to ensure it is stable and rolls it out. Another advantage of doing a staged roll-out in this way is that actual production use will generate more improvement suggestions to be incorporated in subsequent iterations of the same component and/or other components.
The team approach stressed in the iterative methodology increases overall motivation and productivity. Because the same people are involved from beginning to end, they know that the design choices made will ultimately affect their ability to successfully complete the project [19, 52]. Productivity will be enhanced because of the sense of ownership the project team has in the eventual result. While it may seem like the **empowerment** fad, many studies have found that a team charged with a common goal tends to be much more productive than groups of people with individual incentives and shifting assignments.

The fact that an integrated team maintains a thorough understanding of the project is a more tangible benefit. This effect arises simply by having the same individuals involved from the very beginning and listening first hand to the subject matter experts or the product owner describe their needs and objectives [19, 58]. Subsequent feedback during iteration of projects builds upon the initial understanding. Since the same person is listening to the needs and writing the code, less time needs to be spent authoring documents to describe those requirements for eventual hand-off [20, 3-8]. This translates into more time spent writing and testing the actual software.

Table 1. Strengths and Weaknesses Of Iterative Methodology.

<table>
<thead>
<tr>
<th><strong>Iterative Strengths</strong></th>
<th><strong>Iterative Weaknesses</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>o Rapid feedback from actual users</td>
<td>o Difficulty coordinating larger teams</td>
</tr>
<tr>
<td>o Flexible to address evolving requirements</td>
<td>o Easy to have never ending project if not managed properly</td>
</tr>
<tr>
<td>o Design flaws discovered quickly</td>
<td>o Less documentation done</td>
</tr>
<tr>
<td>o Easy to roll-out new functionality in stages</td>
<td>o Less strictly on accomplishing predict features</td>
</tr>
<tr>
<td>o Higher motivation and great productivity</td>
<td>o Time Consuming</td>
</tr>
<tr>
<td>o Very little knowledge loss between phases</td>
<td></td>
</tr>
</tbody>
</table>

Table 1 is a summary of both the strengths and weaknesses of Iterative Methodology.

3.2.2 Weaknesses of Iterative Methodology

Iterative projects tend to be most effective with small, highly skilled teams. It is more difficult to ensure that the components mesh together smoothly in larger projects.
It is much difficult across larger, geographically distributed projects [14, 79]. While steps can be taken to minimize the chances of failure, coordinating large iterative development efforts is typically very hard to accomplish effectively because of the lack of detailed planning documents. Because there are no specific firm cut-off milestones for new features, an iterative project runs the risk of continuing into perpetuity. Even though one of the strengths is the ability to react to changing business needs, [14, 79] the project leader must determine when the major business needs have been met. Otherwise, the project will continue to adapt to ever changing business needs and the software will never end. This will result in never really deploying a finished product to full production use. This is a risk even in a staged roll-out situation because there are always improvements possible to any software.

In any software project, there is always the tendency to borrow time from the final system documentation tasks to resolve more defects or polish certain features more. This risk increases on iterative projects because there is usually no scheduled documentation period. The result is a system that is very hard to maintain or enhance. In a similar way, in an iterative project it is much easier to fix a definite project schedule or dollar budget than determine exactly what features will be able to be built within that timeline [14, 15]. This is simply due to the fact that the features change based on user feedback and the evolution of design.

3.2.3 Strengths of Waterfall Methodology

However, because of the requirements and design documents contains an abstract of the complete system, the project manager can relatively quickly analyze what impact a change will have on the entire system. An example might be if one developer wanted to modify the fields in a database table or a class. The project manager could look up what other components of the system rely on that particular table or class and determine what side effects the change may have [21]. The same documents that take so much time to assemble at the front-end also make dividing up and subsequently coordinating the work easier. Because the design produced in a waterfall approach is so detailed, it is easier to ensure that the pieces will be easier to integrate when the project nears the end of the programming phase.
Even with a waterfall approach, the only way to ensure a precise up-front budget cost is to have an external vendor submit a fixed bid for the remaining phases of the project after one or two of the initial phases have been completed. In this way, financial risk is contained even if the project takes longer than expected. Perhaps one of the most compelling reasons for using the waterfall approach is simply the relatively small amount of time required of the subject matter experts. In a project typically have other primary responsibilities, their time is limited. The waterfall approach ensures that a significant involvement from them is only required during the initial requirements phase as well as part of the design, testing, and implementation phases.

Table 2. Strengths and Weaknesses Of Waterfall Methodology.

<table>
<thead>
<tr>
<th>Waterfall strengths</th>
<th>Waterfall Weaknesses</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>○ Ease in analyzing potential changes</td>
<td>○ Lack of flexibility</td>
</tr>
<tr>
<td>○ Ability to coordinate larger teams, even if geographically distributed</td>
<td>○ Hard to predict all needs in advance</td>
</tr>
<tr>
<td>○ Ability to budget precisely</td>
<td>○ Intangible knowledge lost between hand-offs</td>
</tr>
<tr>
<td>○ Less total time required from subject matter expert</td>
<td>○ Lack of team cohesion</td>
</tr>
<tr>
<td></td>
<td>○ Design flaws not discovered until the Testing phase</td>
</tr>
</tbody>
</table>

They have very little involvement during the entire programming phase [21]. Table 2 is a summary of both the strengths and weaknesses of Waterfall Methodology.

3.2.4 Weaknesses of Waterfall Methodology

Even though the impact of changes can be analyzed more effectively using a waterfall approach, the time required to analyze and implement each change can be significant. This is simply due to the structured nature of the project, and this is particularly acute when the changes are frequent or large. Also regarding improvements in providing wire frame screen mockups and more detailed flowcharts, the front-end planning process has a hard time with predicting effectively the most effective system design on the front-end [21]. One of the most challenging factors that cause this difficulty is the unfamiliarity most Subject Matter Experts are with formal system design techniques.
One complaint that I have heard a lot is the document looks impressive, but I don’t know if the system will meet my needs until I see the actual screens. Even more disturbing is the inevitable loss of knowledge between the planning and programming phases. Even with the most detailed documents, the analysts and architects always have an implicit understanding of the project needs that are very hard to transfer via paper documents. The information loss is particularly harmful to the project when it is developing a relatively new system as opposed to modifying an existing system.

Closely linked to the knowledge loss effect, is the fact that the waterfall methodology discourages team cohesion [21]. Many studies have found that truly effective teams begin a project with a common goal and stay together to the end. The tendency to switch out project staff from phase to phase weakens this overall team cohesion. In fact, it is common for the project manager be the only person that sees a project from beginning to end. The most significant weakness is the possibilities that a poor design choice will not be discovered until the final phases of testing or implementation.

3.3 Scrum Methodology

Another important concept in agile methods is Scrum, which is a process that offers framework and practices to manage complex projects. A project can have only one team or multiple teams which are called scrum teams depending on how many members are working in the project [14, 109]. A scrum team is a self-directed and self-organizing team, which have a scrum master to help the team to focus on to current goals that the team has chosen for itself to be done during iteration or as called as sprint [15].

3.3.1 Scrum Lifecycle

The scrum lifecycle constructs of four phases which are planning, staging, development and release [14, 113]. The planning phase, performed by the stakeholders, establishes the vision of the project and next sprint (iteration), and prioritizes and gives pre-estimations for requirements. In the staging phase the scrum teams take as much prioritized requirements as is possible to be done during one sprint without overtime.
Team’s velocity is compared to the chosen amount of work and the team gives its final estimations of effort needed for each task. If the team has chosen too much work compared to their velocity the tasks with less priority will be dropped out or other way around. The team can also plan exploratory design and prototypes to stage the work planned for the sprint. The development phase which is the actual sprint includes construction of the features, but also testing those and finishing the documentation needed for the features [14, 114].

Releasing phase also called as demo is between sprints and means that the scrum team releases or demonstrates the features done during a sprint to external stakeholders [14, 134]. In case of final sprint the whole incrementally developed product will be released to the customers. Common procedure in scrum are daily scrum meetings which are short measuring meetings, usually no more than fifteen minutes, where everyone shares what they have been doing, what they are planning to do next and they should state if they have problems preventing them to do what they are planning to do [14, 131]. Bearing it in mind that if the team is large, then the time could exceed fifteen minutes. Focus in daily scrum meetings should always be on problematic issues and in issues which might affect others work.

It is not only a reporting meeting of what have been done. If there are multiple scrum teams working for a project then scrum of scrums which is often a daily meeting or as often as agreed between scrum masters of each team. Scrum masters duty is to spread the relevant knowledge to the scrum of scrums representatives and gathered from scrum of scrums to his or hers team representatively [14, 109]. Within the learning organization a Scrum Team is a self-organizing small team who decides for themselves how they satisfy their value goals [22, 221].

