POLITECNICO DI TORINO

Department of Control and Computer Engineering (DAUIN)

Master Thesis

Quality Assurance of Software Products

Supervisor:

Prof. G. Claudio Demartini

Author:

Ebrahim Kargar Nasrabadi

December, 2017

Abstract

The last twenty years has seen major changes in the methodologies, tools and technologies we use to build software, but most significantly, the way we build software has fundamentally evolved in order to improve the quality, accuracy, and speed of delivering software. By all these regular technological disruptions, growing fast has become essential to survival for software companies.

Quality Assurance, methodologies and techniques have been accordingly evolved in software industry. Software quality is one critical component of the criteria used to measure success of a software development project.

The goal of this master thesis is to describe software quality assurance principals. This work mainly focused on critical quality metrics of software products and some of the best practices on software development lifecycle to help organization and teams launching successful software product.

Acknowledgements

First of all, I would like to thanks Prof.Demartini, undoubtedly without his patience and advice would have been difficult to make this work, second of all I would like to give a big thanks to my company supervisor Ing.Antonio Gaspari who trusted me to get enter in a new area of my profession. Finally, I must express my very profound gratitude to my parents for supporting my university studies.

Contents

List of Figures

Chapter 1: Introduction

The last twenty years has seen major changes in the methodologies, tools and technologies we use to build software, but most significantly, the way we build software has fundamentally evolved in order to improve the quality, accuracy, and speed of delivering software projects. By all these regular technological disruptions, growing fast has become essential for the survival of software companies.

Emergence of new paradigms in software developments not only effected methodologies and the process of developing software but also tools and technologies has been subject to massive changes in past few decades.

While some of these new paradigms are concerned with new methodologies and techniques on developing software the others are emerging tools, platforms and framework in software industry.

In first part of this work, we are examining some of these emerging paradigms including: agile software development methodologies, pair programming, code refactoring & code review , continuous integration and continuous delivery (CI/CD), Microservices, software version control systems, etc. In following we are focusing on software quality assurance principles, techniques and tools to demonstrate how quality assurance is related to iterative software development process.

Although a good quality process most often leads to a better quality, it does not mean that one can forget the quality of the software product itself. Instead of the quality processes or the quality of the processes, the goal of this work is to give you a better understanding on the critical metrics need to be consider in order to assure the quality of a software product itself.

As software becomes larger and more sophisticated, the challenge is to develop more complicated software products within the constraints of time and resources without the sacrifice of quality.

1.1 New paradigms in Software development process

The globalization of companies operations and competition between software vendors demand improving quality of delivered software and decreasing the overall cost. The same in fact introduce a lot of problem into software development process as produce distributed organization breaking the co-location rule of modern software development methodologies. [1]

Unlike most manufactured goods, software undergoes continual redesign and upgrading in practice because the system component adapts the general-purpose computer to its varied and often-changing, special-purpose applications. As needs change, the software programs that were designed to meet them must change. A large body of technology has developed over the past 50 years to make software more reliable and hence trustworthy. [2]

Since 2004 a virtual revolution in software development technology has taken place. Every aspect of software development has been affected from the tools, the languages to methodologies and principles. The way we compose functionality from components and services to build applications and the way we deliver the applications has all changed.

Many aspect of the development and delivery process has undergone massive automation or improvements that have all made the process of developing software more continuous.

More productive languages with many built in features, libraries of open source components to leverage, fast iteration and testing helped by high degrees of automation and the use of APIs as a way to leverage other functionality has made the front end process of software development much more productive.

The deployment process has typically been ½ of the entire process of software development. With new tools that integrate the cycle between development, test and production it is possible to reduce the time to deployment to less than a day and in many cases minutes. The combination of these improvements has literally improved productivity and the ability to generate functionality by at least a factor of 10 and sometimes 100 to 1 from older technologies. [3]

In following sections we are going through some of these major changes in software lifecycle managements and the effect of them on the delivery of any software product.

1.1.1 Agile Software Development Methodologies

The field of software development has been never shy of introducing new methodologies. Indeed, in the last 25 years, a large number of different approaches to software development have been introduced, of which only few have survived to be used today.

While no agreement on what the concept of "agile" actually refers to exists, it has generated a lot of interest among practitioners and lately also in the academia. [4]

The "Agile Movement" in software industry saw the light of day with the Agile Software Development Manifesto published by a group of software practitioners and consultants in 2001. The focal values honored by the Agilists are presented in the following subjects:

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

The central values that agile methodologies adhered to software development lifecycle are:

First, the agile movement emphasizes the relationship and communality of software developers and the human role reflected in the contracts, as opposed to institutionalized processes and development tools. In the existing agile practices, this manifests itself in close team relationships, close working environment arrangements, and other procedures boosting team work spirit.

Second, the vital objective of the software team is to continuously turn out tested working software. New releases are produced at frequent intervals, in some approaches even hourly or daily, but more usually bi-monthly or monthly. The developers are urged to keep the code simple, straightforward, and technically as advanced as possible, thus lessening the documentation burden to an appropriate level.

Third, the relationship and cooperation between the developers and the clients is given the preference over strict contracts, although the importance of well drafted contracts does grow at the same pace as the size of the software project. The negotiation process itself should be seen as a means of achieving and maintaining a viable relationship. From a business point of view, agile development is focused on delivering business value immediately as the project starts, thus reducing the risks of non-fulfillment regarding the contract.

Fourth, the development group, comprising both software developers and customer's representatives, should be well-informed, competent and authorized to consider possible adjustment needs emerging during the development process life-cycle. This means that the participants are prepared to make changes and that also the existing contracts are formed with tools that support and allow these enhancements to be made.

Hence agile software development processes has following characteristics:

1. Modularity on development process level
2. Iterative with short cycles enabling fast verifications and corrections
3. Time-bound with iteration cycles from one to six weeks
4. Parsimony in development process removes all unnecessary activities
5. Adaptive with possible emergent new risks
6. Incremental process approach that allows functioning application building in small steps
7. Convergent (and incremental) approach minimizes the risks
8. People-oriented, i.e. agile processes favor people over processes and technology
9. Collaborative and communicative working style [4]

Here we are addressing some of the most applicable agile software development techniques that have gained considerable attention in the software industry, in addition we will observe how each of these techniques impacted quality of software products.

1.1.1.1 Extreme Programming

Extreme Programming (XP) has evolved from the problems caused by the long development cycles of traditional development models. It first started as 'simply an opportunity to get the job done with practices that had been found effective in software development processes during the preceding decades.

Figure 1 depicts a high-level view of the XP project lifecycle, although the term phase may bring connotations of waterfall development the fact is that phases can occur iteratively, something that is apparent in Figure 1 by the fact that it is possible to move back and forth between the Planning, Iterate to Release, and Productionizing phases. Phases aren't necessarily long - the Planning phase may only take several hours for example. Furthermore, XP teams typically don't think of themselves as working in phases, they just think of themselves as working.

Having said this it makes it easier to think about the development effort one phase at a time so let's examine each one in turn: Exploration Phase, Planning Phase , Iterations to Release Phase, Productionizing Phase , Maintenance Phase .[5]

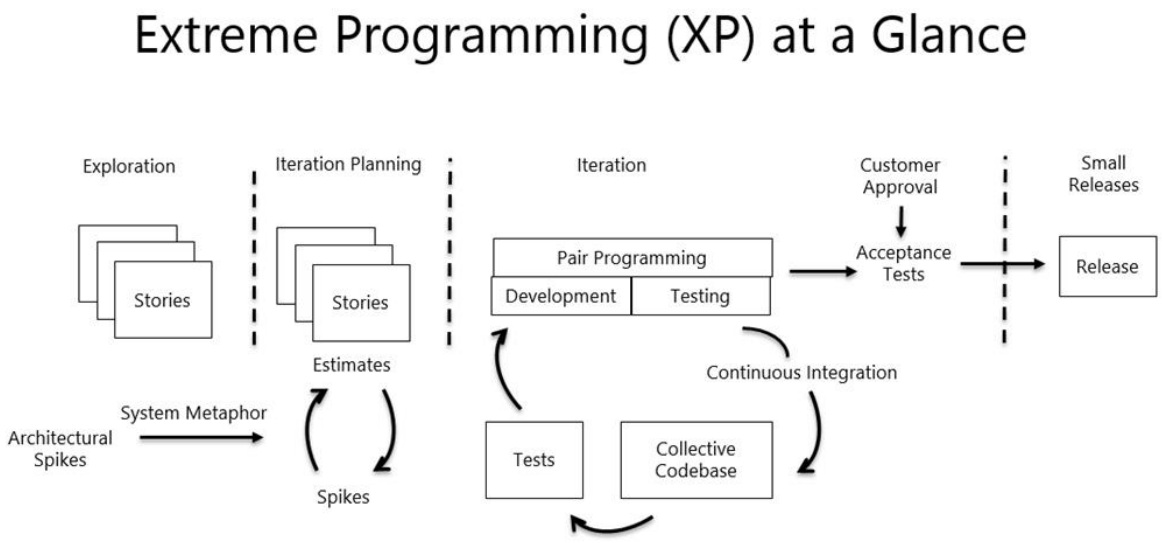


Figure 1.1 Extreme Programming lifecycle

1.1.1.2 Scrum

Scrum is a process framework that has been used to manage complex product development since the early 1990s. Scrum is not a process or a technique for building products; rather, it is a framework within which you can employ various processes and techniques. Scrum makes clear the relative efficacy of your product management and development practices so that you can improve.

The Scrum framework consists of Scrum Teams and their associated roles, events, artifacts, and rules. Each component within the framework serves a specific purpose and is essential to Scrum's success and usage. The rules of Scrum bind together the events, roles, and artifacts, governing the relationships and interaction between them.

The main idea of Scrum is that systems development involves several environmental and technical variables (e.g. requirements, time frame, resources, and technology) that are likely to change during the process. This makes the development process unpredictable and complex, requiring flexibility of the systems development process for it to be able to respond to the changes. [6]

Scrum is founded on empirical process control theory, or empiricism. Empiricism asserts that knowledge comes from experience and making decisions based on what is known. Scrum employs an iterative, incremental approach to optimize predictability and control risk. Three pillars uphold every implementation of empirical process control: transparency, inspection, and adaptation. [6] [7]

Prescribed events are used in Scrum to create regularity and to minimize the need for meetings not defined in Scrum. All events are time-boxed. Once a Sprint begins, its duration is fixed and cannot be shortened or lengthened. The remaining events may end whenever the purpose of the event is achieved; ensuring an appropriate amount of time is spent without allowing waste in the

process. The Scrum Events are: Sprint, Sprint Planning, Daily Scrum, Sprint Review, and Sprint Retrospective.

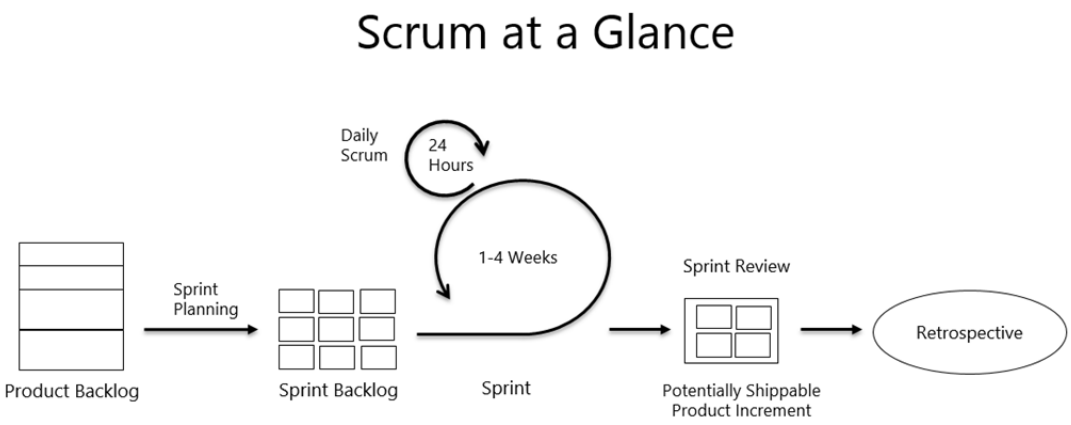


Figure 1.2 Scrum lifecycle

The Sprint

The heart of Scrum is a Sprint, a time-box of one month or less during which a "Done", useable, and potentially releasable product Increment is created. Sprints best have consistent durations throughout a development effort. A new Sprint starts immediately after the conclusion of the previous Sprint.

Sprints contain and consist of the Sprint Planning, Daily Scrums, the development work, the Sprint Review, and the Sprint Retrospective.

During the Sprint:

- No changes are made that would endanger the Sprint Goal.
- Quality goals do not decrease.
- Scope may be clarified and re-negotiated between the Product Owner and Development Team as more is learned.

Each Sprint may be considered a project with no more than a one-month horizon. Like projects, Sprints are used to accomplish something. Each Sprint has a definition of what is to be built, a design and flexible plan that will guide building it, the work, and the resultant product.

Sprints are limited to one calendar month. When a Sprint's horizon is too long the definition of what is being built may change, complexity may rise, and risk may increase. Sprints enable predictability by ensuring inspection and adaptation of progress toward a Sprint Goal at least every calendar month. Sprints also limit risk to one calendar month of cost.

Cancelling a Sprint

A Sprint can be cancelled before the Sprint time-box is over. Only the Product Owner has the authority to cancel the Sprint, although he or she may do so under influence from the stakeholders, the Development Team, or the Scrum Master.

A Sprint would be cancelled if the Sprint Goal becomes obsolete. This might occur if the company changes direction or if market or technology conditions change. In general, a Sprint should be cancelled if it no longer makes sense given the circumstances. But, due to the short duration of Sprints, cancellation rarely makes sense.

When a Sprint is cancelled, any completed and "Done" Product Backlog items are reviewed. If part of the work is potentially releasable, the Product Owner typically accepts it. All incomplete Product Backlog Items are re-estimated and put back on the Product Backlog. The work done on them depreciates quickly and must be frequently re-estimated.

Sprint cancellations consume resources, since everyone has to regroup in another Sprint Planning to start another Sprint. Sprint cancellations are often traumatic to the Scrum Team, and are very uncommon.

Sprint Planning

The work to be performed in the Sprint is planned at the Sprint Planning. This plan is created by the collaborative work of the entire Scrum Team.

Sprint Planning is time-boxed to a maximum of eight hours for a one-month Sprint. For shorter Sprints, the event is usually shorter. The Scrum Master ensures that the event takes place and that attendants understand its purpose. The Scrum Master teaches the Scrum Team to keep it within the time-box.

Sprint Planning answers the following:

- What can be delivered in the Increment resulting from the upcoming Sprint?
- How will the work needed to deliver the Increment be achieved?

Sprint Goal

The Sprint Goal is an objective set for the Sprint that can be met through the implementation of Product Backlog. It provides guidance to the Development Team on why it is building the Increment. It is created during the Sprint Planning meeting. The Sprint Goal gives the Development Team some flexibility regarding the functionality implemented within the Sprint. The selected Product Backlog items deliver one coherent function, which can be the Sprint Goal. The Sprint Goal can be any other coherence that causes the Development Team to work together rather than on separate initiatives.