3.3.2 Members and Roles in the Scrum Team

Scrum team has members for each stage of feature development and recommendation is that a team is no bigger than seven persons. All team members are entitled to a role and every scrum team should have all of the following roles: a customer or a product owner, team member who can be a developer, a tester or a documenter, and a scrum master.
Sometimes team does not have a possibility to get its own product owner who would be responsible of the goals of a sprint for the team and which features and in which order the team will implement [15, 78]. In that situation the team can vote and agree together which to propose to be accepted by the stakeholders in a common planning meeting. The role of a scrum master is an important role in the team, but he or she is also usually acting as a developer or a tester in the team. Scrum master is responsible for ensuring that the team follows commonly agreed rules of the sprint and he or she also follows that the team stays in schedule. Figure 5 depicts a typical scrum team and the team has its own daily scrum meetings which are stand up meetings where the current situation is measured [14, 131-133].

![Scrum Team Diagram](image)

Figure 5. Scrum Team [15,101]

Scrum masters attend to usually daily scrum of scrum meetings when the project has multiple scrum teams and conducts the spring review meetings. Other people who want to follow the team’s daily scrum meetings are called as chickens. They are not allowed to speak during scrum meeting only observing is allowed. The team is responsible of their progress and they need to find resolutions to their own problems during a sprint. The team needs to replace ineffective scrum master if he or she is not effectively helping the team to remove obstacles, is not acting as firewall between stakeholders and scrum team, and does not provide resources when needed [15, 25].
The scrum master does not have authority over team members; he or she is a servant of them [15, 35]. Below diagram describes the roles in scrum team and also the roles of customer representative or shareholders and others connected to scrum team [14, 115-126]. Also it is good to have an elected scrum master who has both technical and managerial skills to handle the team and gives possible guideline to the team. And makes recommendation and suggest possible solutions to resolve blocking issues.

4 Software Testing

The penetration of software in industries, educational institutes, and other enterprises and organizations is a fact of modern life almost too obvious to mention. Nearly all of the businesses in the United States and in most parts of the world depend upon the software industry for product development, production, marketing, support, and services. Reducing the cost of software development and improving software quality are important for the software industry. Coded software always contains errors and the purpose of testing phase is to find the errors in the software. Testing is usually done on multiple levels (module testing, integration testing, functional testing, system testing, acceptance testing).

Scrum methodology has different types of testing are done in each sprint. There are other agile methodologies like Extreme Programming which also have similar testing practices. It is a software development methodology which is intended to improve software quality and responsiveness to changing customer requirements. This methodology is intended to improve productivity and advocates frequent releases in short development cycles, to enable checkpoints and to be able adopt introduced new customer requirements during development. There are acceptance testing, unit testing, performance testing and by-hand (or visual) testing. The division of responsibilities between the testing activities is made very clear [23]
4.1 Types of Software Testing

4.1.1 Unit Testing

A unit in unit testing can be a class or an object. Generally, and in programming languages that are not object-oriented, a unit could be understood as a specific idea and all the code that supports it [23]. Unit tests are often called developer or programmer tests to highlight the developers’ responsibility. The idea behind comprehensive unit testing is to build the developers’ confidence: the more tests there are, the safer it is to make changes to the source code. If, by default, all the tests pass all the time, one can immediately see if a change in the source code broke something. The tests should be isolated in the sense that they only test a very small and specific portion of the software. Unit tests should be kept quick to run: they should, for example, not rely on actual database connections. [24]

Test Driven Development (TDD) is a practice relays on unit testing (test-first) which means that the developer writes unit tests before he or she writes the actual implementation and it is a not new methodology. It mandates that a corresponding unit test must always be created before a single line of actual code and is almost as old method as programming is [25]. Test Driven Development is design and analysis technique which is a quality-first way of thinking and it means that the developer starts development of software by writing automated tests for almost everything first, which are called unit tests, and then writes the code (Szalvay 2004). Unit tests are part of quality control which creates to confidence on the written code [25]. This means that the developer which writing the test imagines the code and its functionality.

4.1.2 Acceptance Testing

While unit testing concentrates on ensuring the correction functionality of individual source code lines, acceptance testing is about making sure the software does what the customer has specified. Acceptance testing is done by or for the customer with the purpose of proving that the product meets the quality requirements and the functional requirements, which have been set. Acceptance testing is performed in an environment that represents the real usage environment of the product [26, 13].
It is done to verify that a feature has been implemented and that the implementation is done correctly. Manual and automated testing can be used in functional tests, based on the ideas from Crispin, L. & House, T.’s book Testing Extreme Programming, [27]. The tests are developed by the whole team in tight collaboration with the customer but there are always dedicated test person who handles the acceptance test collaboration. Unlike in traditional processes where acceptance testing is done at the end of a project, whiles in the agile mode, testing is done continuously while the software is being developed. One fundamental difference between acceptance and unit tests is that an acceptance test usually cannot be run before all the code required by the tested feature has been implemented.

Acceptance tests are black box tests that may access the system through a graphical user interface or other external interface. The functionality tested by them may span over several modules, units, or classes in the software. A single acceptance test may test anything from a simple business rule to an end-to-end functionality. An example of a business rule could be the discount given to loyal customers in an accounting system, while end-to-end functionality could mean entering sales figures in a web application and seeing the results in the reporting system. Acceptance testing also reveals requirement problems, such as system’s inability to really meet customer’s needs or unacceptable system performance [28, 449].

Acceptance tests can also be used in regression testing: they verify that existing functionality does not break when new features are added. The subset of acceptance tests used for testing previously completed features is called smoke tests. Smoke tests provide a wide but shallow testing of the system’s most critical features. The tests are usually quick to run, so that they can be incorporated into the process of building the software. While it is usually impossible to make all the acceptance tests pass during the development of the software, the smoke tests should always pass [27].

4.1.3 Performance and Manual Testing

Performance testing is done to measure the system’s capacity and performance. This kind of testing concentrates on verifying that the system meets the specified performance requirements.
Finding out the maximum capacity of the system and possible unwanted behavior under heavy load are also valuable information provided by performance testing. Optimization of the appropriate parts of the system can then be done based on the feedback from testing, and no time is wasted on speculative optimizations that may only consume time but provide no benefits [23]. Manual testing often means visually inspecting and testing the system. It is usually testing graphical user interfaces. Normally, there are look and feel standards (requirements) to which standard must be met during testing.

Manual testing can also be useful in so-called monkey testing, where the user attempts to break the system by trying out all sorts of valid and invalid inputs and unexpected use cases [24]. Another variation of manual testing is exploratory testing, in which the tests are usually performed without a tightly defined (written) plan [29]. However, exploratory testing is not completely unplanned ad-hoc testing: the degree of freedom may vary in exploratory testing, and some tests may involve more planning and constraints than others. Exploratory testing relies on the human tester's ability to learn and invent new things, and can thus be very useful in discovering new bugs in a semi-systematic way.

Exploratory testing can also be used to cover those areas where test automation would be too laborious or simply impossible. This level of testing should mostly concentrate on finding new bugs and testing things that do not require frequent checking. If bugs are found and corrected as the results of any type of by-hand testing, corresponding automated regression tests should then be created on the appropriate (unit, acceptance, or performance) level of testing.

4.2 Improving Software Testing

Organizations are looking for better ways to conduct software testing before they release their products. Innovations in the field of software testing have improved. Automation is one technique in improving testing. The techniques of writing test scripts, generating test cases, and introducing more advance automated testing can improve the software quality and delivery. Automation has both positive and negative effects to testing, and the both sides are considered in this chapter.
Automation is not effective method for all testing phases; execution and analyzing of tests are the phases most amenable to automate. Software must be tested to have confidence that it will work as it should in its intended environment. Software testing needs to be effective at finding any defects that are there, but it should also be efficient, performing the tests as quickly and cheaply possible [24].

Automating software testing can significantly reduce the effort required for adequate testing, or significantly increase the testing which can be done in limited time. Tests can be run in minutes that would take hours to run manually. Savings as high as 80% of manual testing effort have been achieved. Some organizations have not saved money or effort directly but their test automation has enabled them to produce better quality software more quickly than would have been possible by manual testing [24].


<table>
<thead>
<tr>
<th>Software Testing Tools for Test Automation</th>
<th>Skills Software Test Engineers need</th>
</tr>
</thead>
<tbody>
<tr>
<td>✅ Speeds up the software development life cycle</td>
<td>✅ Trainings from tool manufacturers needed for test engineers</td>
</tr>
<tr>
<td>✅ Finds as many bugs as possible before releasing the products</td>
<td>✅ Scripting and recording test scripts skills.</td>
</tr>
<tr>
<td>✅ Increase application reliability and cost effective.</td>
<td>✅ Ability to analyze and bug error in the tool or scripts</td>
</tr>
<tr>
<td>✅ To generate test scripts that simulates users operating the application under development</td>
<td></td>
</tr>
</tbody>
</table>

Table 3 is a summary of the benefits of automation tool and the skills required.