As the Development Team works, it keeps the Sprint Goal in mind. In order to satisfy the Sprint Goal, it implements the functionality and technology. If the work turns out to be different than

the Development Team expected, they collaborate with the Product Owner to negotiate the scope of Sprint Backlog within the Sprint.

Daily Scrum

The Daily Scrum is a 15-minute time-boxed event for the Development Team to synchronize activities and create a plan for the next 24 hours. This is done by inspecting the work since the last Daily Scrum and forecasting the work that could be done before the next one. The Daily Scrum is held at the same time and places each day to reduce complexity. During the meeting, the Development Team members explain:

- What did I do yesterday that helped the Development Team meet the Sprint Goal?
- What will I do today to help the Development Team meet the Sprint Goal?
- Do I see any impediment that prevents me or the Development Team from meeting the Sprint Goal?

The Development Team uses the Daily Scrum to inspect progress toward the Sprint Goal and to inspect how progress is trending toward completing the work in the Sprint Backlog. The Daily Scrum optimizes the probability that the Development Team will meet the Sprint Goal. Every day, the Development Team should understand how it intends to work together as a self-organizing team to accomplish the Sprint Goal and create the anticipated Increment by the end of the Sprint. The Development Team or team members often meet immediately after the Daily Scrum for detailed discussions, or to adapt, or replan, the rest of the Sprint's work.

The Scrum Master ensures that the Development Team has the meeting, but the Development Team is responsible for conducting the Daily Scrum. The Scrum Master teaches the Development Team to keep the Daily Scrum within the 15-minute time-box.

The Scrum Master enforces the rule that only Development Team members participate in the Daily Scrum.

Daily Scrums improve communications, eliminate other meetings, identify impediments to development for removal, highlight and promote quick decision-making, and improve the Development Team's level of knowledge. This is a key inspect and adapt meeting.

Sprint Review

A Sprint Review is held at the end of the Sprint to inspect the Increment and adapt the Product Backlog if needed. During the Sprint Review, the Scrum Team and stakeholders collaborate about what was done in the Sprint. Based on that and any changes to the Product Backlog during the Sprint, attendees collaborate on the next things that could be done to optimize value. This is an informal meeting, not a status meeting, and the presentation of the Increment is intended to elicit feedback and foster collaboration.

This is a four-hour time-boxed meeting for one-month Sprints. For shorter Sprints, the event is usually shorter. The Scrum Master ensures that the event takes place and that attendants understand its purpose. The Scrum Master teaches all to keep it within the time-box.

The Sprint Review includes the following elements:

- Attendees include the Scrum Team and key stakeholders invited by the Product Owner;
- The Product Owner explains what Product Backlog items have been “Done” and what has not been “Done”;
- The Development Team discusses what went well during the Sprint, what problems it ran into, and how those problems were solved;
- The Development Team demonstrates the work that it has “Done” and answers questions about the Increment;
- The Product Owner discusses the Product Backlog as it stands. He or she projects likely completion dates based on progress to date (if needed);
- The entire group collaborates on what to do next, so that the Sprint Review provides valuable input to subsequent Sprint Planning;
- Review of how the marketplace or potential use of the product might have changed what is the most valuable thing to do next; and,
- Review of the timeline, budget, potential capabilities, and marketplace for the next anticipated release of the product.

The result of the Sprint Review is a revised Product Backlog that defines the probable Product Backlog items for the next Sprint. The Product Backlog may also be adjusted overall to meet new opportunities.

Sprint Retrospective

The Sprint Retrospective is an opportunity for the Scrum Team to inspect itself and create a plan for improvements to be enacted during the next Sprint.

The Sprint Retrospective occurs after the Sprint Review and prior to the next Sprint Planning. This is a three-hour time-boxed meeting for one-month Sprints. For shorter Sprints, the event is usually shorter. The Scrum Master ensures that the event takes place and that attendants understand its purpose. The Scrum Master teaches all to keep it within the time-box. The Scrum Master participates as a peer team member in the meeting from the accountability over the Scrum process.

The purpose of the Sprint Retrospective is to:

- Inspect how the last Sprint went with regards to people, relationships, process, and tools;
- Identify and order the major items that went well and potential improvements; and,
- Create a plan for implementing improvements to the way the Scrum Team does its work.

The Scrum Master encourages the Scrum Team to improve, within the Scrum process framework, its development process and practices to make it more effective and enjoyable for the next Sprint. During each Sprint Retrospective, the Scrum Team plans ways to increase product quality by adapting the definition of “Done” as appropriate.

By the end of the Sprint Retrospective, the Scrum Team should have identified improvements that it will implement in the next Sprint. Implementing these improvements in the next Sprint is the adaptation to the inspection of the Scrum Team itself. Although improvements may be implemented at any time, the Sprint Retrospective provides a formal opportunity to focus on inspection and adaptation. [6]

1.1.1.3 Kanban

Kanban is a method for managing knowledge work with an emphasis on just in time delivery while not overloading the team members. Development process is transparent in this methodology from definition of a task to its delivery to the customer, is displayed for participants to see. The Kanban Method is as an approach to incremental, evolutionary process and systems change for organizations. It uses a work-in-progress limited pull system as the core mechanism to expose system operation (or process) problems and stimulate collaboration to continuously improve the system. Visualization is an important aspect of Kanban as it allows understanding the work and the workflow. Kanban method consists of four basic principles:

0-Start with existing process: the Kanban method does not prescribe a specific set of roles or process steps. The Kanban method starts with existing roles and processes and stimulates continuous, incremental and evolutionary changes to the system. The Kanban method is a change management method.

1. Agree to pursue incremental, evolutionary change: the organization (or team) must agree that continuous, incremental and evolutionary change is the way to make system improvements and make them stick.

2. Respect the current process, roles, responsibilities and titles: it is likely that the organization currently has some elements that work acceptably and are worth preserving. The Kanban method seeks to drive out fear in order to facilitate future change. It attempts to eliminate initial fears by agreeing to respect current roles, responsibilities and job titles with the goal of gaining broader support.

3. Leadership at all levels: acts of leadership at all levels in the organization, from individual contributors to senior management, are encouraged. [7]

Traditionally, Kanban has been a physical board, with magnets, plastic chips, or sticky notes on a whiteboard to represent work items. However, in recent years, more and more project management software tools have created online Kanban boards.

A Kanban board, whether it is physical or online, is made up of different swim lanes or columns. The simplest boards have three columns: to do, in progress, and done. The columns for a software development project may consist of analysis, development, testing, approval, and deployment columns.

Kanban cards (like sticky notes) represent the work and each card is placed on the board in the lane that represents the status of that work. These cards communicate status at a glance. You could also use different color cards to represent different details.

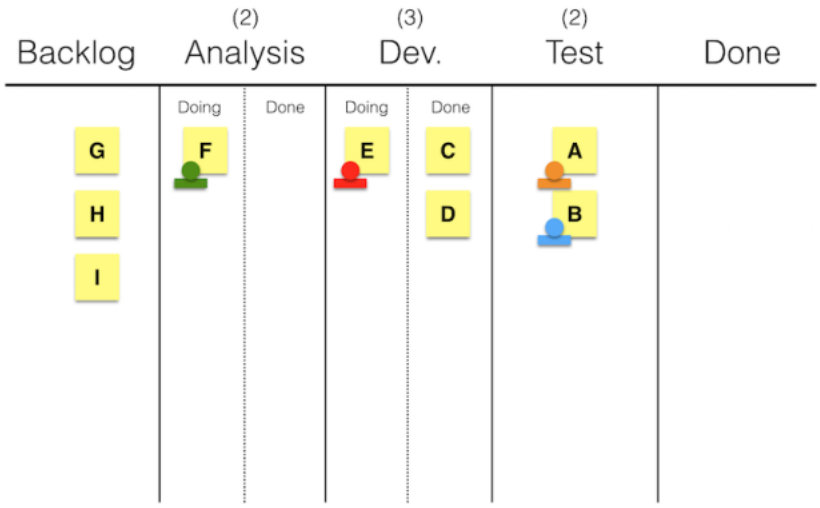


Figure 1.3 Kanban Board Example

Kanban Method enables incremental process improvement through repeated discovery of issues affecting process performance. It supports gradual, continual improvement toward higher performance and greater quality. It drives process evolution, rather than a revolution. Here is how Kanban principles help software development team to deliver a product with a better quality:

I. Optimize Existing Processes – Introduction of visualization and the limiting of work-in-progress (WIP) will catalyze change with minimal disruption.
II. Deliver with Higher Quality – Limiting work-in-progress and defining policies for work prioritization will bring greater focus on quality. Policies can also address quality criteria directly.
III. Improve Lead Time Predictability – There is a correlation between the amount of work-in-progress, lead time and defect rates. Limiting WIP makes lead times dependable and keeps defect rates low.
IV. Improve Employee Satisfaction – Kanban reduces context switching and pulls work at the rate the team can complete it. Working at a more even, predictable pace, means employees are never overloaded.
V. Provide Slack to Enable Improvement – Creating slack in the value chain improves responsiveness to urgent requests and bandwidth to enable process improvement and quality improvement.
VI. Simplify Prioritization – Kanban enables fast reprioritization to accommodate changes in the market.
VII. Provide a Transparency on the System Design and Operation – Improved visibility builds trust with customers and managers. It also shows the effects of actions or inactions. As a result, collaboration improves.

VIII. Enables Emergence of a "High-Maturity" Organization – As improvements are implemented, organizational maturity improves leading to better decision making and improved risk management. Risk, managed appropriately, brings predictable results. [8]

1.1.1.4 Other Agile Methods

In the previous subsections, several methods for producing software in an agile manner and their potential impact on the quality of software have been introduced. The focus has been on methods that provide comprehensive frameworks with more or less tangible processes and practices that cover as much of the software development process as possible.

The agile approach has, however, recently brought a great deal of research and lots of interesting new ideas, which either have not been documented thoroughly (e.g., Lean Software Development ,Feature-Driven development, Open Source Software development, Adaptive Software development) or have been published just very recently. [7]

1.1.2 Code Refactoring & Code Review

Code Refactoring is a maintenance task in which source code is restructured to enhance its quality while the external behavior of the system is preserved. The term "code review" can refer to a range of activities, from simply reading some code over your teammate's shoulder to a 20-person meeting where you dissect code line by line.

Refactoring process is often referred as a reconstruction process of the source code to minimize fault or defect, either as an act of correction or prevention. For example, code clones and other redundancy in a code are considered as threat for future development of the software. These faults in source code are known as "bad smells" in code. Other benefit of refactoring apart from improving the code quality is to improve the structure of the software so that it becomes easier to understand. [9]

There are few benefits from software refactoring. The benefits are as follows.

a. To remove duplicated code and other bad smells.
b. To improve software design quality.
c. To increase understandability of the code.
d. To reduce project evolution time, especially in source code management activities.

The participants in a code review are the author, who writes the code and sends it for review, and the reviewer, who reads the code and decides when it's ready to be merged in to the team's codebase. A review can have multiple reviewers.

Before the code review begins, the author must create a changelist. This is a set of changes to source code that the author wants to merge in to the team's codebase.

A review begins when the author sends their changelist to the reviewer. Code reviews happen in rounds. Each round is one complete round-trip between the author and reviewer: the author sends changes, and the reviewer responds with written feedback on those changes. Every code review has one or more rounds.The review ends when the reviewer approves the changes.

It's easy for an author to interpret criticism of their code as an implication that they are an incompetent programmer. Code reviews are an opportunity to share knowledge and make informed engineering decisions. But that can't happen if the author perceives the discussion as a personal attack. [10]

Some refactoring scenarios are found to improve some quality aspects of software and weaken others. These findings lead to a conclusion that refactoring does not always improve all software quality aspects.

Developers stated that refactoring is "code transformation that improves some aspects of program behavior such as readability, maintainability, or performance". [11]

Refactoring requires multi-dimensional assessment. Therefore, we believe that it is incorrect to limit the study of the impact of a certain refactoring scenario on quality to a certain quality attribute, obtain some negative results, and state a general conclusion that the considered refactoring scenario causes the software quality to weaken. [12]

1.1.3 Pair Programming

Pair-programming is a technique in Agile Software Development where two engineers share a single workstation. In this technique, one engineer is the driver, whom has control of the mouse and keyboard to write the code, while the other serves as the navigator, reviewing the code that the other is writing while providing tactical and analytical feedback. This pair will trade-off on these roles at regular intervals, giving each other equal chance to both execute on the work or direct it.

The ultimate goal of pair-programming is to provide a means to achieve better quality in software, while providing many secondary benefits that improve the ability of a team to continue delivering useful features to customers.

Although Code reviews often find typos and simple mistakes, but they do not provide the same level of insight into questions about the quality of software architecture and design. [13]

These are significant benefits of pair programming:

- Many mistakes get caught as they are being typed in rather than in QA test or in the field (Continuous code reviews)
- The end defect content is statistically lower (continuous code reviews)
- The designs are better and code length shorter (ongoing brainstorming and pair relaying)
- The team solves problems faster (pair relaying)

- People learn significantly more, about the system and about software development (line-of-sight learning)
- The project ends up with multiple people understanding each piece of the system
- People learn to work together and talk more often together, giving better information flow and team dynamics
- People enjoy their work more.

The development cost for these benefits is not 100% that might be expected, but is approximately 15%. This is repaid in shorter and less expensive testing, quality assurance, and field support. [14]

1.1.4 Continuous Integration and Continuous Delivery (CI/CD)

Software integration is the practice of linking together subsystems or components of software to produce a single unified system. It is a part of any software development lifecycle as software is usually developed through different phases and by a team of engineers. Software integration is done in traditional software development lifecycle as an independent step in a later phase, after the implementation of software is completed.

It means basically that each developer integrates his/her work continuously (at least once a day). This practice assures that small parts are added immediately once they are implemented and are ready to be a part of the system, before they become complicated. Applying the practice of Continuous Integration requires that each time a new part is added to the system, automatic test cases are created and added to cover the whole system including the newly added parts. It also requires that the software gets tested and built automatically and the developer whose codes are added receives an immediate feedback about the newly integrated codes.

Any feedback at this stage will be considered by the developer as soon as possible. This helps in identifying bugs and issues while the developer's knowledge about his/her codes are still fresh. In order for developers to benefit from implementing the practice of CI, they should change their typical day-today software development habits. CI requires each individual to commit code frequently, no to commit broken code, fix broken builds immediately, and write automated tests, all written tests and inspections must pass, run private builds and avoid getting broken code.