A mature test automation regime will allow testing at the touch of a button with tests run overnight when machines would otherwise idle. Automated tests are repeatable, using exactly the same inputs in the same sequence time and again, something that cannot be guaranteed with manual testing. Automated testing enables even the smallest of maintenance changes to be fully tested with minimal effort. Test automation also eliminates many menial chores. The more boring testing seems, the greater the need for tool support [24]. As seen in Table 3, test automation tools give great benefit to test but also require some specialized skill from the tester engineer to achieve.
4.2.1 Automating Software Testing

Software testing is an important part of the software development process. The purpose of testing software is to uncover mistakes in written programs and to confirm that the product itself satisfies the needs that have been given at the beginning of the software development process. Test Automation is the use of software to control the execution of tests, the comparison of actual outcomes to predicted outcomes, the setting up of test preconditions, and other test control and test reporting functions. Commonly, test automation involves automating a manual process already in place that uses a formalized testing process.

Test Automation is a quality method used very often in large and continuous development projects which include several releases in longer run. This description is a practical approach to test automation and it follows The Automated Testing Lifecycle Methodology (ATLM) which provides more guidance for this area. Ideologically building automated functional test cases happens hand in hand with the feature development and automating a test case is possible for components that include formal behavioral descriptions [30]. Test Automation usually includes unit, functional, acceptance and performance testing depending on what level of testing is needed in the project. The test automation system requires a framework which contains the written or recorded test cases. There are several different testing tools available and different types of testing needs different types of tools. The test cases have to be manageable, modifiable and run by the system. Complex testing is needed when the system under development is more complex.

The developer of a feature takes care of the unit testing. The developer can comment and help the functional tester by describing more deeply the feature which is under development if needed. Once the feature is ready the automated functional test case can be run and it has to pass before the feature can be claimed as finished in the end of a sprint. Functional test cases purpose is to test the systems features as the customer would use those [30]. Functional test cases can also try to break the system by doing steps in a wrong way as a possible customer could do those. Acceptance tests are usually designed either by the customer or by the functional testers.
Acceptance tests go through the systems basic functionalities so that the customer can accept the system to be ready for usage. Performance testing is done usually to systems which have to handle high load of users. The testing can be completely automated for maximum performance tests and for stability tests and usually performance testing uses different test tools than functionality testing. Well suited tools and competent testers are the key to make test automation to work. The benefits of test automation are discussed more deeply in the chapter 4.3 [31].

Testing is not easy work as many misleadingly may think. Instead, it is intellectual and creative work. Mark Fewster, 1999 [32, 5] observed that there are four attributes that describe the quality of a test case:

- **Effectiveness**: The most important of these is how effective defect detection is; a test case should be designed to find defects as effectively as possible.
- **Exemplary**: An exemplary test case could test more than one thing, thereby reducing the total number of test cases required.
- **Economical**: A good test case is economical to perform, analyze and debug.
- **Evolvable**: How evolvable a case is, means how much maintenance effort is required on the test case each time the software changes.

![Figure 6. Attributes of Software Testing [32, 5]](image)

A test case can be illustrated by considering the four attributes as shown in Figure 6. It shows the four qualities attributes of a test case. Whether a test is automated or performed manually does not affect its effectiveness or how exemplary it is. Automating a test affect only how economic and evolvable it is. When the test is automated for the first time, it is less evolvable and economical than a manual testing.
After the automated test has been run a number of times it will become much more economic than the same test performed manually. In average, 3-5 regression tests runs before paying back. Mark Fewster 1999 [32, 2-6] states that “The greater the measure of each attribute the greater the area enclosed by the joining lines and the better the test case.”

4.2.2 Software Test Activities

In this study, the software test activities are centred on Test Automation. The tasks of testing are planning the testing (doing a test plan, test cases), creating the test environment, executing the tests, evaluating the test results and verification (finding the faults). According to Fewster & Graham, 1999: [32, 6], the test process can be divided into a number of distinct activities, which may be considered the development life cycle of test cases. Figure 7 illustrates the software activities with emphasis on test automation.

![Software Testing Activities Diagram]

Figure 7. Software Testing Activities [32, 6]

The first activity is to identify test conditions: to determine ‘what’ can be tested and prioritize these test conditions. A test condition is an event that could be verified by a test. This is followed by designing test cases. A test case is a set of tests performed in a sequence and related to a test objective and test case design determines how the ‘what’ will be tested. The test should specify the preconditions, test steps, expected outcome. The actual outcome must be carefully examined and verified. Test cases are implemented by preparing test scripts, test inputs, test data and expected outcomes.
Building test cases in advance will save time later. The software under test is executed using the test cases. For manual testing, in practice, it is test engineer who starts to do testing following a documented manual procedure. He or she would enter inputs, observe outcomes, and make notes about any problems. For automated testing, this may involve starting the test tool and selecting the wanted test cases to be executed.

The actual outcome of each test must be investigated to see if the software being tested worked correctly. After execution test reports are generated and test case outcome is compared with the expected outcome. When the actual and expected outcomes do not match, for manual testing, test engineer will investigate the root cause of failure, while automation tool can only compare against the reference but cannot verify. It is therefore the responsibility of the test engineer to analyze the report and find the cause of the failure.

4.3 Benefits of Automated Software Testing

Test Automation can enable some testing tasks to be performed more efficiently than manual testing, and thus increase quality and productivity.

4.3.1 Software Quality

Running regression tests on a new version of a program is the most obvious task. A clear benefit of automation is the ability to run the test more often. This will lead to greater confidence in the system. Repeating the same tests so many times tempts every tester to skip most of the acceptance test, not paying careful attention to the results. By the time they repeat the same test, testers working manually are likely to miss obvious errors. Cem Kaner & David L. Pels 1998: [34, 197] observed that “a well-designed automated test might miss fewer errors than a small group of bored human tester”.

Automating boring tasks, such as repeatedly entering the same test inputs, give greater accuracy. Through automation the possible human mistakes are excluded.
4.3.2 Productivity in Software Development

Test automation increases the reusability. Some tests need to be repeated several times and in some tests the same inputs need to be entered repeatedly. If tests are automated they can be repeated exactly in the same way every time. This will give a level of consistency to the tests, which would take far more effort to achieve by testing manually.

Some test cases are difficult to test manually [35, 30]. For example, attempting to do a full-scale live test of an online system with hundreds of users may be impossible, but the input from hundreds of users can be simulated using automated tests. Test automation will lead to earlier time to market. Automation tool could be run 24 hours a day. Some systems even have multi-device functionality. One system can control several devices to run the case simultaneously. Once a set of tests has been automated, it can be repeated far more quickly than it would be manually, so the testing elapsed time can be shortened. In current severe mobile phone market, one day earlier to market can bring significant revenue and profit.

Unfortunately, though, test automation is no magic tool in the SW industry nor is it a wonder medicine to cure the lack of testing. Test automation just provides means to achieve some of the attributes mentioned earlier. What is to be remembered is that this is the case if and only if enough resources and employees with programming skills are assigned to the test automation task [35, 22].

4.4 Limitations of Automated Software Testing

Before implementing test automation, there are some aspects that need to be considered carefully:

- Skilled personnel or adequate resource for automation
- Significant maintenance costs
- Automated testing does not replace manual testing totally.
4.4.1 Lack of Skilled Personnel

Test automation is programming and thus it needs skillful people for making the scripts. It always beneficial to have a test automation specialist with the knowledge of both testing and programming skills because successful test automation requires some programming related concepts. Test scripts should be very re-usable and maintainable. One test case should perform one distinct task instead of many superficial tasks (high cohesion) and results should be independent of each other (low coupling) [36, 29]. The mentioned relation between test automation and programming implies that the test automation regime must be carefully designed.

4.4.2 Significant Maintenance Costs

Maintenance costs are more significant for automated testing than manual testing. A test tool does nothing before configuration is ready. After environment is set up, a lot of maintenance work is needed. For example, as the system changes some tests will become redundant. Some functionality may have been dropped from the product. Updating test case file or reference file needs a lot of effort.

4.4.3 Inadequacy to Replace Manual Testing

It is not possible to automate all tests. There will always be some testing that is much easier to do manually than automatically, some test case is so difficult to automate. For example, testing the suitability of a color scheme, tests that involve turning power off or on. Test that should probably not be automated include tests that are run rarely.

A tool can only identify differences between the actual and expected outcomes. That is why the verification of correctness of the result is difficult. The tool may indicate that all tests have passed, when they have only matched the expected outcomes. However, a human tester can use his or her creativity and imagination to improve the tests by noting additional things to test afterwards.
It is well known and agreed that Test automation does not improve the testing process, nor can it achieve better quality software [36, 367]. Rather, software quality can only be achieved by better specification, coding or by selecting better test cases. Testing is only a way to measure the quality, not a means to improve it. It is therefore necessary to know what and when to automate.

4.5 Decision Making in Test Automation

It is difficult to make decisions in automation. Cem Kaner 2009: [35, 282] observed that it could take ten times as long to automate and document tests as it takes to create them and run them once. There are contrary opinions concerning test automation timing:

- Test should be automated early;
- Test should be automated early in order to repay the cost of automation.
- Increase testing productivity during the peak bug-finding period.
- Create a standard acceptance test that will be run so often.
- Test should not be automated early;
- Test should not be automated early because early stages in development do not have stable environment
- The program will change too much at early stage. The program is unstable and subject to significant design change.
- It reduces available testing time during early testing. This delays discovery and correction of many errors.
- Some project needs only two or three testing cycles, payback usually after 3 to 5 testing cycles.[35]

4.5.1 Appropriate Task to Automate

It is not necessary to automate everything in order to get significant benefits from test automation. Pettichord Bret, 2003 [37] observed that if 10% of the tests are run 90% of the time, automating then alone might be worth doing.
There are some factors that could be taken into account when deciding what to automate:

- **Tests that are relatively easy to automate**: If the test is easy and possible to automate which might cost less than hiring a manual tester.
- **Tests that are run often (regression testing)**: Such as basic acceptance test which should be run every time for a new release.
- **Tests for the most important functions**: Main functionalities and requirements can be the target scope for automation. In the case of mobile UI testing, UI functionalities are set as most important, if there is error in the UI navigation, end-user would not be able to access or get the expected result.
- **Tests that is boring to run**: Testing how many times the application flicks in a day could be a good example to automate and not to be manually tested by man. It would definitely be a bore test case to test by human.
- **Tests that require detailed accuracy**: Automation tool may be seen as better than human being when deal with accuracy: For example calculating micro seconds of application launching and exiting. [37]

### 4.5.2 Appropriate Time to Automate

Much effort has to be spent even before test automation actually starts. Motwani 2001[39] summarized that the following assets have to be taken care of:

- Stability of product/application is ensured
- Scope of automation has been defined
- Individual test case to be automated has been identified
- The right tool has to be decided
- The right mode (script recording/script development) has been decided. [39]

The first aspect that needs to be ensured is that the software is fairly stable in terms of functionality and the test environment/tool is in good shape. Much maintenance work is needed if functionality and user interface change. Before deploying test automation, it is essential to define the scope coverage of the automation tool [40, 262]. The scope may include regression testing or acceptance testing.
It is also essential to prioritize the automated test cases. It is very important to include the right test cases in the suite. Automation suite should be looked upon as a baseline test suite to be used in conjunction with manual testing, rather than as a replacement for it. There are hundreds of automation tools available in the market. It should be carefully decided which tool would be most suitable for automating the testing.

4.5.3 Common Problems of Test Automation

There are a number of problems that may be encountered in trying to automate testing. Problems which come as a complete surprise are usually difficult to deal with, so having some idea of the type of problems which may be encountered will help to understand the characterization of automation better. Some of the most common problems are:

- Unrealistic expectations.
- Poor testing practice.
- Expectation that automated tests will find a lot of new defects.
- False sense of security.
- Maintenance of automated tests.
- Technical problems.
- Organizational issues.

There is a tendency to be optimistic about what can be achieved by a new tool. It is in the human nature to hope that this solution will at least solve all of the problems currently experienced. Vendors naturally emphasize the benefits and successes, and may play down the amount of effort needed to achieve lasting benefits [40, 263]. The effect of optimism and salesmanship together is to encourage unrealistic expectations. If management expectations are unrealistic, then no matter how well the tool is implemented from a technical point of view, it will not meet expectations.

If testing practice is poor and with poorly organized test set, with little or inconsistent documentation, and tests that are not very good in finding defects, automating testing is not a good idea. It is far better to improve the effectiveness of testing first than to improve the efficiency of poor testing. Automating chaos just gives faster chaos.
A test is most likely to find a defect the first time it is run. If a test has already run and passed, running the same test again is much less likely to find a new defect, unless the test is exercising code that has been changed or could be affected by a change made in a different part of the software, or is being run in a different environment.

Test execution tools are *replay tools* that mean regression testing tools. Their use is in repeating tests that have already run [32, 42]. This is very useful, but it is not likely to find a large number of new defects, particularly when run in the same hardware and software environment as before. Tests that do not find defects are not worthless, even though good test design should be directed at trying to find defects. Knowing that a set of tests has passed again gives confidence that the software is still working as well as it was before, and that changes elsewhere have not had unforeseen effects. Just because a test suite runs without finding any defects, it does not mean that there are no defects in the software. The tests may be incomplete, or may contain defects themselves. If the expected outcomes are incorrect, automated tests will simply preserve those defective results indefinitely.

When software is changed it is often necessary to update some, or even all, of the tests so they can be re-run successfully. This is particularly true for automated tests. Test maintenance effort has been the death of many test automation initiatives. When it takes more effort to update the tests than it would take to re-run those tests manually, test automation will be abandoned. Commercial test execution tools are software products, sold by vendor companies. As third-party software products, they are not immune from defects or problems of support [32, 42-63].

It is perhaps a double disappointment to find that a testing tool has not been well tested, but unfortunately, it does happen. Interoperability of the tool with other software, either application inside the company or third-party products, can be a serious problem. The technological environment changes so rapidly that it is hard for the vendors to keep up. Many tools have looked ideal on paper, but have simple failed to work in some environments. The commercial test execution tools are large and complex products, and detailed technical knowledge is required in order to gain the best from the tool. Training supplied by the vendor or distributor is essential for all those who will use the tool directly, particularly the people who automate the tests.
In addition to technical problems with the tools themselves, it is possible to experience technical problems with the software to test [32, 22]. If software is not designed and built with testability in mind, it can be very difficult to test, either manually or automatically. Trying to use tools to test such software is an added complication which can only make test automation even more difficult. Automating testing is not a trivial exercise, and it needs to be well supported by management and implemented into the culture of the organization. Time must be allocated for choosing tools, for training, for experimenting and learning what works best, and for promotion tool use within the organization.

Test automation is an infrastructure issue, not just a project issue. In larger organizations, test automation can rarely be justified on the basis of a single project, since the project will bear all of the start-up costs and teething problems and may reap little of the benefits. If the scope of test automation is only for one project, people will then be assigned to new projects, and the impetus will be lost [32, 25]. Test automation often falls into decay at precisely the time it could provide the most value, for example when the software is updated. Standards are needed to insure consistent ways of using the tools throughout the organization. Otherwise every group may develop different approaches to test automation, making it difficult to transfer or share automated tests and testers between groups.

In the past, most of the GUI automation tools used script recording feature. They record and playback what tester did. It then generates a script that could be run to re-execute the test. Script development, on the other hand, implies writing the scripts in the language used by the tool. Using the recording mode alone will lead to the automation suite non-reusable and using the scripting mode alone will require more investment of effort and time. It might be worthwhile spending some time to decide the right mode or right mix of modes [32, 517]. There has been recent advancement in the GUI &UI automation tools and none of them can do without scripting.
5 Techniques in Scripting

The scripting techniques play a major role from the commencement of the project to the maintenance stage. Some scripting techniques automatically drive the test cases to modular and maintainable structure [32, 75]. In this study, three different scripting techniques are analyzed: shared, data-driven and keyword-driven scripts.

5.1 Shared Scripts

Shared scripts technique follows normal procedural programming – the test scripts consist of conditional logic, loops and function calls that are procedural scripts performing some common operation. The name ‘shared scripts’ is derived from it process of usage. The automated test cases (the actual test scripts) can use the shared scripts to perform some operations [32, 79]. These shared scripts should compose a script library which the actual scripts should use.

The evident advantage of shared scripts performing the common operations involves maintenance – changes in SW propagate only to minimal number of scripts, usually to one shared script, since the duplication between scripts is minimized. The most important drawback of the shared scripts technique is that the dependence between test cases and the test scripts is linear, which is a major cause for high maintenance costs, since the more scripts used the more time maintenance takes [32, 192].