The table below contains each one of the framework quality attributes and how it differs when the software is integrated traditionally and when integrated continuously.

criteria	**Before continuous integration**	**After continuous integration**
Time to develop	Developers work on the whole release requirements	Developers work on a single requirement
Introduced bugs	As testing is done only after finishing a large part of code, the number of bugs is greater than when testing is done after implementing only small parts of code	Every feature is tested and fixed after it is completed and it doesn't have to wait until the whole release is finished
Time to deliver	After finishing the release and testing it	After finishing a single requirement and testing it
Test quality	Testing is done to the releases as a whole Testing environment is not the same as the production environment	Testing a feature is done once it is integrated Small parts are tested individually and testing results are sent immediately back to the developers Testing environment is almost exactly the same as the production environment
Documentation	The release is documented properly The release specification is documented before starting the work And the development work is documented in detail after finishing	The requirements are documented But there is a minimal amount of documents by the developers The automatic tools generate statistics and data about the developed features based on the requirements, the testing results and the generated bugs etc.
Change management	Change is only accepted after a long process and approvals Changes are introduced through a whole new patch or release	Change is accepted at any time only by adding new requirements by the owner to the business analysis and software development teams
Cost model	The time and effort of manually doing the software build and integration	The cost of getting continuous integration server and tools

Table 1.1 Impact of CI/CD in Software Quality

Software producers are shifting their development practices towards continuous integration as it showed a significant improvement to the overall software quality. Many of the risks involved in the software integration process were mitigated as a result of integrating parts of the developed software continuously once it is ready. [15]

1.1.5 Microservices and Distributed Software Architecture pattern

In recent years, a shift of focus in developer communities and publications could be observed: from people and processes (e.g., agile practices such as user storytelling and test automation) to integration technology and application hosting (e.g., RESTful HTTP, cloud computing, DevOps).

Under the umbrella term Microservices, new interest in software architecture and design can be observed at present (similar in intensity to the early days of the patterns movement.

Project team members are no longer considered to be "architecture astronauts" when considering and arguing about Microservices architectures. Agile architecture represents a consensus position between process and structure. Successful Microservices architecture designs and Microservices deployments are made possible by modern software engineering paradigms and recent advances in Web application development – for instance:

a. Domain-driven design and test-driven development
b. IDEAL pipesand-filters chaining of fine-grained processing logic
c. Polyglot programming and persistence
d. Build and test process automation and continuous deployment, e.g., into lightweight containers and cloud computing environments and
e. Lean approaches to systems management closely intertwined with software construction.

Team organization is vital for success. Microservice architectures allow to assign the responsibility for all concerns of certain business capabilities – from requirements to operations and from individual to teams. [16]

In addition to that full automation of quality assurance and software deployment allows for early fault and error detection, thus reducing repair times both during development and during operations. Microservice architectures enable scalability, agility and reliability, although monitoring and fault tolerance are difficult for a distributed system. [17]

1.1.6 Other trends

Software development is going through more changes than ever. Among them some of the most impactful changes are Distributed version control systems (DVCS) and containers. Distributed version control systems (DVCS) allow team members to review each other's code, work from any location, and easily branch and merge their work. Further, with each developer working in a local repo, their changes—and, more importantly, their missteps—are isolated. Although a

number of these systems exist, Git is the most widely used and takes the pain out of collaborating on code.

Containers are among the newest trends in software development, allowing engineers to move programs more easily between different computing environments. Despite being relatively new, just over a third of companies responding said they were already using

Chapter 2 Agile Software Quality Assurance

Quality Assurance, methodologies and techniques have been accordingly evolved in software industry. Software quality is one critical component of the criteria used to measure success of a software development project.

IEEE Standard for Software Quality Assurance Processes defines Software Quality Assurance:

"A set of activities that define and assess the adequacy of software processes to provide evidence that establishes confidence that the software processes are appropriate for and produce software products of suitable quality for their intended purposes. A key attribute of SQA is the objectivity of the SQA function with respect to the project. The SQA function may also be organizationally independent of the project; that is, free from technical, managerial, and financial pressures from the project." [18]

The International Standards Organization ISO 9000 defines quality as the totality of characteristics of an entity that bear on its ability to satisfy stated or implied needs. Where 'stated needs' means those needs that are specified as requirements by the customer in a contract, and 'implied needs' are those needs that are identified and defined by the company providing the product.[19]

Agile software development methodologies have since their inception claimed to improve the quality of the software product. The agile practitioners have also claimed that use of the agile approach has greatly improved the quality of their products. However, software quality is a rather complex concept; in fact some have defined the entire discipline of software engineering as the production of quality software.

From an agile perspective quality has been defined by some practitioners as follows:

McBreen defines agile quality assurance as the development of software that can respond to change as the customer requires it to change. This implies that the frequent delivery of tested, working, and customer-approved software at the end of each iteration is an important aspect of agile quality assurance. [20]

Ambler considers agile quality to be a result of practices such as effective collaborative work, incremental development, and iterative development as implemented through techniques such as refactoring, test-driven development, modelling, and effective communication techniques. [21]

In this chapter, the agile methodology analyzed from the good enough quality viewpoints, accordingly we are examining some of these existed techniques in agile support achieving quality.

2.1 Software Quality Metrics

Before diving into the fact that how these agile techniques affect quality of software lets discuss what are the most important quality metrics in a software product. Table 2.1 is given a summary of the quality metrics that define agile quality.

Parameter	**Description**
Functionality	The ability of a system to perform according to defined specification.
Performance	Appropriate performance of a system under cases not covered by the specification. This is complementary to correctness.
Extendibility	A system that is easy to adapt to new specification.
Reusability	Software that is composed of elements that can be used to construct different applications.
Compatibility	Software that is composed of elements that can easily combine with other elements.
Efficiency	The ability of a system to place as few demands as possible to hardware resources, such as memory, bandwidth used in communication and processor time.
Portability	The ease of installing the software product on different hardware and software platforms.
Timeliness	Releasing the software before or exactly when it is needed by the users.
Integrity	How well the software protects its programs and data against unauthorized access.
Verifiability and Validation	How easy it is to test the system.
Usability	The ease with which people of various backgrounds can learn and use the software.
Maintainability	The ease of changing the software to correct defects or meet new requirements [20].
Cost-effectiveness	The ability of a system to be completed within a given budget.

Table 2.1 DESCRIPTION OF SOFTWARE QUALITY ASSURANCE Metrics

In order to give a better understanding of the information in Table2.1 the impact of agile methodologies on each of these factors will now be discussed. For each parameter we have listed some practices that we believe will improve on the way agile development implements software quality assurance.

Functionality

Functionality is a metric to verify that a software application performs and functions correctly according to design specifications. Functionality metrics check if the core application functions, text input, menu functions and installation and setup on localized machines, etc. working as expected.

Performance

Performance is a pervasive metric of quality of software systems; everything affects it, from the software itself to all underlying layers, such as operating system, middleware, hardware, communication networks, etc.

Extendibility

Extendibility of a system is a general feature of all developed software applications; however emphasis should be on technical excellence and good design. To improve on these techniques use of modeling techniques for software architecture should be in agile development.

Reusability

This quality factor is generally implemented through the concepts of Object-Oriented technology. More work on agility and software architecture, and patterns can improve the reusability of agile products.

Compatibility

Agile techniques that ensure correctness of a system include the following: A general feature of all Object-Oriented (OO) developed softwares. Possible improvement on the agile approach includes design and architectural considerations that aim for platform independence.

Efficiency

Agile techniques that ensure efficiency of a system include the following: application of good coding standards. To improve on the techniques designs based on the most efficient algorithms are encouraged.

Robustness

Agile techniques that ensure robustness of a system originally defined as a major part of Object-Oriented design which agile development follows.This is case dependent; however agile development ensures robustness in the general sense through the development standards that are inherent to particular development platform in use.

Portability

Originally defined as a major part of Object-Oriented design and now further enhanced by the concepts of distributed computing and web services, this quality factor is generally implemented through the concepts of Object-Oriented design.

Timeliness

Agile techniques that ensure timeliness of a system include the following: iterative development, quick delivery, and short cycles. This can be improved upon by reducing the time for the deployment process.

Integrity

Integrity of a system is ensured at operating system level and also at the development platform level. Improving the integrity of the techniques that define the product would improve system integrity.

Verifiability and Validation

Agile techniques that ensure verification and validation of a system include the following: test-driven-development, unit tests and frequent integration.

To improve on these techniques more tools could be developed to link the existing testing approaches the concepts of test-driven-development.

Usability

Agile techniques that ensure ease of use of a system include the following: since the customer is part of the team, and customers give feedback frequently, they will likely recommend a system that is easy to use. The frequent visual feedback that customers get during the delivery of iteration allows them to provide useful feedback to improve the usability of the system. These can be improved upon by designing for the least qualified user in the organization.

Maintainability

The application of Object-Oriented design principles leads to maintainable systems. Development technologies that improve the interfaces between different object modules can have a positive impact on maintainability.

Correctness

Agile techniques that ensure correctness of a system include the following as elicited from the generic principles that guide agile development: writing code from minimal requirements, specification, which is obtained by direct communication with the customer, allowing the customer to change requirements, user stories, and test-first development. Since all the development in agile processes is done iteratively these techniques ensure the correctness at iteration level before making the decision to continue or cancel the project.

These agile techniques can be improved by implementing the following: Consider the possibility of using formal specification in agile development (which some developers are already using), possible use of general scenarios to define requirements. [22] [23]

Cost-effectiveness

Agile techniques that ensure cost-effectiveness of a system include the following: controlling the scope creep, for example in the Scrum methodology the Sprint technique is used which prevents introduction of requirement changes until the end of the iteration (Sprint).

Possible improvements include avoiding scope creep without locking requirement changes. It is generally difficulty to convince a customer to sign a contract for a project whose cost is based on the cost of each iteration. The advantage of costing based on iterations however is that since iterations are short (one to four weeks) the customer gets frequent feedback on the project costs.

2.2 Agile Quality Assurance Techniques

In trying to have better understanding of start-of-art agile techniques defined for the quality of a software product, here we would introduce some these techniques and we will see how applying each of these methods in software development lifecycle could improve quality of software in its entire lifecycle.

2.2.1 Test-Driven Development techniques (TDD)

One of the primary techniques in agile quality assurance is Test-Driven Development. TDD also known as test-first programming or test-first development, is an evolutionary (iterative and incremental) approach to programming where agile software developers must first write a test that fails before they write new functional code.

The steps of TDD in detail, as shown in are:

i. Quickly add a test, basically just enough code so that the tests now fail.
ii. Run the tests, often the complete test suite, although for sake of speed they may run only a subset to ensure that the new test does in fact fail.
iii. Update the functional code so it passes the new test.
iv. Run the tests again.
v. If the tests fail return to step 3.
vi. Once the tests pass the next step is to start over (Agilists may also want to refactor any duplication out of their design as needed).

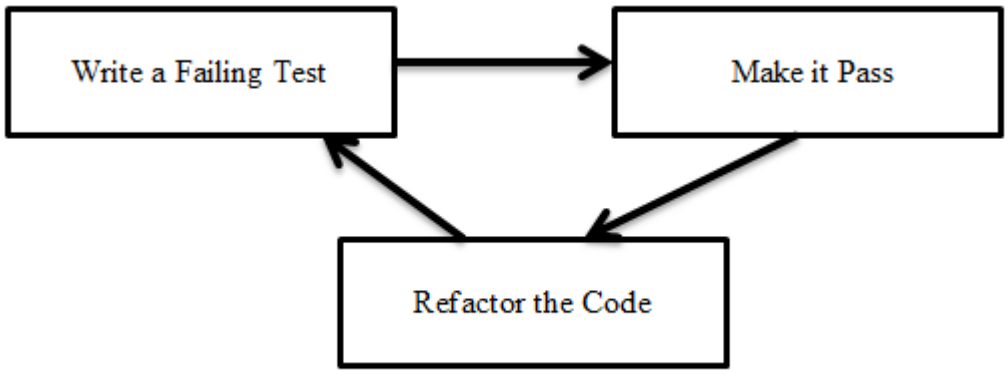


Figure 2.4 Test-Driven Development Lifecycle

There are several advantages of TDD for agile software development. First, TDD forces developers to think about what new functional code should do before they write it—in other words, to do detailed design just in time (JIT) before writing the code.

Second, it ensures that agile developers have testing code available to validate their work, ensuring that they test as often and early as possible. Third, it gives them the courage to refactor their code to keep it the highest quality possible, because they know there is a test suite in place that will detect if they have "broken" anything as the result of refactoring.

When agile developers become "test infected" they start to realize that tests are truly first-class artifacts that need to be developed and maintained throughout a project. They also realize that tests, when created properly, are more than just tests. For example, agilists consider acceptance tests to be first-class requirement artifacts—if an acceptance test defines criteria that the system must exhibit then clearly it is a requirement. Another common philosophy is that unit tests are detailed design artifacts. With a TDD based approach agilists write their unit test before writing their source code; in other words, they think through what the source code must do before they write it. The implication is that a unit test, when written before the domain code, effectively becomes a design specification for that portion of code. These philosophies can dramatically reduce the amount of work agilists need to do. [21]

2.2.1.1 Unit Testing

Unit tests are written before the source code and ran directly after the implementation is complete. Test-driven development forces the source code to be testable and guarantees that unit tests are written.
When user stories are combined with the technique of unit testing then each implementation of the user stories is tested as the system is developed ensuring correctness. This approach is followed for each software quality assurance parameter.
The information from the unit test results is used to determine whether the implemented code is good enough to be integrated which requires that the implemented code passes all unit tests.

Today, almost every programming language has its own unit testing framework (e.g., JUnit for Java, NUnit for C#), which enables the use of small, automatically executable unit tests. Unit testing has become an accepted practice, often even mandated by development processes (e.g., test-driven development). Nevertheless, software quality remains an issue. [24]
Furthermore Software engineering researchers therefore argue that there is a need to push automation in testing further — to even automatically generate unit tests.

2.2.1.2 Integration testing

Integration testing tests interfaces between components, interactions with different parts of a system, such as the operating system, file system, hardware, or interfaces between systems.
The greater the scope of integration, the more difficult it becomes for testers to isolate failures to a specific component or system, which may lead to increased risk. At each stage of integration, testers concentrate solely on the integration itself. For example, if they are integrating module A with module B they are interested in testing the communication between the modules, not the functionality of either module. Both functional and structural approaches may be used. [25]

2.2.2 Behavior-Driven Development techniques (BDD)

Behavior-Driven Development was originally invented by Dan North in the early to mid-2000s as an easier way to teach and practice Test-Driven Development. BDD adds a cycle around the TDD cycle, so that you start with a behavior and let that drive your tests, and then let the tests drive the development. Ideally, BDD is driven by some kind of acceptance test, but that's not 100% necessary. As long as you have the expected behavior defined, you're ok.