5.2 Data-Driven Scripts

Data-driven scripting technique differentiates from shared scripts by separating input data and the logic of the script. The data is stored in a separate file from where the control script reads it [32, 83]. The immediate benefit of this kind of approach is that the particular test script can be used by more than one test case breaking the linear dependence between test cases and test scripts. The additional test cases are also easier to add, since only a new test data file must be written. The drawbacks are that higher initial effort and programming skills are needed and that test automation must be well managed as a whole [32, 89].
5.3 Keyword-Driven Scripts

The keyword-driven approach is an improved technique from data-driven. It also separates test execution logic from the test data, but it makes it easier to add new test cases. There exists only one control script and it is a simplistic version compared to the data-driven technique, since it only redirects the control to the correct support script. The support scripts, called keywords, perform all the executions. The input of each test case is written into a test file containing keyword-parameter pairs that correspond to any particular test step implicitly defining each test case.

Figure 8. Types of Automation Script [21]

The keyword-driven approach has multiple advantages: Changes in the SUT has an impact on only a few scripts (keywords) reducing the maintenance needed, changing the test execution tool to another causes only a rewriting of the keywords (to support the other tool) and that the test cases can be written by testers having no automation skills. Also the number of keywords is dependent on the size of the program thus adding new test cases can be added by writing new test files [32, 91]. Out of the three techniques discussed, it is generally admitted that the shared script is the easiest to learn, the hardest to use and maintain, and the least scalable. While the keyword-driven scripting technique is the hardest to learn, the easiest to use and maintain, and the most scalable and reusable, as seen from the diagram in Figure 8.


6 Reporting of Results

Reporting agile testing results does not focus only on the number of test cases executed or defects found but also on the business risk level of the product. The test reports play a major part in test automation, since practical reports might save a lot of time when investigating the execution results. This is due to the fact that in an immature automation regime one fault might result in dozens of errors and investigating all the errors would be unnecessary and inefficient. It is very essential for test reports to have Definition of Actual /Expected Outcomes. The actual outcome and the predefined expected outcome should always be stated in test report in order to give clear understating to the report. The tester is able to analyze if the test failed was as a result of false negatives, i.e. a non-bug reported as an error [4, 17].

A statistical analysis is a must feature that the tool must be able to generate. These statistics must at least include the number of passed and failed tests and the total number of tests that were run. These statistics should then be transferred to the test round statistics used to keep track of the progress of the testing activities. The requirement for automated statistics rises from the testing process, which states that some metrics must be provided to keep track of the testing process and the system maturity. Remember, face value is not enough; metrics must also be used when giving report to the developers and business stakeholders.

Reporting the results does not mean creating thick reports at the end of the sprint. Rather, reporting should be continuously and it should be done automatically while the work is done [4, 22]. Timely feedbacks are relevant to software development. With a good tool that has been correctly configured, automated statistics can be generated and helps to reduce work load significantly. During the sprint and at the end of each sprint, the quality and business risk of the product must be reported. The stakeholders need the report or information to make decisions.
6.1 Feedback to Developers

Giving feedback to developers can be done in various ways. In small projects, face to face communication is often the best way, with possibly some written documentation. As the scale grows, more and more formal communication is needed and even a traditional defect tracking system is a good option. Counting defects and reporting the number of defects by status would not be a major aspect to follow in an agile team.

6.2 Business Risk Level Report to Stakeholders

Reporting the business risk level is based on the feature risk analysis. The features with the most potential damage and a great chance of failure bring the most business risk. The business risk level can be reported as the number of test cases passed compared to the total number of test cases planned to be tested. If the test cases are classified according to their criticality, a more detailed analysis can be derived.

7 Case Study: UI Testing Using Agile Processes and Methodologies

7.1 UI Testing System Overview

Innovation in the field of software testing have proved the techniques of writing script, generating test cases and introducing test automation for software application testing. Software testing has evolved over the years to incorporate tools for test automation. The purpose of these tools is to speed up the software development lifecycle to find many bugs as possible before releasing the products. This is to reduce cost of software development by automating effective regression testing and to increase application reliability [36, 2].

UI software testing is the process of testing a product that uses a user interface, to ensure it meets its written specifications. This is normally done through the use of a variety of test cases, verifying how the application and the user interact and if it meet the application requirement. For example, how the application handles keyboard and mouse input and how it displays screen text, images, buttons, menus, dialog boxes, icons, toolbars etc [36, 4].
7.1.1 Nokia Differentiating Software (DSW) in Agile Mode

Nokia advocates the use of agile processes and methodologies in all software development projects and teams. This is because Nokia perceives agile to have a faster value, and to be responsive, innovative, productive, transparent and above all less risky. There is no need to waste time on extensive pre-planning but quick to react to changes in business requirements and bringing innovative ideas to the table. The project team and business should co-operate to find the best solutions. There is less documentation and managerial overhead, and the project outcome is always demonstrated after each sprint (DEMO).

The Nokia Differentiating Software (DSW) is a sub-entity responsible for creating a competitive product and Nokia differentiating SW on top of shared assets in order to enable the creation of Nokia Symbian devices with world class consumer experience. In fulfilling this mission, the differentiating software sub-entity enables product-driven SW differentiation (including relevant UX design, technical design, SW development, and SW validation, pre-integration and releasing activities) by adopting the agile processes and methodologies. Differentiating Software has strong local presence in all product creation sites ensuring full product SW concepting support and on-site visibility for product driven SW development - while maintaining also global operational efficiency.

7.1.2 Validation and Pre-Integration Team Charter

The purpose of DSW Validation and Pre-Integration is to provide verified and pre-integrated high quality DSW releases to product programs and families. Validation and pre-integration also provides error management interface to customers and ensures error correction and fix releasing. Successful operation will require close cooperation with DSW development team, delivery operations and product programs. With regards to this study, our focus is on how UI testing is done in the Nokia DSW with much focus on Nokia DSW Validation & Pre-Integration.
7.2 Nokia Agile Testing Overview

Agile testing is part of agile development. In agile development, a product is developed in short cycles, typically four weeks. After every cycle, the customer is provided with a potentially shippable product which provides him business value. Tester works as member of the development team from day one and the scrum team has shared responsibility for quality. The quality of the released product is assured with adequate testing by the whole team, not just testers. Test levels are concurrent; the whole team participates in testing.

Figure 9. Example of Agile Project

There is a continuous feedback and reporting and these feedback loops instantly from testers to developers. As shown in Figure 9, agile testing has to adapt to the short release cycle of agile development and in each release cycle, items are taken from the product backlog to the sprint backlog for development.
7.2.1 Agile Team Testing Skills

The scrum team has shared responsibility for quality and testing. Therefore it is not possible to define testing skill requirements for individual team members - as long as the team has the required abilities. Nevertheless, to assume test-related roles, the team members must possess certain testing skills - e.g. anyone who takes the role of an automation tester, should have the necessary expertise. In order to be effective in testing, all the scrum team members must have a basic understanding of the agile testing practices. The more expertise there is in the team, the more effective it can become.

Although there are no fixed roles in an agile team, members of the team with great expertise in some field can take up some roles to foster productivity and effective execution of task. Testing techniques should be one of the areas where the tester should be able to help the team most. In agile there is no extensive test planning, test preparation or test execution phases. The tester must have the ability to apply different testing techniques in innovative and creative ways in order to understand the vital testing goals and apply the most suitable testing techniques to reach these goals.

With regards to automation in agile mode, it is important for the tester to understand the development process to be able to optimize the whole testing process. The tester must have the ability to apply test automation where it best fit and understand where test automation should be applied and how. In agile there usually is no extensive phase for creating extensive tool support for testing. The team must understand what the most important tools are and how they can be applied in different situations. The agile tester has the possibility to apply the traditional testing techniques and skills in more innovative ways than before. He must have the ability to apply testing tools in innovative and creative ways. In addition, he must possess new skills, and most importantly; he must have an agile mindset.

7.2.2 Test Automation process in Nokia DSW

Efficient agile SW development requires the use of test automation. Transformation to agile usually starts from unit testing. However, it should not end there as also other testing needs automation.
The DSW Validation and Pre-Integration have been running most of their UI test cases manually. For this case study, the team selected one project to transform the project to a full agile project since most of the projects are not fully in an agile mode. Nokia IT agile test process strongly recommends automating cases as much as possible.

Automated UI test cases have not been implemented due to a number of reasons. In order to begin the process of automating UI test, the DSW and entire Nokia follow some generic guidelines from Meszaros et al. Test Automation Manifesto [42]. Based on many years of building and maintaining automated unit and acceptance tests, Meszaros et al [42] created their Test Automation Manifesto. The Manifesto contains good advice to remember as one creates automated tests.