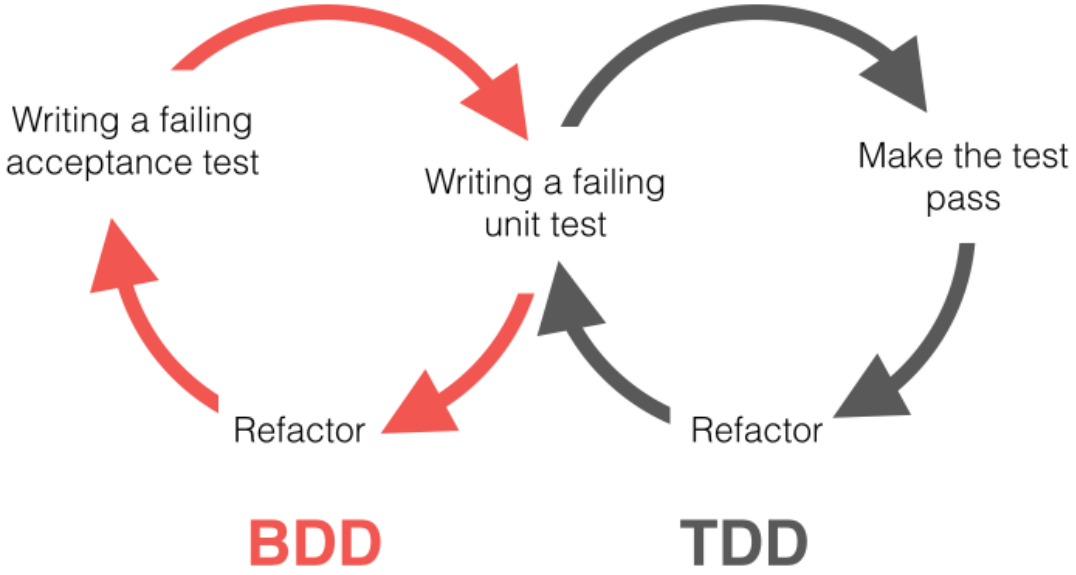


Figure 2.2: Behavior-Driven lifecycle

In following we are examining some of these behavior-driven testing techniques. [25]

2.2.2.1 Acceptance Testing

Acceptance testing is often the responsibility of customers or users of a system; other stakeholders may be involved as well. The goal in acceptance testing is to establish confidence in the system, parts of the system or specific non-functional characteristics of the system. Finding defects is not the main focus in acceptance testing.
Typical forms of acceptance testing include the following:

- User acceptance testing - Typically verifies the fitness for use of the system by business users.
- Operational (acceptance) testing- The acceptance of the system by the system administrators.
- Contract and regulation acceptance testing - Contract acceptance testing is performed against a contract's acceptance criteria for producing custom-developed software.
- Alpha and beta (or field) testing - Developers of market, or COTS, software often want to get feedback from potential or existing customers in their market before the software product is put up for sale commercially.

From descriptions of basic testing levels we can see that not only testers are involved in testing process but also developers, system administrators and last, but not least, users and customers.

It shows us the necessity of understanding what is the particular mission for each of involved stakeholders.
More information about software testing can be found in ISTQB's (International Software Testing Qualification Board) materials accessible from its Web pages. There can be found information about test levels, test types, test techniques and much more information about software testing. One particular type of testing is important for this thesis - Testing related to changes (confirmation testing (re-testing) and regression testing). [26]

2.2.2.2 Regression testing

Regression testing is the testing process of a previously tested program following modification to ensure that defects have not been introduced or uncovered in unchanged areas of the software, as a result of the changes made. It is performed when the software or its environment is changed.
Regression testing is the repeated testing of an already tested program, after modification, to discover any defects introduced or uncovered as a result of the changes. These defects may be either in the software being tested, or in another related software component.
It is performed when the software or its environment is changed. The extent of regression testing is based on the risk of not finding defects in software that was working previously. Regression testing should cover not only area in program that is directly related to the changes but should cover whole functionality of software.
Important thing that should be introduced is that regression testing should be performed at all test levels and should affect not only program code but also other components of software. [26]

2.2.2.3 System testing

System testing is concerned with the behavior of a whole system/product as defined by the scope of a development project or program.
In system testing, the test environment should correspond to the final target or production environment as much as possible in order to minimize the risk of environment-specific failures not being found in testing. System testing may include tests based on risks and/or on requirements specifications, business processes, use cases, or other high level descriptions of system behavior, interactions with the operating system, and system resources. An independent test team often carries out system testing.

2.2.2.4 Usability Testing

The Usability Professionals Association (UPA) definition focuses more on the product development process: "Usability is an approach to product development that incorporates direct user feedback throughout the development cycle in order to reduce costs and create products and tools that meet user needs." [27]

Usability testing refers to evaluating a product or service by testing it with representative users. Typically, during a test, participants will try to complete typical tasks while observers watch, listen and takes notes. The goal is to identify any usability problems, collect qualitative and quantitative data and determine the participant's satisfaction with the product.

Usability testing lets the design and development teams identify problems before they are coded. The earlier issues are identified and fixed, the less expensive the fixes will be in terms of both staff time and possible impact to the schedule.

During a usability test, you will:

i. Learn if participants are able to complete specified tasks successfully and identify how long it takes to complete specified tasks.
ii. Find out how satisfied participants are with your software or other product
iii. Identify changes required to improve user performance and satisfaction

In a usability test, there are two groups: users and observers. Ideally, the two groups do not know one another, so the observers can gather more objective data. If you are setting up a usability test, you construct a scenario for users to accomplish a set of tasks – ones that a new visitor to your website would need to accomplish – like signing up or inviting a friend or making a purchase. The users try to accomplish this set of tasks under controlled conditions.

While the users try to accomplish those tasks, the observers watch and/or measure their overall success in accomplishing those goals. They can either take notes while observing or record the session with audio or video for convenient recall. The observers take note of where the users succeed and where they have trouble, so they can revisit their designs at a later date and make improvements.
Some of the most applicable usability testing methods are:

Card Sorting:
Often used for testing a taxonomy or navigation structure, users organize sets of items into groups and give names or labels to them. This type of testing is informative for determining what to call various screens, pages or functions and how to group them.

In-person Testing:
This type of test is run by one or more observers in a fixed environment such as a conference room, either with small groups or individuals. Users are asked to accomplish a set of tasks and the observer can interact with them at any point to ask questions or to probe further.

Remote Testing:
In remote testing, users conduct a series of tasks in their own environment – and their attempts to accomplish tasks are often recorded via a browser webcam. This type of testing can be done either with a moderator (using webinar or conference call technology) or as a self-guided test.

A/B Testing:
A/B testing is a type of test that doesn't involve simulated experiences or observation; it puts two live variations of a website or app to the test and sends half of the traffic to one and half to the other, tallying the data for which variation had a higher conversion rate. [27]

Chapter 3 Performance Testing

Performance testing is a type of testing intended to determine the responsiveness, throughput, reliability, and/or scalability of a system under a given workload. Performance testing ensures software applications will perform well under their expected workload.

Performance testing is commonly conducted to accomplish the following goals:

- Assess production readiness
- Evaluate against performance criteria
- Compare performance characteristics of multiple systems or system configurations
- Find the source of performance problems
- Support system tuning
- Find throughput levels

Performance Testing is done to provide stakeholders with information about their application regarding speed, stability and scalability. More importantly, Performance Testing uncovers what needs to be improved before the product goes to market. Without Performance Testing, software is likely to suffer from issues such as:

- Running slow while several users use it simultaneously
- Inconsistencies across different operating systems and poor usability.
- User dissatisfaction with the performance characteristics of the system.
- Revenue losses or damaged brand credibility due to scalability or stability issues

Performance testing will determine whether or not their software meets speed, scalability and stability requirements under expected workloads. Applications sent to market with poor performance metrics due to non-existent or poor performance testing are likely to gain a bad reputation and fail to meet expected sales goals. [28]

This chapter provides a set of foundational building blocks on which to base your understanding of performance testing principles, ultimately leading to successful performance-testing projects.

3.1 Software Performance Metrics

Metric is a standard unit of measurement which calculates the results. Software Metrics are used for the evaluation of software product and its services.

Accordingly performance testing metrics are used to evaluate the application performance parameters and to identify which areas of the application are creating performance bottlenecks.

Quantitative results always provide the best outcome of an activity and metrics help in getting these quantitative results.

Following points also highlight the importance of performance testing metrics:

- Metrics are used to improve the product quality to achieve better customer satisfaction
- Metrics provide easy and clear output of the activity and identify areas which need more attention
- Metrics help in comparing the results of different test to find out the impact of any change in application
- Metrics are monitors which provide the exact cause of problem
- Metrics establish baseline for all tests
- Metrics track project progress

There are dozens of metrics generated in performance test but it's not possible to evaluate all of them. Here we are evaluating some of the specific performance metrics of a software product.

Total Page Views per Week

Page view is the request on server for that specific page with all its embedded objects. This metric provide the information about the weekly traffic on AUT. Choosing the page views metric over page hits or byte transferred is preferred because it's commonly used as an indicator for website traffic. Moreover, choosing the weekly limit also provides more realistic application traffic analysis over choosing the specific hours and day traffic.

Total Hits per Week

A hit is any resource (web page, image, files etc.) request received by the web server from the client. Several hits are made on server when client request for a web page. Web pages normally made of number of images and files and number of hits to web server for a specific web page will be equal to the number of resources it contains.

Total User Sessions per Week

A user session is a unique user visit on the website. This user uniqueness is maintained with many different approaches like with username and its password, browser cookies and the user machine IP address. Tracking user session is very handy in load testing because it's not only provide the information of number of users accessing the application but also provide the user

navigation trends on the application which is very important to simulate real user load during testing.

Average Hit Size

This is the average amount of data user received from the web server against a particular hit. Average hit size is measured in Kbytes.

Page Request Distribution

Page request distribution metric represent the user request distribution in percentage across all the website pages. This metric provides useful information on user trends and help out in deciding the user distribution on performance schedule.

User Abandonment

This metric provides the information on amount of time an average user waits for a page load before exiting from the application in dissatisfied manner. This value helps in deciding about the user acceptable response time limit.

Interaction Speed

Interaction speed represents the user interaction speed with the applications. This variable represents the how fast a user perform business actions on a web page and navigates between different web pages.

Latency Tolerance

This variable provide the information of how much a user waits for page response before taking next action which could be application abandonment, page reload etc. Today on average a user waits for 3 seconds for a webpage to load before taking any action.

Connection speed

It's important to know what percentage of users is using which internet connection. Software application response time will be much lower on fast internet connection as compared to lower connection. Moreover for realistic performance test, this internet connection bandwidth should be considered for virtual users as well.

User Geographical Location

User geographical location also greatly affects the user experience on the application. Application response time will be less for lesser number of hops between the client and the server.

Average Response Time

There is no doubt that response time is the most user concerned performance parameter. A software application or transaction with slow response time will never be accepted to users. So each executed transaction response time is monitored during every second of the scenario run to

evaluate the running users' impact on performance. These days average transaction response should not be more 3 seconds otherwise user abandonment started.

Throughput

Throughput graph provide the information on server response during specific time period of the performance test. This is also extremely important metric as it provides the information of web server response against the user requests. Throughput value is also directly proportional to user load unless web server is successfully able to handle that specific user load under those conditions.

Resource Usage

This metric provides information on machine resource usage including: CPU utilization, Memory Usage, Disk Space. [29]

3.2 Software Performance testing types

The following table declared the most common types of performance testing for Web applications.

Type	Purpose	Notes
Performance test	To determine or validate speed, scalability, and/or stability.	A performance test is a technical investigation done to determine or validate the responsiveness, speed, scalability, and/or stability characteristics of the product under test.
Load test	To verify application behavior under normal and peak load conditions.	Load testing is conducted to verify that your application can meet your desired performance objectives; these performance objectives are often specified in a service level agreement (SLA). A load test enables you to measure response times, throughput rates, and resource-utilization levels, and to identify your application's breaking point, assuming that the breaking point occurs below the peak load condition.
Stress test	To determine or validate an application's behavior when it is pushed beyond normal or peak load conditions.	The goal of stress testing is to reveal application bugs that surface only under high load conditions. These bugs can include such things as synchronization issues, race conditions, and memory leaks.

Table 3.1 Types of Performance Testing

Performance testing:

This type of testing determines or validates the speed, scalability, and/or stability characteristics of the system or application under test. Performance is concerned with achieving response times, throughput, and resource utilization levels that meet the performance objectives for the project or product. In this guide, performance testing represents the superset of all of the other subcategories of performance-related testing.

Load testing:

This subcategory of performance testing is focused on determining or validating performance characteristics of the system or application under test when subjected to workloads and load volumes anticipated during production operations.

Stress testing:

This subcategory of performance testing is focused on determining or validating performance characteristics of the system or application under test when subjected to conditions beyond those anticipated during production operations. Stress tests may also include tests focused on determining or validating performance characteristics of the system or application under test when subjected to other stressful conditions, such as limited memory, insufficient disk space, or server failure.

These tests are designed to determine under what conditions an application will fail, how it will fail, and what indicators can be monitored to warn of an impending failure. [30]

Any of these software performance test subsets has its impact on the quality of software products and accordingly generated value and emerging challenges on quality under each of performance tests is different from the others.

In the following table we described potential benefit of each of these performance testing types on the quality of software product.

Type	Benefits	Challenges
Performance test	-Determines the speed, scalability and stability characteristics of an application -Identifies mismatches between performance related expectations and reality. -Supports tuning, capacity planning, and optimization efforts.	-May not detect some functional defects that only appear under load. -If not carefully designed and validated, may only be indicative of performance characteristics in a very small number of production scenarios. -Unless tests are conducted on the production hardware, from the same machines the users will be using, there will always be a degree of uncertainty in the results.
Load test	-Determines the throughput required to support the anticipated peak production load. -Determines the adequacy of a hardware environment. -Detects concurrency issues. -Detects functionality errors under load. -Collects data for scalability and capacity-planning purposes. -Helps to determine how many users the application can handle before performance is compromised. -Helps to determine how much load the hardware can handle before resource utilization limits are exceeded.	-Is not designed to primarily focus on speed of response. -Results should only be used for comparison with other related load tests.

Stress test	-Determines if data can be corrupted by overstressing the system. -Provides an estimate of how far beyond the target load an application can go before causing failures and errors in addition to slowness. -Allows you to establish application-monitoring triggers to warn of impending failures. -Ensures that security vulnerabilities are not opened up by stressful conditions. -Determines the side effects of common hardware or supporting application failures. • Helps to determine what kinds of failures are most valuable to plan for.	-Because stress tests are unrealistic by design, some stakeholders may dismiss test results. -It is often difficult to know how much stress is worth applying. -It is possible to cause application and/or network failures that may result in significant disruption if not isolated to the test environment.

Table 3.2 Benefits of Performance Testing Types on the quality of software products

3.3 Software performance testing tools

There are a wide variety of performance testing tools available in market, they are either commercial or open-source.

Choosing the right performance testing tools depend on many factors including:

- Project requirements (pre-considerations)
- Availability of skilled resources
- Hardware requirements
- Protocol/technology/platform support
- Licensing model
- Scripting effort
- Solution for integration with test automation tool/existed platforms
- Desired Reporting & Monitoring features
- In-house versus Outsourced
- Budget $$$$$$ [30]

Here we overview some of most well-known performance testing tools.

IBM Rational Performance Tester:

IBM Rational Performance Tester helps you test earlier and more frequently as part of a DevOps approach. It validates the scalability of web and server applications, identifies the presence and cause of system performance bottlenecks and reduces load testing.