- Concise – Test should be as simple as possible and no simpler.
- Self Checking – Test should report its results such that no human interpretation is necessary.
- Repeatable – Test can be run repeatedly without human intervention.
- Robust – Test produces same result now and forever. Tests are not affected by changes in the external environment.
- Sufficient – Tests verify all the requirements of the software being tested.
- Necessary – Everything in each test contributes to the specification of desired behavior.
- Clear – Every statement is easy to understand.
- Efficient – Tests run in a reasonable amount of time.
- Specific – Each test failure points to a specific piece of broken functionality (e.g. each test case tests one possible point of failure).
- Independent – Each test can be run by itself or in a suite with an arbitrary set of other tests in any order.
- Maintainable – Tests should be easy to modify and extend.
- Traceable – Tests should be traceable to the requirements; requirements should be traceable to the tests. [42]

Most of these characteristics are further analyzed during implementation and completion of the automated project and not just from the onset.
7.2.3 Test Automation in the Agile Mode

Agile test automation is the application of agile development principles to the test automation problem. As James Bach states, "the gross summary of the method is this: instead of establishing a team of test automation experts who create a huge test automation framework that takes months to make work, identify at least one test tool smith who pairs with testers and in less than 40 hours (preferably less than 16) identifies and implements some technology that helps make that tester more productive. Then repeat. Think of test automation as any tool supported testing, and think of test tools as any tool that may help (especially cheap or free tools). Avoid expensive tools, such as those from IBM/Rational or Mercury". [29],

7.3 Nokia DSW Agile Test processes

The test process flow gives an overview of how the different practices of agile testing are applied, as illustrated in Figure 10.

![Figure 10. Overview if Agile Testing Process [41]](image)

From the onset, sprint backlog are required to be created. Sprint Backlog is made during sprint planning meeting. Test planning begins in the sprint planning meeting. Before committing to the sprint backlog items, the team must have at least a rough idea of the effort, tools, environments, test data and people required for testing. During the sprint planning meeting, the product specifications were discussed and activities for the sprints were laid out in brief.
The test team then deliberated on ways and estimated roughly how much effort would be need to execute the task. Before rushing into writing and automating test cases or exploratory testing, it is a good idea to use some time for planning and defining the test approach. The scrum team discussed what test levels are necessary and what features are prioritized. There was arranged a workshop where automation expects were present. Test conditions, Testability check and Feature risk analysis were analyzed during the workshop and it was planned to have biweekly status meeting during implementation.

After planning, there is deployment and implementation. In component creation and testing, continuous testing approach is upheld for every release as work is in agile mode. The goal of this practice is to produce good quality right from the start - and not to try to stick (test) it on top of the product at the end. In this pilot project, Automation manifesto [42] was used and it was agreed to automate the BAT cases based on the user stories. In order to decide when the user story is done one should be able to verify the completeness of a user story. This leads into thinking about the testing while creating the user story which can be only a good thing. It is recommended for the testers to work together with product owners and fine tune the user stories so that it is testable. INVEST approach to user story design which means; Independent, Negotiable, Valuable, Estimable, Small, and Testable should be used. [42]

Being able to test the user story through automation aids ensures that validation of user stories and specifications of the program are basically covered. It was intended to implement the automation directly based on the old manual test plans which were planned based on the user stories. This did not work very well since not all the user stories were testable through automation and the tool had some limitations. Some specific rules needed to be created for the test plan. It is always likely to modify pre-defined test cases a bit, if a manual test plan has to be used in test automation. It was noted that the tester must have a possibility to split or restructure the test case to better suit the idea of automating. A rigid and unchangeable test plan does not suit or work in agile mode, there must be some freedom of action. I was familiar with the idea between automation of testing and the test plan and was able to improve the test plan, since I was the writer of the test plan for the application.
The most important points that one needed to emphasise was that each test case must have only one objective, and the expected results need to be described thoroughly. Expected results are as important as ever, since the tester or writer of automated test cases can not guess the expected result of certain actions. In automated testing, the expected result must be told as explicitly as possible. For example, in a program where the icons of visible list should correspond to the list name, there exists an option to verify the select list correspond with the icon shown on the screen. In this case, the functionality known as verify is used in the automation script.

Continuous integration and test automation has a strong role and is an integrated part of software testing activity in Nokia. It defines the quality of the product by regressively testing the areas in a short time. Proper guidance, strategy and framework make the automation effective and right investment for testing. Test automation means tool support for all aspects of a test project, not just test execution. One of the goals of agile testing is to automate as much as possible, where it fits. With continuous integration, only a small amount of new code is introduced to the code repository at a time.

When regression testing is executed with every build, there are only small changes that are tested each time. This makes managing defects very simple. The found defects are instantly communicated to the developers and fixes are done right away. During the sprint, the team strives at automating as many tests as possible, thus continuously improving the coverage of automated regression testing. In order to achieve this, the team had to finally decide on the tool that best fits the purpose of automating the application.

7.3.1 Test Automation Tool Selection

When selecting test automation tools, it is recommended to focus on the people aspects first - to make sure the iterations and the retrospective are working. Upon the retrospective based adaptation, the team should be able to select the tools they need. Agile testing requires new tools and the best tools are the ones user has adequate skills and the tools should to be adaptive.
The effective agile test automation tools should support commencement of the test automation effort immediately, using a test-first approach and encourage good programming practices for the code aspect. The tool should support writing test automation code using real languages, with real Integrated Development Environment (IDEs). It should separate the essence of the test from the implementation details and foster collaboration.

A small sampling of agile-friendly tools and frameworks is summarized in the following:

- Ward Cunningham’s original Fit has inspired a whole bunch of related tools/frameworks/libraries including Fitnesse, ZiBreve, Green Pepper, and StoryTestIQ.
- Concordion takes a slightly different approach to creating executable specifications where the test hooks are embedded in attributes in HTML, so the specification is in natural language rather than a table.
- SeleniumRC and Watir tests are expressed in Ruby; Ruby makes good DSLs.

![An Example of UI Element on a Nokia Product](41)

Figure 11. An Example of UI Element on a Nokia Product [41]

An example of UI Elements on a Nokia product is shown in Figure 11, and for automation of UI testing in Nokia, Testability Driver known internally as MATTI (Mobile Automation Tool for Testability Interface) is used. Matti is a generic driver which provides access to Qt and symbian software for test automation harnesses.
Matti tool can be useful for testing components which invoke UI and unit-testing of that component would require considerably lot of work to simulate user interactions. MATTI is an interface for both precisely controlling a system under test with inputs and gaining direct visibility access to the user interface components for output. Matti aims to bring Nokia product testability to the same level as found on desktop systems.

Although in S60 Avkon there is no simple way to traverse all UI components for any given application, there is therefore a special mechanism to hook the UI objects was created. This mechanism allows storage of the information regarding the type of the component. Using the type information and pointer into hooked UI object we are able to traverse the UI and read the information for all UI components. With MATTI, the SUT is presented to the developer as a set of objects that can be verified and manipulated as needed, and are synchronized to the status of the SUT.

**MATTI and Open Source**

There are three key tools for Test Automation [39]. As shown in Figure 12, they are the Language, Driver and Harness. The Ruby language is used for interacting with MATTI and MATTI driver itself is also developed in Ruby. As such, automated tests based on MATTI can be executed on computers using a variety of Ruby-supporting operating systems, such as Windows, MacOS or Linux. While MATTI is designed for mobile applications, there are also no limitations on the type of the system under test.

![Diagram of Matti and Open Source](image)

Figure 12. Matti and Open Source [41]
Visibility and control data channels needed for MATTI operations to run on all systems. Examples of possible SUT types are S60 and QT on various platforms (Maemo, Symbian, Windows etc.) Figure 13 depicts the relationship between events and UI elements of the SUT and MATTI test script code. No changes to the tested applications are required. Test scripts that make use of MATTI can be developed in any development environment that makes Ruby use possible, for example Carbide or Eclipse with the RDT plugin or the editor provided with the Ruby installation package. Test scripts can be executed directly from the IDEs or from a console, also by other scripts. There are two parts to MATTI.

![Figure 13. Matti components and their relationship](image)

Figure 13 displays the principal MATTI components and their relationship. There is a host for executing tests. The host might be the same PC used for creating test scripts. The host creates and sends commands to the SUT in accordance to the test script and interprets results. A plugin service is also executed in the SUT. It gathers visibility data, translating it into a common format used by MATTI. The service is also responsible for translating control actions into interaction with the SUT, such as pressing a key. There are a variety of ways in which these two parts can communicate.
7.3.2 Scripting and Maintenance

Before scripting commenced, I had to bear in mind that easy maintenance of test scripts is as important as easy maintenance of the program code [32, 191]. As one begins to script, it is crucial to consider the maintenance of the test cases and the scripts. It might be even more important, since every change in the program’s functionality might cause changes in one or, in the worst case, in multiple test scripts. The goal of easy maintenance is that one change in the functionality of the program results in one or, at the most crucial changes, in few test scripts. Thus, reducing the workload of maintenance should be taken as a real challenge.

To achieve the easy maintenance, the scripts should be independent. It can be achieved by multiple ways, some of which are similar to the normal programming: Creating a test harness or in simple terms a kind of wrapper which is used to redirect the commands to the actual test tool, or an abstraction layer to define logical names for physical UI elements [32, 113-114, 534; 34, 23]. Another solution to decrease maintenance is to use data-driven or keyword-driven scripting techniques, possibly combined with an abstraction layer. This approach has also similarities to the normal programming, namely to the bridge design pattern.

The most important thing to remember is that the maintenance of the automated test cases is more significant than in manual testing because the test automation tool has no intelligence to decide any details in runtime – everything must be explicitly defined beforehand [32, 191]. A manual tester can adapt to new interface changes, but the automation tool does only what it is instructed to do, it does not automatically notice changes in the interface nor can it adapt to those changes. One of the most common changes in the S60 phones is a changed menu in some application. A test automation expert must be aware that it can happen also dynamically, that is in a certain state there might be different menu items than in another state.
7.3.3 Running the Tests

After the scripts were made, test cases were able to run without human supervision. The tester is able to leave the Matti tool to run the selected test cases independently during night hours or other free hours. The results are collected to a separate result file, where it is possible to detect the possible error points. One important part of the tool is the Recovery System that makes it possible to reconnect the tool when connection to the mobile phone is lost. Also when there is a case that is unable to run, the script is written such that after unsuccessful attempts to run a particular case it continues with the next test case, even if something went wrong with the previous test case.

Each test run or event had a timeout if 30 seconds. When nothing happens after waiting for 30 seconds, the test case fails. After the timeout, the next test case is available to be run. It was not productive to supervise the testing tool and therefore I made some modifications to the tool to send an automatic notification e-mail to me to inform me when all of the test cases to be run are complete. One point that was discovered by running the same test cases repeatedly was the load effect to the application itself. The application started to slow down, which led to the fact that at some point it would crash. The reason to this was probably some memory leak on SUT, a poor network connection since the application depended heavily on the network, or bad performance of the device due to inconsistency of stability of the platform during testing.

7.3.4 Reporting

Reporting the results does not mean creating thick reports at the end of the sprint. Rather, reporting should be done continuously every day and it should be automatic as the work is done. With good tools and the right setup of these tools it is possible. During this pilot project, it was noted that reporting was limited to only the mail notification and manual verification of results by the tester. Based on this observation, I proposed that the result be automatically generated to the main testing reporting tool Quality center. A manual verification of the results should be done before it is uploaded to the reporting tool. There could be another script run after the tester verifies that the right result was generated.
7.4 Discussion of Test Automation in Nokia DSW

7.4.1 Advantages and Disadvantages

The test automation was usually viewed as a positive technique but very little has been achieved from automation in the organisation. The problem seems to be that the starting of the test automation to a particular application was seen difficult with many blocking or dependency factors making advancing of test automation to be very slow. The testing tool selection is also an important task, since it affects the whole success of the test automation. However in the case of a big company like Nokia, tool selection normally comes for the main test automation team. The main team would provide support to the recommended tool and would not give much support to other tools. It is sad but true that these tools have some revolution for a certain time frame, and there come newer tools.

People are supposed to learn to use the tools and get some training and time to focus on test automation but frequent changes in preferred tools does not help the entire organisation. Tool selection depends on the SUT and about the resources that one wants to put to a test automation tasks. The test plan helps with the writing of the manual test case to automated test case. Earlier, it was noticed that the best way to do the test plan is to take test automation into consideration when writing the test plan. The manual test case description should be directly written to automated test case. This also may depend on the test case description and the requirements of the testing environment.

The development group should have some knowledge of the automation of testing. This will enable the group to give some support to the automation team. They know what kind of problems may be invoked by the application and how to handle them. Developer creates module testing, and they could easily create automated test cases, when the framework has been done. Test scripting is not easy but worth learning and trying. The execution of a number of test cases out-of-office hours, without supervision of human effort, was very beneficial. This saved the time for the creator of the test scripts and freed the machines to the daytime for other testing activities.
After running a number of test cases, the next action to take was for the tester or script developer to check the results and distinct the test script bugs from the application bugs. Test automation reveals unexpected errors - and therefore adds a new view to the testing of the software. This also covers the problem of the intuitive testing which by Kaner (1992) is missing from the Test Automation. Quite a number of the errors are found when writing the automated test scripts. The unexpected errors are those that are errors that can be found, although another aspect of the software was under test.

Writing of test scripts is easy and fast for person with programming experience. Automation could be used as a bridge between programmer and tester, because the tester has to have knowledge of application domain and also about testing methods in general. There are also disadvantages and inefficiencies caused by the tool. The main problem was that the tool could not test everything but that I also to remind us all that test automation is a mean to achieve quality but not an end to the mean of achieving quality. This is why in the DSW manual and other special testing are also very crucial in the testing processes.

7.4.2 Manual and Special Testing

Usually, some manual exploratory testing is required to complement automated testing. In other cases, user acceptance testing or usability testing with end-users might be necessary. Performance testing can be planned and executed either as a service from outside the scrum team, or by having a professional performance tester in the team to help out. We could see that we definitely need to complement automation with manual and this brings us the issue of the pros and cons of manual and automation testing. The question to be asked is when to compare manual and automated testing, features are comparable and needs to be compared objectively. The focus is on some important aspects of these both testing methodologies and their differences are viewed.
Bach, 1998 [29] considers the following assumptions with doing automation of testing:

a. Manual test case can be mapped to a sequence of actions that can be described to the computer. However, this lacks the intuitive search method that experienced testers have in manual testing.

b. The sequence of actions is useful to repeat often. If case is run once, and everything is ok, then the case will not reveal any error if it is run again. However, since a human probably will not run the test case exactly like the first time, which means that there is some variability which means that something different, has been tested, so the coverage is better in manual testing. In addition, the variability of humans may not be obtained, because usually it is enough to run the test case once, and then move to the next case, so actually this benefit of human testers cannot be obtained. Actually in manual testing you execute the same test many times, while developing the test case.

c. The sequence mentioned above can be automated. Some things are easier to do by humans than computers. For example, interpreting the test results and analyzing.

d. Once something is automated, then it can be done faster, because no human effort is needed. But bear in mind that the results have to be analyzed and the scripts may need updates periodically.

e. Human error is diminished. Some errors are but the automation project may have its own, one could call them test bugs, this would require that the outputs of test automation are reviewed in order to guarantee their quality.

f. Comparing the manual testing with automated testing money-wise is impossible, since they are two different processes.
g. The value of automated testing is higher than of manual testing. Only rarely and if everything works as it should.

h. The cost to automate the testing will be less than three times the cost of a single, manual, pass through the same process. The cost of automation of testing depends on many factors, about the SUT, test tool used and resources available. [29]

It is often mentioned that a great deal of errors is found outside actual test cases, this means that all testing cannot be automated, because the intuitive thinking of a human in testing would be lost. The second point mentioned is a two-handed sword. The test cases should be run similarly every time, so that testing environment is the same - however, if this is true, then it is enough, just to run the test case once and be happy with the results. The only action to take in this case is to wait for the repair of the test case and do regression tests to the targeted application. However, the automation has to be done with quality - the secret is to think about testing first and automation second.

7.5 Recommendation for UI Agile Testing

During the development of the test automation I noticed some points that could be done differently next time to improve UI testing. Some issues concern to the responsibility issues and others to the problems that were raised by the application. One issue was that the synchronisation of the execution of the test cases would improve the execution of the test cases considerably. At some point, I noticed that the manual testers were doing the same test cases, that I had executed automatically a couple of days ago.

Clear division of responsibilities is needed and the automated test should be trusted and results should be made visible to the public by the scripter and also to avoid duplication of work. In Nokia, a specific tool is used in reporting faults and execution of certain test cases, but as mentioned early in chapter six, it would be rewarding to have another script that would upload the result generated from the run cases to the reporting tool. This would reduce redundancy in testing.
Most of the applications developed by the DSW team are interdependent to each other; it is advisable to co ordinate test automation activities much effectively and reuses the scripts developed by testers (Automation experts). No matter the state of the application, it is quite probably the application will change slightly during the development of the test scripts. If the application changes a lot, then the frame and the test data has to be verified each time manually.

If the results of previous test execution runs are saved, then probably all that is needed to do is to, is to execute every test case again, and detect where the root causes are. The cause may be in the application or in the framework of automation. *Deliberate* verification of the corrections was randomised. I executed all of the test cases at frequent intervals, but it would have required some extra effort to find out whether some correction was done and when. However, regression testing for cases that failed in last test run is possible, because the tool reports all errors and gives reasons to the failure. When the test run is done from the test plan directly - then you can see directly the status and execute those test cases that belong to the regression test set. The verification was done, but at certain cases it was noticed, that in fact there was an error in the test data, which caused that the test case failed, although the application work as specified.
8 Discussion

In this paper, a critical analysis on how DSW Agile UI Testing was made with much emphasis on agile test automation to ascertain the current state and determine ways to improving the organization’s test processes. With my programming experience, although quite limited, I noticed that if the right approaches and analysis for testing is not considered, testing of software may be a very time-consuming and tiring task. The simplest type of test automation such as verifying icons and titles might either be challenging or very efficient depending on the state and stage of the software development. Regression testing on a stable application in an agile mode with the help of simple scripts to verify features in the application is made much efficient when automated.