Key features:

Create test scripts without programming, saving time and reducing test complexity.

- ✓ Offers flexible modeling and emulation of diverse user populations while minimizing the memory and processor footprint.
- ✓ Supports load testing against a broad base of applications such as HTTP, SAP, Siebel, SIP, TCP Socket and Citrix.
- ✓ Use the cloud to enable large-scale and globally distributed performance testing.
- ✓ Generates performance and throughput reports in real time, offering immediate awareness of performance problems at any time during a test.

HP LoadRunner

This is an HP product which can be used as a performance testing tool. This can be bought as an HP product from its HP software division. Also, it is very much useful in understanding and determining the performance and outcome of the system when there is an actual load. One of the key attractive features of this testing tool is that it can create and handle thousands of users at the same time.

This tool enables you to gather all the required information with respect to the performance and also based on the infrastructure. The LoadRunner comprises of different tools; namely, Virtual User Generator, Controller, Load Generator and Analysis.

WebLOAD:

WebLOAD is a load testing tool from Radview is a performance testing software that used for performance and scalability but also for verifiability (validating the correctness of return results). Test scripts are written in Javascript (with COM/Java object integration), and the tool supports multiple protocols for testing all tiers of an app such as web (REST/HTTP with Ajax support), SOAP/XML, and other protocols callable from within scripts such as FTP, SMTP, etc.

LoadUI NG Pro:

LoadUI NG Pro makes it easy to model real-world loads on your API. LoadUI supports REST, SOAP, JMS, MQTT and many other API formats. LoadUI's easy-to-use graphical interface makes it simple for new users to setup load scenarios, and also provides advanced scripting features for those with more experience.

This tool allows you to spend more time on analyzing the results than on configuring and building tests by hand. You can learn even more about your application by making use of LoadUI's ability to report on server performance data. You can set up local or globally distributed load agents for your load testing scenarios

Apache JMeter

It is a Java platform application. It is mainly considered as a performance testing tool and it can also be integrated with the test plan. In addition to the load *test plan*, you can also create a functional test plan. This tool has the capacity to be loaded into a server or network so as to check on its performance and analyze its working under different conditions.

Initially, it was introduced for testing the web applications, but later its scope had widened. It is of a great use in testing the functional performance of the resources such as Servlets, Perl Scripts and JAVA objects.

BlazeMeter:

A performance engineering platform for DevOps, BlazeMeter enables you to run and analyze JMeter and other open-source load tests from anywhere. Check out the documentation here.Key Features:

- ✓ Quickly pinpoints defects using inbuilt error report function
- ✓ Detailed and interactive report timeline graph to make the report clearer
- ✓ Thorough testing using both, API backend and frontend
- ✓ Uses YAML and JSON syntax for developers using these languages [31]

Conclusion:

Quantitative results always provide a clear picture of the situation and keep track of changes. Performance testing metrics provides quantitative results which helps performance analysis of a software. Usage of these metrics always depends on the scope of the software product and its requirements. Having crystal clear requirements before starting the performance testing activity is a fundamental step. A test with vague requirements can never achieve its desired results. So in first phase of a software performance test, there is need to identify the set of metrics required for gathering complete list of performance requirements. Different stakeholders can be contacted to collect performance requirements. [29]

Chapter 4 Software Automation Testing

4.1 Why Automation?

Software Testing is the process of bringing the latent defects into the identifiable ones. This crucial phase of the software development lifecycle uncovers the potential defects in a software product. Regardless of time-consuming and resource-hungry nature of testing, we can never ignore it. Every newly developed or modified engineering product is required to pass rigorous tests so as to ensure the quality of the developed product.

Testing phase, being a major challenge in software development, it can be considered as a fair opportunity that can considerably help to improve and optimize software's cost, quality and time to market. This improvement is much desired in the present scenario when software industries are facing tough international competition and trying to shrivel their budgets and schedules.

We could classify the Software testing into two basic categories: a) Manual Testing and b) Automated Testing. Since long and now also, we are conducting manual testing of software products; in this type of testing a human tester executes the application and initiates various tests over it by interpreting and analyzing the behavior of the product on various input conditions. The human tester later prepares the reports and provides comments on the quality-state of the product by comparing the actual results against the expected results.

On the other hand an Automated Testing (AT) refers to the use of some standard software solutions to control the execution of test-cases on the Software Under Test (SUT). This process also involves setting up the preconditions, matching the actual results against the predicted ones and then documenting the observations according to some standard protocol.

Automated testing requires writing up some special computer programs to find bugs or defects in SUT. It is an excellent approach to replace the laborious and time consuming manual testing. Automated testing has various advantages and it is always suggested for the quality improvements of the application as it provides formal test coverage, avoid human errors and speed up the test execution process. Also, as it speeds up the execution process, it is most effective solution for meeting the strict deadlines.

The effect of test automation is measured along the software development lifecycle. It is a common observation in all the software projects that there is a positive cost and time impacts of test automation and quality is also improved in most of the cases as program is found incorrect fewer numbers of times with automated test cases than with manual testing. The availability increases in all the cases and relative time in testing is also fairly decreased because of test automation. [32]

4.2 Software Automation Testing Process and Planning

In any type of design in software test automation frameworks, in order to test a software application, the following steps are required:

1. **Studying and choosing the right framework/tool**

To be able to choose an appropriate framework/tool for automated testing, a pre-study must be conducted. This pre-study entailed reading about a selection of frameworks/tool existing at the time. To facilitate the decision, a number of requirements like cost of license, ability to develop different test cases, ease of implementation, etc. might be decided upon.

2. **Designing the test cases**

To design robust set of test cases, each of test cases should be as atomic as possible and mainly tested one element, the reason being that is that if one test case failed, no other test cases would be affected. The majority of the testing procedures consisted of verifying that a link was not broken or that a button worked as expected.

3. **Writing the test cases**

Since the testing script might be developed by programming of each test case from scratch, a substantial amount of code is going to be written. More importantly if the purpose of the writing test cases is for commercial software of a company, in order to continue using the test cases and develop them further, the code need to be clear and have a good structure.

4. **Execution and results evaluation**

After executing the test suites, assertions can be used to conform that the function worked as expected, by checking that software observed behavior in practice is as the expected logic.

5. **Possible improvements on Software**

Evaluating the results, you may find failure in some tests scenario lead to the possible changes or improvements into the structure of software, hence in order to fix potential bugs or improve some features you may continuously work to improve your product quality.

6. **Iterate on running the test cases**

To make sure that the test cases do not fail after any modifications without reason and that there were no bugs in the code, the tests need to be run repeatedly. During development the tests case runs continuously to make sure that any changes worked as expected. When the implementation is almost finished, you should execute a more extensive run of the tests to try and catch any problems that would only manifest sporadically. For this purpose the whole set of test on test suite could be run based on a regular job each day with the results of each run documented and evaluated.

4.3 Software automation Testing tools

We see a lot of improvements in software test automation in the past five years. As it happens in any sector of software industry, there were set a lot of trends. It usually starts with record and play approach and evolved to a modular approach and moving towards the data driven and keyword driven approach. Of course, these trends started a lot of debates on which design is better or more suitable for your team, your business and your needs.

Finding the right testing automation tool for a given context and to a given purpose is a difficult practical problem. There is a vast number of software testing and test automation tools available, both commercial and open source. The process of choosing the right tool requires, at least in theory, finding of a set of suitable candidate tools, comparison of those candidate tools and finally, selection of the most appropriate, efficient and effective one for the testing needs and tasks in the context in question.

In the following sections a few of testing automation tools that have been introduced.

Selenium

Selenium is an open source suite of testing tools which is one of the most established frameworks regarding test automation of web applications. Selenium has an IDE that can be used for test generation as well as the WebDriver that can be used for browser automation and programming test cases. The browser automation used by WebDriver works with most common browsers, such as Chrome, Firefox and Safari. The WebDriver framework supports test cases written in several different scripting languages, including Python, Java, JavaScript and C# .

Sahi

Sahi is a tool that's available in a free open source version as well as a Pro version at a monthly cost. The basis for test automation with Sahi is that test cases can be generated using Click & Record (C&R) and then the generated test cases can be built upon using the "Sahi Script" language (which is an extension of JavaScript). Regarding the web browser control there are Java and Ruby drivers available that works with most popular browsers. Additionally the Pro version includes built-in features for generating reports, storing reports in database, taking snapshots and more.

Since Sahi is based on generating scripts automatically using the C&R feature, theoretically the required programming skills are basic.

DalekJS

DalekJS is a UI testing tool that uses a browser automation technique, with which the Web-Driver JSON-Wire protocol is used to communicate with the browsers. Tests are written in JavaScript. DalekJS is still under development and is not recommended for production use by its creators.

Jasmine
Jasmine is a framework for testing JavaScript code. It is behavior-driven, not dependent on other JavaScript frameworks and does not rely on DOM or on browsers which makes it useful for different kinds of testing. Jasmine uses a syntax that is created with the purpose to be easily read and understood, so tests are written in such a way that they can be read as sentences. [33]

4.4 Automation Testing with Selenium

In the finishing part of this work we are discussing two example of test automation, first of all we overview Selenium as one of the widely used open source automation testing tools for regression and front-end testing in following we are discuss on SoapUI and its feature to implement a API automation test.

Selenium was created by Jason Huggins working in Thought Works in 2004. He was working on a web application that required regular testing. He realized that manual testing replication was becoming more and more inefficient; he created a JavaScript program that would automatically control the browser's action. He named this program JavaScript Test Runner. Afterward he completed this JavaScript Runner open source which was later re-named as Selenium Core. Selenium is a set of different software tools each with a different approach to supporting test automation. The entire suite of tools results in a rich set of testing functions specifically geared to the needs of testing of web applications of all types. [34]

Selenium operations are extremely flexible, it allows you to locate on every element of your page using different set of properties including element ID, CSS, XPath .Selenium is open source, with robust set of tools that supports rapid development of test automation specifically for web-based applications. Selenium provides a record tool for authoring tests even without learning too much about test scripting languages. Selenium is also a portable software testing framework for web applications. Some of its key features are:

- Compatible to runs in many browsers and operating systems
- Extendible by using many programming languages and testing frameworks.
- Providing a set of different tools each with a different approach for supporting test automation.

The entire suite of tools results in a rich set of testing functions specifically geared to the needs of testing of web applications of all types.

Basic Selenium Components
There are three versions of Selenium, which can be used in combination or isolation to create complete automation suite for the web applications. Each one has a specific role in aiding the development of web application test automation.

1) Selenium IDE
2) Selenium Core
3) Selenium RC

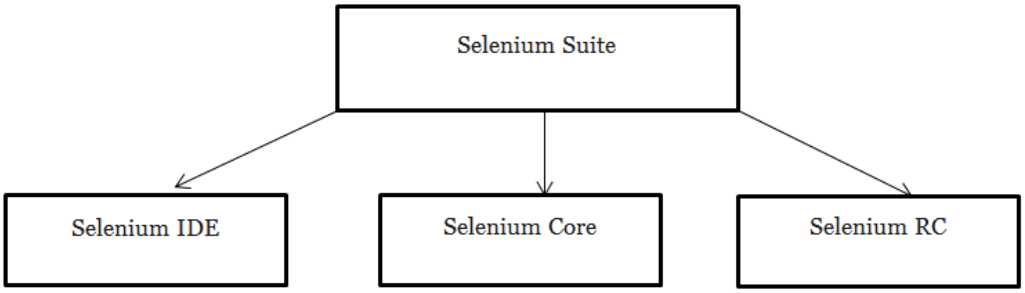


Figure 4.1 Selenium Suite Components

1. Selenium IDE

Selenium IDE is an integrated development environment for Selenium tests. Selenium IDE was originally created by Shinya Kasatani and donated to Selenium project in 2006. It is implemented as a Firefox extension, and has a recording feature.
Selenium-IDE also offers full editing of test cases for more precision and control.

2. Selenium Core

Selenium Core is a test tool for web applications. It is a simpler form of Selenium, and suitable for non-developers. Browser compatibility testing: To test the application if it works correctly on different browsers and operating systems. The same script can run on any Selenium platform. Selenium Core tests run directly in a browser, just as real users do. They run in Internet Explorer, Mozilla and Firefox on Windows, Linux and Macintosh.

Some disadvantages:

- It cannot handle file upload.
- It cannot switch between http and https protocols.
- It is a bit complicated for first time users to setup and start to use it.

3. Selenium Remote Control

Selenium Remote Control (RC) is the solution for tests that need more than simple browser actions and linear execution. We can use Selenium-RC whenever our test requires logic which is not supported by Selenium-IDE. Selenium-RC uses the full power of programming languages to create more complex tests like reading and writing files, querying a database, and emailing test results.

Features:

a) We can use Java syntax to write test script
b) Easy to conduct Data-Driven testing by reading test data from files
c) We can store test results into a file or generate graphical reports in HTML, PDF format. [35]

An example of automated UI test using Selenium

To give an insight in the possible applications of automated testing, in this part of the work we are providing an example of automated UI test using Selenium.
Every test case composed of a test scenario was made for declaring test purpose. A code example for one of the test scenario can be seen in figure 4.2.

```
Feature: Testing on Amazon - listamagazzino page

  Background:
    Given I have this users data for listamagazzino page:

      | USERNAME     | PASSWORD |
      | user_example | test!    |

    When I land on listamagazzino page and title is as expected


  @listamagazzino_page
    Scenario: I click on Aggiorna inventario button and Annulla in lista maggazino page
      And I click on Aggiorna inventario button
      And I check elements on inventario alert
      And I check Aggiorna inventario button
      And I click Annulla button on alert

  @listamagazzino_page
    Scenario: I sign out from lista magazzino page
      And I click on name-image from catalogue page
      And I click on sign-out button from catalogue page
      Then I check login page from catalogue page
```

Figure 4.2 Sample implementation of automated UI test I

Test cases work independently of each other and are made as atomic as possible. This strategy was used to ensure that test cases would not fail because of the influence of each other.
The main part in making the tests atomic was using a new driver for each test case, meaning opening and closing a new window of the browser before and after each test. This made the tests slower, but ensured that the test cases would not cause each other to crash because they were using the same driver. Initially the same driver was used for each suite of tests, this made the tests faster but also meant that a domino effect would arise if one of the tests failed and the driver ended up at the wrong place in the browser automation.

One of the best practices to write the test cases from scratch is page object model (POM) ,using this methods could take the advantage of developing stable test cases and to have as much control as possible during the development since it is a possible foundation for future development of tests for the UI.
According to Selenium documentation during the implementation of the test cases the most efficient attributes on HTML elements is ID of element. If implementation of your UI doesn't provide you the feature to have a unique ID for all the elements on a page, you could use attributes like XPath and CSS selector attributes. A code example for locating an element using an XPath can be seen in figure 4.3. You could see a more detailed code by going through the appendix.

```
public class ListaMagazzinoPageStepDefs {

    public TestBaseClass script;
    String url = TestBaseClass.baseUrl;

    @Before("@listamagazzino_page")
    public void setup() { script = new TestBaseClass(); }

    @After("@listamagazzino_page")
    public void tearDown() {
        script.tearDown();
    }

    @Given("^I have this users data for listamagazzino page:$")
    public void lista_magazzino_page(DataTable dataUser) throws Throwable {

        List<List<String>> data = dataUser.raw();

        script.goToHomePage(url);
        script.testPausePageUntilDocumentIsReady();
        script.testParamPausePage(5000);

        script.testSendKeyId("username", data.get(1).get(0));
        script.testSendKeyId("password", data.get(1).get(1));
        script.testClickButtonLinkId("signIn");
        script.testPausePage();
    }
```

Figure 4.3 Sample implementation of automated UI test II

4.5 API Test Automation Using SoapUI

API test automation ensures consistency in testing and enables continuous improvement in the software quality. In addition API test automation has the potential of significantly accelerating the testing and development process. Automation enables API tests to be executed either at pre-determined intervals or to be triggered by an event, like code commit. Increasingly testing and development teams are moving towards API test automation and integrating their testing tools with continuous integration (CI) frameworks like Jenkins, Travis CI.
Hence there are a various set of tools with remarkable features for API test automation, between them SoapUI from SMARTBEAR provides an easy to use 'point and click' capability that lets teams to bring REST and SOAP tests into automation platforms easily and without writing complicated scripts. SoapUI Pro includes a configurable command-line interface tool, this enables you to run your tests from any task scheduler or as an integrated part of your build process. [36]

Here we provide an example of creating test automation over a REST API using SOAPUI and Jenkins. Below are the required steps to set up a test suite and integrate it with automation frameworks.

Step 1: Create a new project in the SoapUI dashboard and add you REST data Model
Step 2: Once you've added your Rest data model, SoapUI will check the web service and return all the operations/methods you can call on that service, now you could generate your test suite, when we make sure that SoapUI can send requests to our web service and return a response, we could set up test steps and write test script, finally when you have created the API test suite, run it to ensure that the test is configured correctly.

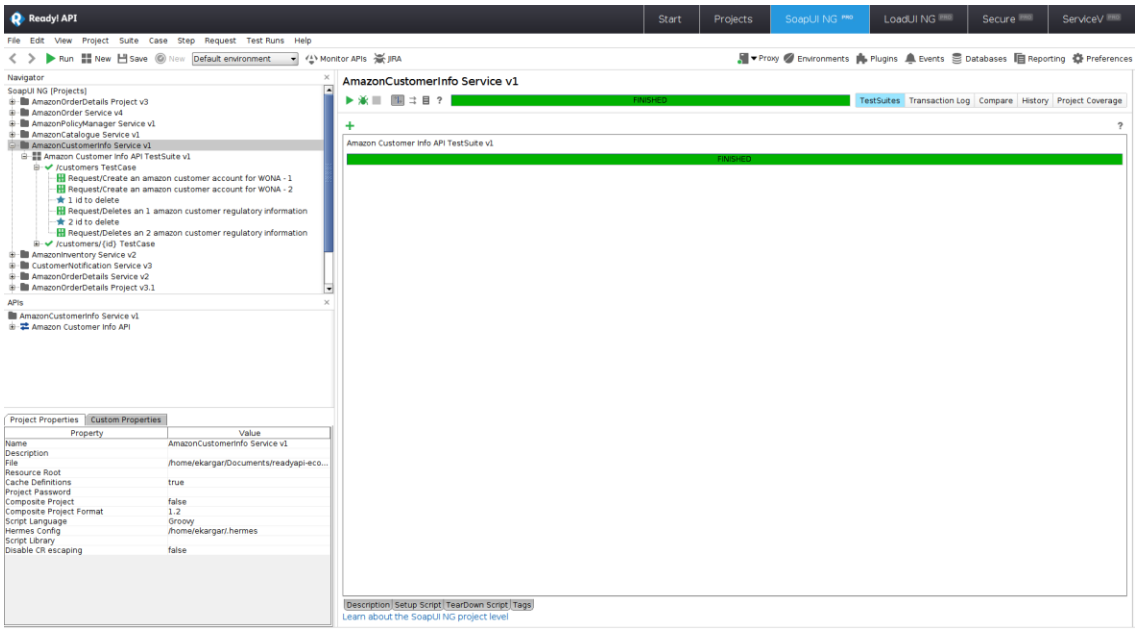


Figure 4.4 An Example of API Automation test

Here is an example of script has been written in Groovy as one of the most used scripting language for writing test scripts.

```groovy
import org.apache.commons.lang.math.RandomUtils
import java.util.Random
import org.apache.commons.lang.math.RandomUtils
import java.util.Random
import groovy.json.JsonSlurper
import groovy.json.JsonOutput

def couponUidList = context.expand( '${getallcouponuidfromdb#Response#$[\'coupons\']}' )

def dataFromJson = new JsonSlurper().parseText(couponUidList)

Integer randomInteger = RandomUtils.nextInt(new Random(),dataFromJson.size())

def locked = true
/////
def couponObject = dataFromJson[randomInteger]
if(couponObject.locked.equals(false)) {
    locked;
} else {
    locked = false;
}
////
def bodyUpdateStatus = '''
{
  "couponUID": '''+ "${couponObject.couponUID}".toInteger() + ''',
  "lock": '''+ "${locked}".toBoolean() + '''
}'''

testRunner.testCase.testSuite.getTestCaseByName("TestCase 1-Patch-on-order-to-lock-unlock").getTestStepByName("Patch - Request 1").setPropertyValue("Request", bodyUpdateStatus)
```

Figure 4.5 an example of automation script written in Groovy

Step 3: In order to make the process of testing API automated, we need to integrate our project with Jenkins, to do this download the pre-built API automation script from SoapUI Interface, create a job in Jenkins by setting up a name and downloaded build parameters. Furthermore at this stage you could automate the job in Jenkins to be executed periodically, you might set up the result of every execution to be delivered as a HTML and graphical report.

Conclusion:

Software automation testing has a prime importance in software's verification and validation. It is important because of two main reasons, first, it assures software quality, and second, nearly 60% of the total software's cost is spend over different types of testing.
Although automation of test cases have a high implementation and maintenance costs, from our experiments we have found that, automation of test cases can give remarkable returns in the long runs where we run and rerun the automated-tests, multiple times. We have also found that test automation has positive effects on software quality. Hence we can claim that test automation increases the overall effectiveness of the testing process when we have repetitive testing tasks which are similar. This work can be extended in future by adding more variable automation cost factors in the analysis to make it more precise and accurate.

Chapter 5 Conclusion

Software industry is extremely disruptive, during working on this thesis we have recognized that how some of the new methodologies and tools in software industry could have a massive impact on software development lifecycle.

Furthermore, we have learned how considering quality as a critical factor in software product lifecycle could enable teams to leverage the risk and increase the chance of launching a successful software product. To have a clear understanding of how we could measure quality of a product, we overviewed some of the most critical metrics and their impact on the software development process of an organization.

We saw how agile manifestation and its technique including scrum, Kanban, extreme programming, Test-Driven development, Behavior-Driven Development etc. could affect the process of developing a software product mostly in quality perspective.

In the third chapter, we have discussed the most relevant performance principals in software development lifecycle. Based on these observations and practices we figure out the list of the critical software performance metrics and their impact on software quality ,particularity we saw how considering these set of metrics in software development process in advance could help us to identify root cause of the problems and leverage the risk of launching a successful product.

Test automation, beyond a shadow of doubt are superior to manual testing, thus in chapter 4 we analyzed how automated testing process could result in continuous quality improvement of a software product. Despite the fact that a wide range of quality assurance process including documentation, frontend or backend testing could be automated, in this chapter we provide two examples on automated user interface and API testing.

This study was the result of practicing some of the software quality techniques during my internship by working on Microservice as product, it covered a broad range of aspects in software quality domain. Although finding an applicable technique ensured launching successful software relies on many factors including organization process, size of the project, customer needs etc., a possible future work in this regard could be an examination on each of above-mentioned agile techniques in a comparative framework in order to find deploying which methods may help us to launch software product in agile environment. Other studies can be conducted to find a comprehensive agile quality model considering critical software quality metrics.

References

1- *Kumlander, D. (2010). Towards a New Paradigm of Software Development: an Ambassador Driven Process in Distributed Software Companies. Advanced Techniques in Computing Sciences and Software Engineering, 487-490.*
2- *By Bijay K. Jayaswal and Peter C. Patton, Software Development Methodology Today. Available from World Wide Web: http://www.informit.com/articles/article.aspx?p=605374*
3- *CLOUDRAMBLINGS , SOFTWARE DEVELOPMENT IN DISRUPTION – THE NEW PARADIGM OF SOFTWARE. Available from World Wide Web: https://cloudramblings.me/2015/09/30/software-development-in-disruption-the-new-paradigm-of-software/*
4- *Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2017). Agile software development methods: Review and analysis. arXiv preprint arXiv:1709.08439.*
5- *Agile Modeling, AM Throughout the XP Lifecycle. Available from World Wide Web: http://www.agilemodeling.com/essays/agileModelingXPLifecycle.htm*
6- *Scrumguides, The Scrum Guide. Available from World Wide Web: http://www.scrumguides.org/scrum-guide.html#definition*
7- *Anand, R. V., & Dinakaran, M. (2016). Popular Agile Methods in Software Development: Review and Analysis. International Journal of Applied Engineering Research, 11(5), 3433-3437.*
8- *David Anderson,Getting Started with Kanban for Software Development, Available from World Wide Web: https://dzone.com/refcardz/getting-started-kanban*
9- *Rochimah, S., Arifiani, S., & Insanittaqwa, V. F. (2015). Non-Source Code Refactoring: A Systematic Literature Review. International Journal of Software Engineering and Its Applications, 9(6), 197-214.*
10- *Michael Lynch, How to Do Code Reviews Like a Human . Available from World Wide Web: https://mtlynch.io/human-code-reviews-1/*
11- *Kim, M., Zimmermann, T., & Nagappan, N. (2014). An empirical study of refactoringchallenges and benefits at microsoft. IEEE Transactions on Software Engineering, 40(7), 633-649.*
12- *Al Dallal, J., & Abdin, A. (2017). Empirical Evaluation of the Impact of Object-Oriented Code Refactoring on Quality Attributes: A Systematic Literature Review. IEEE Transactions on Software Engineering.*
13- *Jameson Nyeholt , What is Pair-Programming and why do we do it?. Available from World Wide Web: https://www.jamasoftware.com/blog/pair-programming-why-we-do-it/*
14- *Cockburn, A., & Williams, L. (2000). The costs and benefits of pair programming. Extreme programming examined, 223-247.*
15- *Hamdan, S., & Alramouni, S. (2015). A quality framework for software continuous integration. Procedia Manufacturing, 3, 2019-2025.*
16- *Zimmermann, O. (2016). Microservices tenets: agile approach to service development and deployment. Computer Science-Research and Development, 32(3), 301-310.*
17- *Hasselbring, W., & Steinacker, G. (2017, April). Microservice Architectures for Scalability, Agility and Reliability in E-Commerce. In Software Architecture Workshops (ICSAW), 2017 IEEE International Conference on (pp. 243-246). IEEE.*
18- *730-2014 - IEEE Standard for Software Quality Assurance Processes,Revision of IEEE Std 730-2002 (Revision of IEEE Std 730-1998)*
19- *26- International Organization for Standardization, Quality management principles, 2015 | Edition: 2 .Available from World Wide Web: https://www.iso.org/files/live/sites/isoorg/files/archive/pdf/en/qmp_2012.pdf*
20- *McBreen, P., Quality Assurance and Testing in Agile Projects, McBreen Consulting, [online]. Available from World Wide Web: http://www.mcbreen.ab.ca/talks/CAMUG.pdf*
21- *Ambler, S. (2005). Quality in an agile world. Software Quality Professional, 7(4), 34.*
22- *Gaffney Jr, J. E. (1981, January). Metrics in software quality assurance. In Proceedings of the ACM'81 conference (pp. 126-130). ACM.*
23- *Defining Agile Software Quality Assurance, E. Mnkandla, Member, IEEE, Proceedings of the International Conference on Software Engineering Advances (ICSEA'06)*

24- *Daka, E., & Fraser, G. (2014, November). A survey on unit testing practices and problems. In Software Reliability Engineering (ISSRE), 2014 IEEE 25th International Symposium on (pp. 201-211). IEEE.*
25- *Gáspár Nagy and Seb Rose,The BDD Books - Discovery ,Explore behaviour using examples.*
26- *Ivanko, J. (2011). A proposal of regression testing process for a small organization developing COST software (Doctoral dissertation, Masarykova univerzita, Fakulta informatiky)*
27- *Usability Testing. Available from World Wide Web: https://www.usability.gov/how-to-and-tools/methods/usability-testing.html*
28- *Performance Testing Tutorial: Types, Process & Important Metrics. Available from World Wide Web: https://www.guru99.com/performance-testing.html*
29- *Sajid Manzor ,Web applications Performance Testing Metrics. Available from World Wide Web: http://www.agileload.com/agileload/blog/2013/02/18/web-applications-performance-testing-metrics*
30- *Meier, J., Farre, C., Bansode, P., Barber, S., & Rea, D. (2007). Performance testing guidance for web applications: patterns & practices. Microsoft press.*
31- *STACKIFY,Top Load Testing Tools: 50 Useful Tools for Load Testing Websites, Apps, and More. Available from World Wide Web: https://stackify.com/top-load-testing-tools/*
32- *Kumar, D., & Mishra, K. K. (2016). The Impacts of Test Automation on Software's Cost, Quality and Time to Market. Procedia Computer Science, 79, 8-15.*
33- *Premal Dave,Top 10 Automated Software Testing Tools. Available from World Wide Web: https://dzone.com/articles/top-10-automated-software-testing-tools*
34- *Yadav, A. P., & Kumar, A. (2015). AN AUTOMATION TESTING USING SELENIUM TOOL.Chicago*
35- *Selenium Documentation, Test Automation for Web Applications. Available from World Wide Web: http://www.seleniumhq.org/docs/01_introducing_selenium.jsp*
36- *SMARTBEAR,Automated testing for REST and SOAP API. Available from World Wide Web: https://smartbear.com/product/ready-api/soapui/features/automated-api-testing/*

Appendix

This section includes some coding examples, reports and in-place tools I've experienced during my work.