During the development of the software, the test run scripts were very useful by revealing errors in the regression sense. They verified that changes made did not break the application or affected anything elsewhere. From this experience in this project, I have got much interested in test automation and have learnt efficient ways to conduct UI testing in more complex software environment. Agile Testing is quite an interesting aspect of the agile software development. Therefore the need of critically analyzing what and how to test in agile mode is crucial in software development.

Different testing and scripting techniques have considerably discussed in this paper. To ensure productivity, quality and efficiency in software delivery, one has to understand the current situation and the right choices to make. Many writers referred to in this paper have recommended that agile test automation is very useful for testing while testers and developers should not disregard other types of testing especially, manual testing. It is clear that the tools cannot replace the human effort. They can be supplemental or partial and the obvious benefits in agile testing: consistency, speed and accuracy in running and reporting of the test cases. In addition, the test planning has to be done by a human. The current tools are not sufficient and the application domain knowledge is essential in planning of testing. Test automation provides some answers to the problem of software complexity. Software that consists of several lines of source code is quite hard to comprehend. However, it may be seen as a simple UI effect such as a flash on the screen in some micro seconds and hard to test manually.
9 Conclusion

The objectives were to evaluate the most benefit gained and the limitations of test automation and to enhance testing by augmenting manual testing with agile automation testing. The study aimed to concentrate on how to improve the efficiency and effectiveness of UI testing. A pilot project was carried out in Nokia DSW agile team and a critical analysis on how DSW Agile UI testing was made with much emphasis on agile test automation. It was observed that there were missing processes that makes a team agile. The study ascertained the current state of the team. Based on literature reviews and industrial experiences with proven success stories, the most appropriate ways to implement agile ideologies to achieve significant result from UI test automation were observed and taken into use during the implementation of the pilot project. The goal for the study was achieved and ways to improving the organization’s test processes were determined and recommended.

During this project, answers to two key questions emerged. When can test automation be done and how can the UI test automation be done? When test automation is best suitable for the project and not necessarily at a specific state in the development, also how to automate UI test is having the right skill, tools and environment that users can easily adapt. In an agile project, people have a greater tendency to have a must of having test automation as part of agile testing. We came to a conclusion that there are general benefits gained and also limitations of test automation, such as the speed of the software development which was very slow at the early phases of development but accelerated in the later part. UI test automation is a very new area of software testing and there is much room for improvement.

Agile is getting things done and achieving results. To achieve the set goal in agile testing, the need for automation, how to automate, with what resources and when to automate must always be considered. Projects vary and it is appropriate for a team to find their own appropriate time to start test automation and implement base on the resources and skills. Agile testing is not mainly test automation. Rather agile testing is enhanced by augmenting manual testing with agile automation testing. This study has deeply enlightened my knowledge in agile testing and automated UI testing in an agile project.
References

1. Cohn Mike User Stories Applied For Agile Software Development. Publisher Addison Wesley; 2003

2. Craig Larman, Bass Vodde Scaling Lean & Agile Development: Thinking and Organizational Tools for Large-Scale Scrum; 2009


URL:http://danube.com/news/Scrum_An_Agile_Development_Favorite 
Accessed 24 February 2011

10. Cockburn and Williams. The Costs and Benefits of Pair Programming 
http://collaboration.csc.ncsu.edu/laurie/Papers/XPSardinia.PDF 
Accessed 24 February 2011

11. Mike Cohn. Agile Estimating and planning 
Publisher: Prentice Hall; 2005

http://www.agilemodeling.com/essays/activeStakeholderParticipation.htm 
Accessed 14 June 2010

13. Westerhoff [Online] URL: 
Accessed 24 September 2008

14. Craig Larman. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd edition); 2004


Publisher Pearson Education Inc; 2004

17. Amsol. Agile Development [Online] URL: 
http://www.amsolinc.com/Amsoladvantages/AgileDevelopment.aspx 
Accessed 10 December 2010


28. Sommerville I. Software engineering (5th edition.)Publisher Addison-Wesley, Reading, MA; 1996


32. Fewster Mark & Graham Dorothy. Software Test Automation: effective use of test execution tools. Publisher Addison-Wesley; 1999

33. Mark Fewster and Dorothy Graham. Software Test Automation ; 1999


42. Meszaros, Smith et al., The Test Automation Manifesto. Extreme Programming and Agile Methods XP/Agile Universe; 2003
Check List for Agile Testing

Testability check

Quality of the backlog items
Are the backlog items documented at a consistent and appropriate level of detail?
Does the scrum team have a common view of the functionality of the backlog items?
Is expected behaviour documented for all anticipated error conditions?
Does the scrum team have a common view of the non-functional requirements for the backlog items?
Are all performance objectives properly specified?
Are all security and safety considerations properly specified?
Are all usability requirements properly specified?
Is each backlog item written in clear, concise, unambiguous language?

Arrangements needed for testing
Does the scrum team have a common view of the test goals for each backlog item?
Does the scrum team have a common view of business criticality of each backlog item?
Does the scrum team have a common view of how to test each backlog item: e.g. dynamic testing, demonstration or review?
Does the scrum team have a common view of what test levels are needed to test each backlog item?
Does the scrum team have a common view of how much effort is needed to do test planning, preparation, execution and reporting?
**Competence check list**

- **Must:** Every team member has good knowledge about TDD
- **Must:** Every team member has some knowledge about Exploratory Testing
- **Should:** following Agile practices should be understood *(bolded are primary XP practices)*
  - User Stories & Scenarios
  - Refactoring
  - Whole Team
  - Informative workspace
  - Ten minute bould
  - Continuous Integration
  - Incremental design

**Organisational check list**

- **Must:** Already in Scrum
- **Must:** Nominated Scrum Master
- **Must:** Retrospectives held in each iteration
- **Should:** QA team is an integral part of the development team,
  - **Must:** Plan of transferring QT team to be an integral part of the development team exists.
- **Should:** 'Requirements' are defined as User Stories
- **Should:** Project Management supports agile manifesto:
  - *Individuals and interactions* over processes and tools
  - *Working software* over comprehensive documentation
  - *Customer collaboration* over contract negotiation
  - *Responding to change* over following a plan

**Development environment check list**

- **Must:** Automated Build environment
- **Should:** continuous integration environment
- **Must:** xUnit test framework in use in TDD
- **Must:** Automated Acceptance Test framework, e.g. Selenium / Watir / FIT
- **Must:** Defect Tracking System to track down defects
Extra Reference Materials


Cohn, Mike, User Stories Applied For Agile Software Development, Addison Wesley, 2003


Mike Cohn Jan, 2010... Succeeding with Agile software development using scrum

Mike Cohn 2009: Succeeding with Agile: Software Development Using Scrum. Published: Oct 26, 2009 by Addison-Wesley Professional. Part of the Addison-Wesley Signature Series (Cohn) series.

Mike Cohn 2006: Agile Estimating and Planning by Mike Cohn Published: Prentice Hall 14 Mar 2006


Rachel Davies: Agile Coaching Published: 5 Jun 2009 Pragmatic Bookshelf
Robert K. Wysocki 2009: Effective Project Management: Traditional, Agile, Extreme
Publisher: John Wiley & Sons 2009 -04-27

Roman Pichler 2010: Agile Product Management with Scrum: Creating Products that Customers Love. Published: Mar 22, 2010 by Addison-Wesley Professional. Part of the Addison-Wesley Signature Series (Cohn) series.

Articles:


Andrew Lance
http://www.stickyminds.com/sitewide.asp?ObjectId=6057&Function=edetail&ObjectType=ART


Appendix 2


**Internet Pages**

Nokia Corporation

http://www.nokia.com
http://academic.udayton.edu/bradhume/phone/corp/corporations/htm/nokia.htm
www.executivebrief.com/agile/how-to-scrum/


Nokia. Series 60 Platform Product Overview.
Appendix 2

6 (6)


http://en.wikipedia.org/wiki/Test_automation
Categories: Software test automation GUI automation

TestQuest. TestQuest Interface for Symbian OS. www-document.

Symbianone. TestQuest Pro Testing automation.


http://www.theautomatedtester.co.uk/blog/2009.htm

http://www.sdmagazine.com/breakrm/features/s992f2.shtml (06.04.1999)