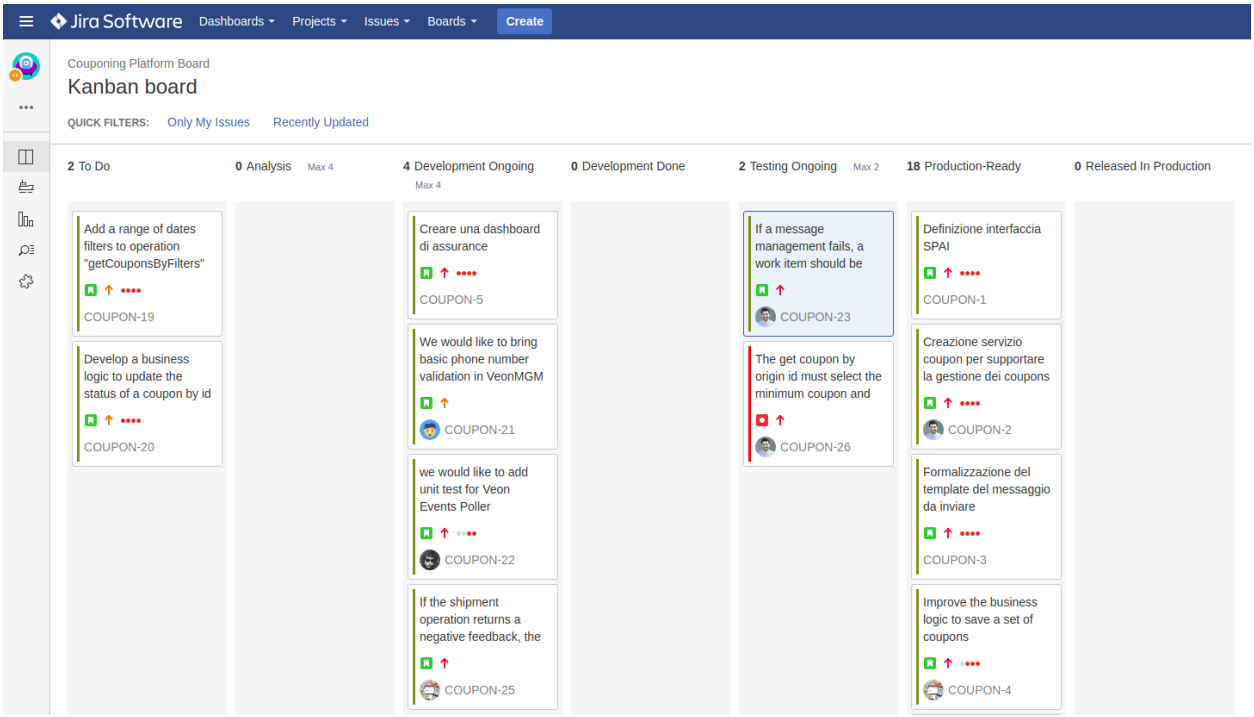


Figure 5 an Example of a project Kanban board

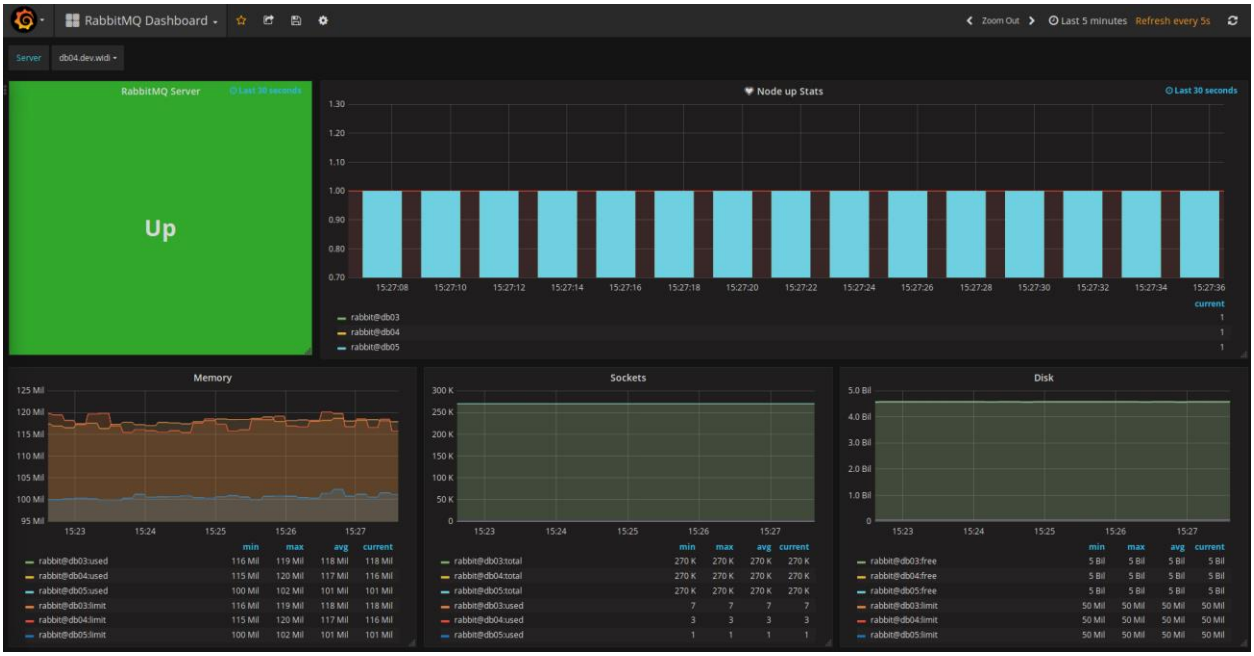


Figure 6 Microservice Monitoring Dashboard using Grafana

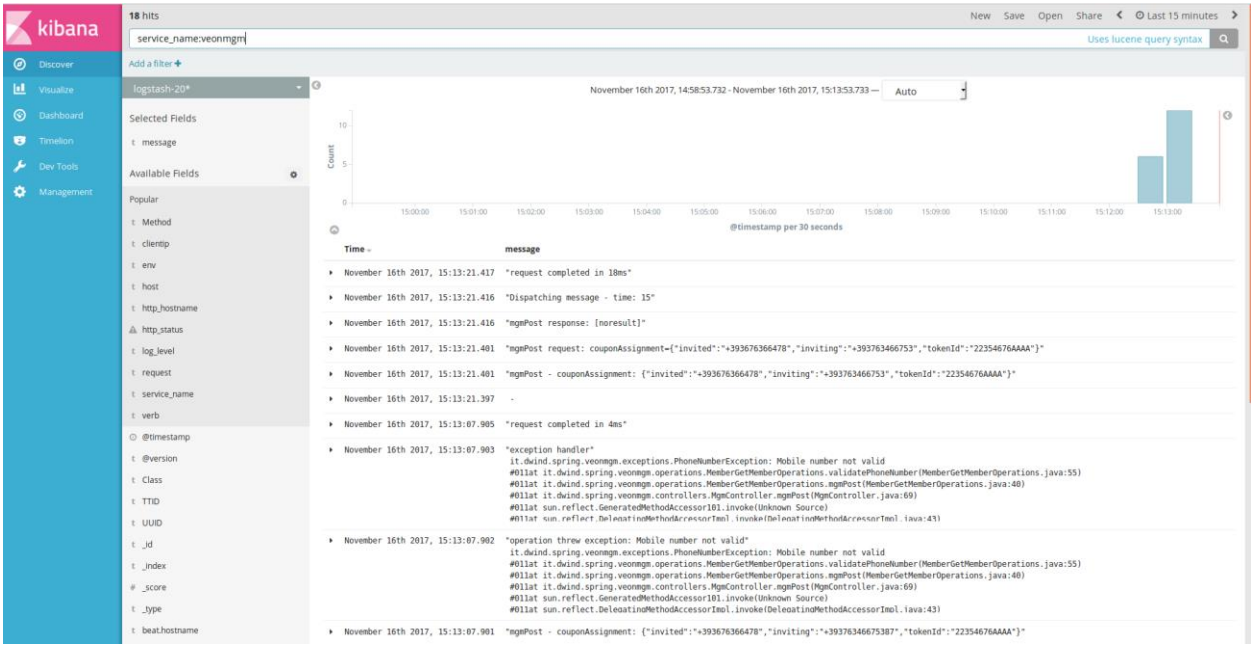


Figure 7 Kibana - a data visualization and log exploration tool

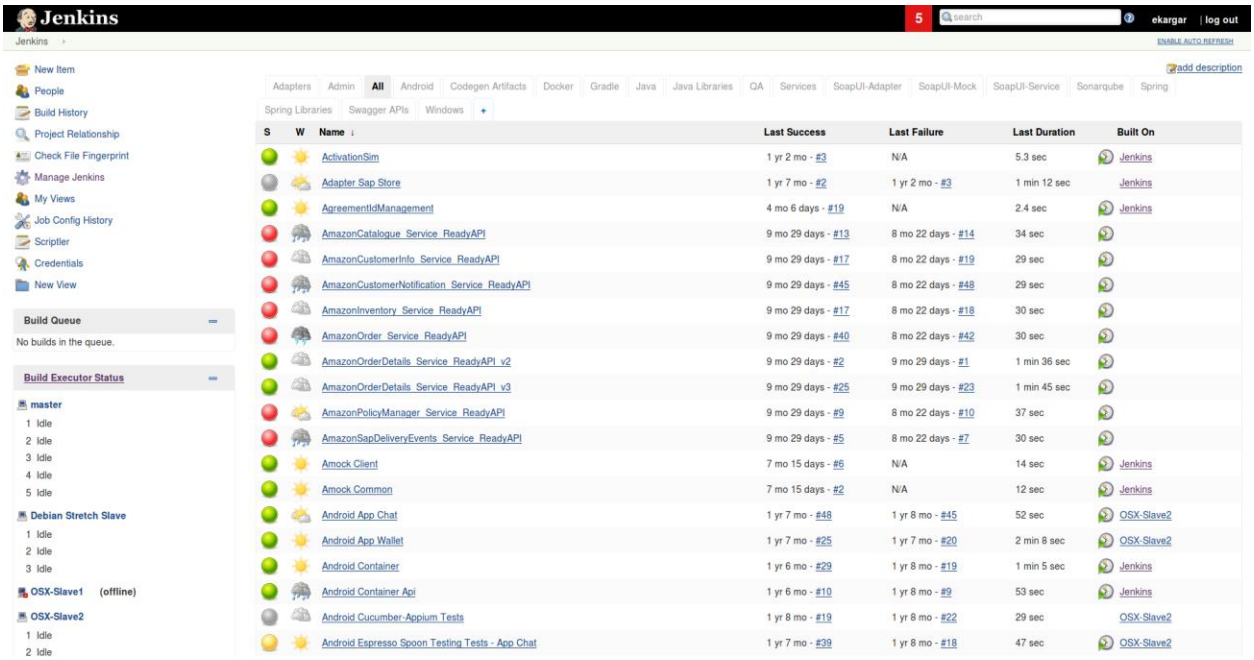


Figure 8 Jenkins - Continuous Integration tool in-place

```
import org.mockito.ArgumentCaptor;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.runners.MockitoJUnitRunner;
import static org.junit.Assert.*;
import static org.mockito.Matchers.any;
import static org.mockito.Mockito.*;

/**
 * Created on 18/10/17.
 */
@RunWith(MockitoJUnitRunner.class)
public class AssignmentManagerTest {

    @InjectMocks
    private AssignmentManager mUnit;

    @Mock
    private AssignmentsRepository mAssignmentsRepository;

    @Mock
    private AwardsRepository mAwardsRepository;

    @Mock
    private WorkItemsRepository mWorkItemsRepository;

    @Mock
    private RabbitLifeCycleDelayPublisher mRabbitLifeCycleDelayPublisher;

    //----------------------------- createAssignment -----------------------------
    @Test
    public void testCreateAssignment() {
        int expectedResult = 1;
        int awardResult1 = 2;
        int awardResult2 = 3;

        CouponAssignment couponAssignment = CouponAssignment.builder()
                .invited("invited")
                .inviting("inviting")
                .tokenId("tokenId")
```

```
                .build();

        ArgumentCaptor<Assignment> assignmentCaptor = ArgumentCaptor.forClass(Assignment.class);
        when(mAssignmentsRepository.save(assignmentCaptor.capture())).thenReturn(expectedResult);

        ArgumentCaptor<Award> awardCaptor = ArgumentCaptor.forClass(Award.class);
        when(mAwardsRepository.save(awardCaptor.capture()))
                .thenReturn(awardResult1)
                .thenReturn(awardResult2);

        ArgumentCaptor<WorkItem> workItemCaptor = ArgumentCaptor.forClass(WorkItem.class);
        when(mWorkItemsRepository.save(workItemCaptor.capture())).thenReturn(0);

        Integer result = mUnit.assignmentOperation(couponAssignment);

        verify(mAssignmentsRepository, times(1)).save(any(Assignment.class));
        assertNotNull(assignmentCaptor.getValue());
        assertEquals(assignmentCaptor.getValue().getExternalId(), couponAssignment.getTokenId());

        verify(mAwardsRepository, times(2)).save(any(Award.class));
        assertNotNull(awardCaptor.getAllValues());

        assertNotNull(awardCaptor.getAllValues().get(0));
        checkAwardSave(awardCaptor.getAllValues().get(0), expectedResult, couponAssignment);

        assertNotNull(awardCaptor.getAllValues().get(1));
        checkAwardSave(awardCaptor.getAllValues().get(1), expectedResult, couponAssignment);

        verify(mWorkItemsRepository, times(2)).save(any(WorkItem.class));
        assertNotNull(workItemCaptor.getAllValues());

        assertNotNull(workItemCaptor.getAllValues().get(0));
        checkWorkItemSave(workItemCaptor.getAllValues().get(0), awardResult1, awardResult2);

        assertNotNull(workItemCaptor.getAllValues().get(1));
        checkWorkItemSave  (workItemCaptor.getAllValues().get(1), awardResult1, awardResult2);

        assertTrue(result == expectedResult);
    }

    private void checkAwardSave(Award award, Integer expectedAssignUid, CouponAssignment
couponAssignment) {

        assertNull(award.getCouponUid());
        assertEquals(award.getAssignUid(), expectedAssignUid);
        PartId partId = PartId.valueOf(award.getPartId());

        switch (partId) {
            case INVITED:
                assertEquals(award.getDestination(), couponAssignment.getInvited());
                break;
            case INVITING:
                assertEquals(award.getDestination(), couponAssignment.getInviting());
                break;
            default:
                assertTrue(false);
        }
    }

    private void checkWorkItemSave(WorkItem workItem, Integer awardUid1, Integer awardUid2) {
        assertTrue(workItem.getAwardUid().equals(awardUid1) ||
workItem.getAwardUid().equals(awardUid2));
        assertNotNull(workItem.getItemType());
        assertEquals(workItem.getItemType(), WorkItemType.ASSIGNMENT.getCode());

        assertNotNull(workItem.getResultSuccess());
        assertTrue(workItem.getResultSuccess());
    }

}
```

An example of Unit Test

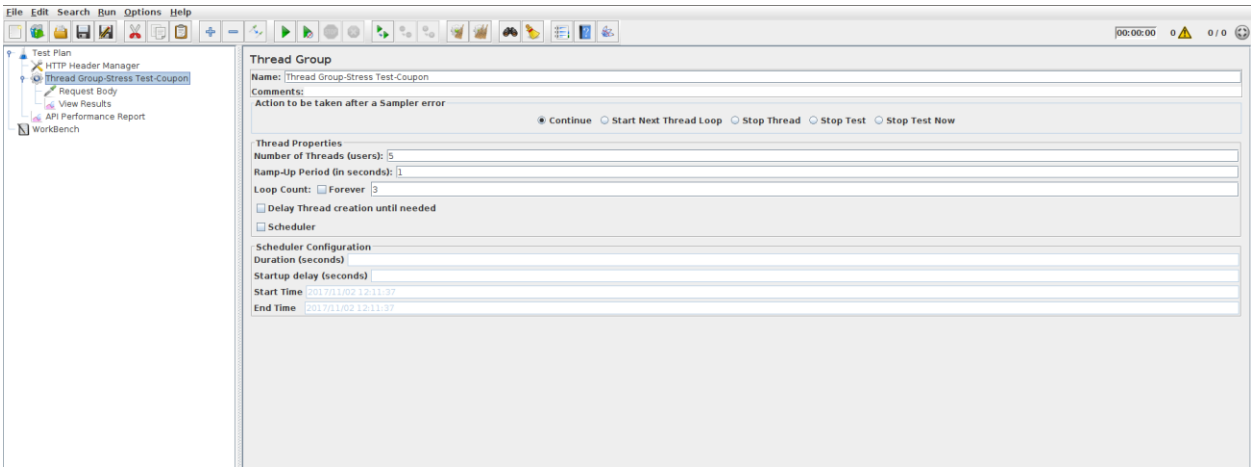


Figure 9 Apache Jmeter -An open source tool for software performance testing

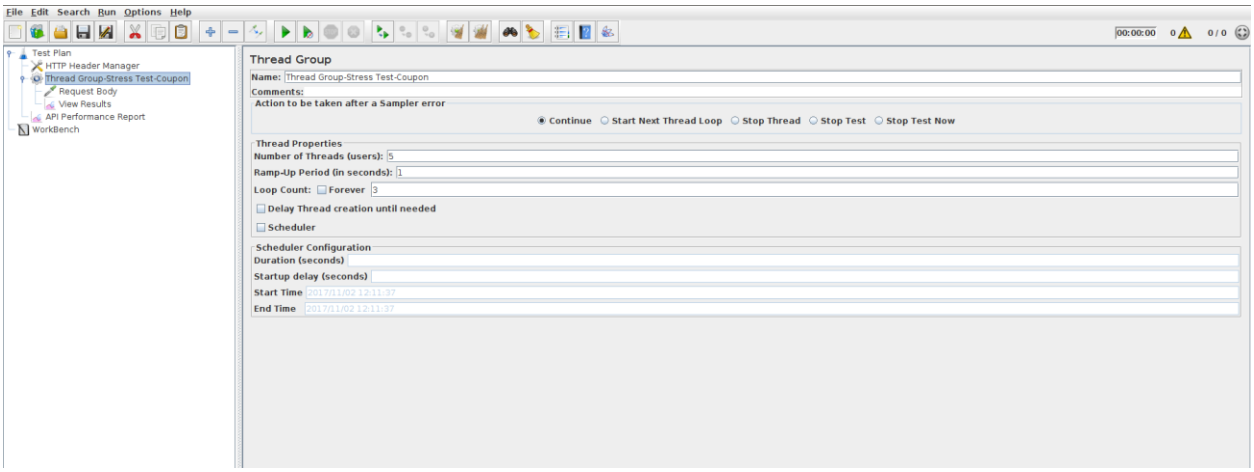


Figure 10 an example of performance test case in Jmeter

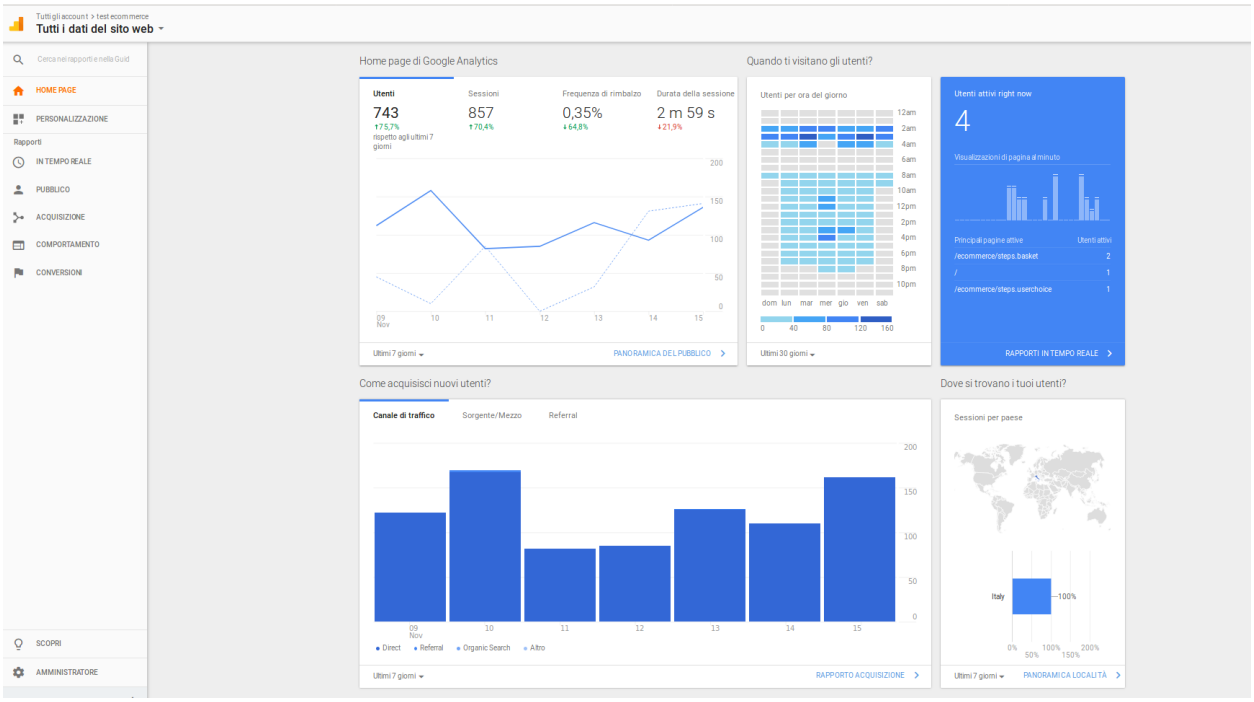


Figure 11 Usability Test using Google Analytics

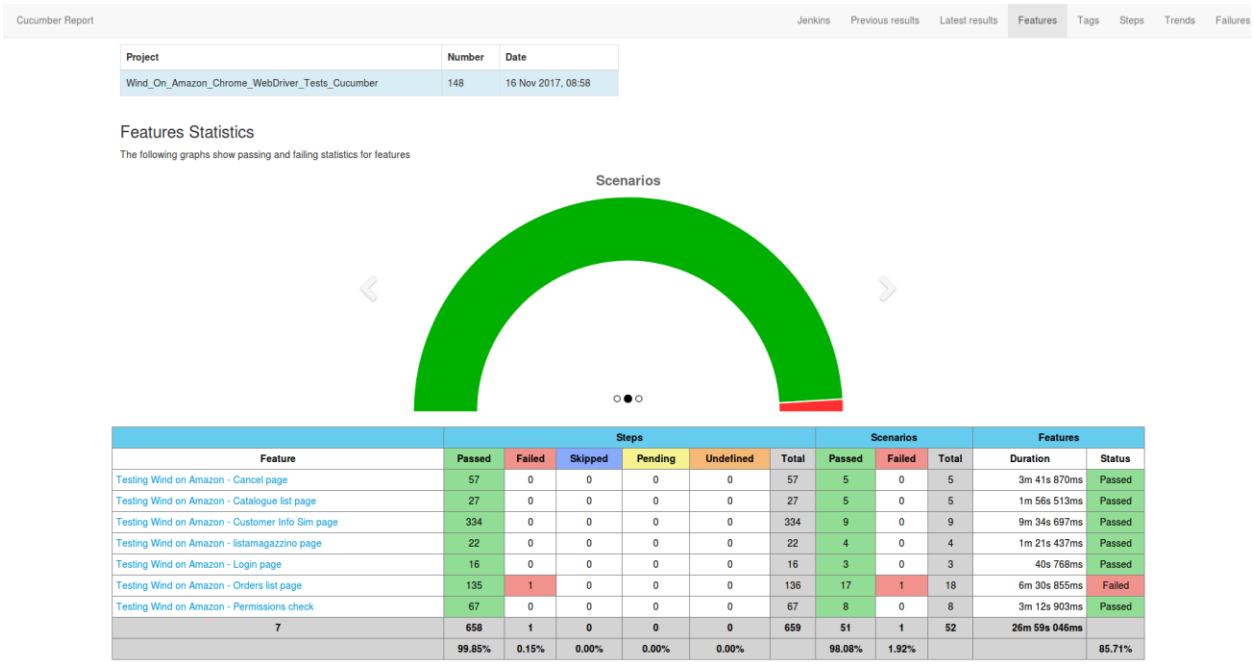
```
Feature: Testing Wind on Amazon - listamagazzino page

  Background:
    Given I have this users data for listamagazzino page:

      | USERNAME       | PASSWORD |
      | wona_abox_root | Wroot1!  |

    When I land on listamagazzino page and title is as expected

  @listamagazzino_page
  Scenario: I click on Aggiorna inventario button and Annulla in lista maggazino page
      And I click on Aggiorna inventario button
      And I check elements on inventario alert
      And I check Aggiorna inventario button
      And I click Annulla button on alert

  @listamagazzino_page
  Scenario: I click on Aggiorna inventario button and Aggiorna in lista magazzino page
    And I click on Aggiorna inventario button
    And I check elements on inventario alert
    And I check Annulla button
    And I click Aggiorna button on inventario alert
    Then I click ok button on Aggiornamento completato alert

  @listamagazzino_page
    Scenario: I sign out from lista magazzino page
      And I click on name-image from catalogue page
      And I click on sign-out button from catalogue page
      Then I check login page from catalogue page

  @listamagazzino_page
  Scenario: I land on listamaggazino page and buttons are enabled
    And I check Aggiorna inventario button is enabled
    And I check Esporta csv button is enabled
```

An example of Scenario file in Gherkins used for UI automation test by Selenium

```java
import cucumber.api.DataTable;
import cucumber.api.PendingException;
import cucumber.api.java.After;
import cucumber.api.java.Before;
import cucumber.api.java.en.And;
import cucumber.api.java.en.Given;
import cucumber.api.java.en.Then;
import java.util.List;

/**
 * Created by ekargar on 24/05/17.
 */
public class ListaMagazzinoPageStepDefs {

    public TestBaseClass script;
    String url = TestBaseClass.baseUrl;

    @Before("@listamagazzino_page")
    public void setup() { script = new TestBaseClass(); }

    @After("@listamagazzino_page")
    public void tearDown() {
        script.tearDown();
    }

    @Given("^I have this users data for listamagazzino page:$")
    public void lista_magazzino_page(DataTable dataUser) throws Throwable {

        List<List<String>> data = dataUser.raw();

        script.goToHomePage(url);
        script.testPausePageUntilDocumentIsReady();
        script.testParamPausePage(5000);

        script.testSendKeyId("username", data.get(1).get(0));
        script.testSendKeyId("password", data.get(1).get(1));
        script.testClickButtonLinkId("signIn");
        script.testPausePage();
    }


    @And("^I click on Aggiorna inventario button$")
    public void i_Click_On_Aggiorna_Inventario_Button() throws Throwable {
        script.testPausePage();
        script.testClickButtonLinkId("inventoryUpdate");
    }

    @And("^I check elements on inventario alert$")
    public void i_check_elements_on_inventario_alert() throws Throwable {
        script.testPausePage();
        script.assertElementTextEqualsXpath(".//*[@id='gwt-uid-11']", "Aggiornamento
inventario");
        script.assertElementTextEqualsXpath("//div[3]/div/div/div[3]/div/div/div[1]/div",
"L'operazione di aggiornamento dell'inventario non è reversibile.");
        script.assertElementTextEqualsXpath("//div[3]/div/div/div[3]/div/div/div[3]/div", "Sei
sicuro di volerla eseguire?");
    }


    @And("^I check Aggiorna inventario button$")
    public void i_Check_Aggiorna_Inventario_Button() throws Throwable {
        script.testPausePage();
        script.assertElementTextEqualsId("updateInventory", "Aggiorna");
    }

    @And("^I check Annulla button$")
    public void i_Check_Annulla_Button() throws Throwable {
       script.testPausePage();
       script.assertElementTextEqualsId("cancel", "Annulla");
    }
```

```
    @And("^I check elements on Aggiornamento completato alert$")
    public void i_Check_elements_On_Aggiornamento_Completato_Alert() throws Throwable {

script.assertElementTextEqualsXpath("//div[3]/div/div/div[3]/div/div/div[1]/div","Aggiornamento
completato");
        script.testPausePage();
    }

    @And("^I click ok button on Aggiornamento completato alert$")
    public void i_click_ok_button_on_Aggiornamento_completato_alert() throws Throwable {
        script.testPausePage();
        script.testClickButtonLinkId("updateInventory");
    }


    @And("^I click Aggiorna button on inventario alert$")
    public void i_Click_Aggiorna_Button_On_Inventario_Alert() throws Throwable {
        script.testPausePage();
        script.testClickButtonLinkId("updateInventory");
    }

    @And("^I check Esporta csv button is enabled$")
    public void i_Check_EsportaCsv_Button_enabled() throws Throwable {
        script.testPausePage();
        script.testIsButtonEnabledXpath(".//*[@id='wona-
3655307']/div/div[2]/div/div/div/div[3]/div/div[2]/div/div/div[1]/div/div[2]/div/div/div/div/div[
1]/div");
    }

    @And("^I check Aggiorna inventario button is enabled$")
    public void i_Check_Aggiorna_inventario_Button_enabled() throws Throwable {
        script.testPausePage();
        script.testIsButtonEnabledId("inventoryUpdate");
    }
}
```

An example of UI automation test using Selenium & Java



	Steps						Scenarios			Features	
Feature	Passed	Failed	Skipped	Pending	Undefined	Total	Passed	Failed	Total	Duration	Status
Testing Wind on Amazon - Cancel page	57	0	0	0	0	57	5	0	5	3m 41s 870ms	Passed
Testing Wind on Amazon - Catalogue list page	27	0	0	0	0	27	5	0	5	1m 56s 513ms	Passed
Testing Wind on Amazon - Customer Info Sim page	334	0	0	0	0	334	9	0	9	9m 34s 697ms	Passed
Testing Wind on Amazon - listamagazzino page	22	0	0	0	0	22	4	0	4	1m 21s 437ms	Passed
Testing Wind on Amazon - Login page	16	0	0	0	0	16	3	0	3	40s 768ms	Passed
Testing Wind on Amazon - Orders list page	135	1	0	0	0	136	17	1	18	6m 30s 855ms	Failed
Testing Wind on Amazon - Permissions check	67	0	0	0	0	67	8	0	8	3m 12s 903ms	Passed
7	658	1	0	0	0	659	51	1	52	26m 59s 046ms	
	99.85%	0.15%	0.00%	0.00%	0.00%		98.08%	1.92%			85.71%

Figure 12 Automated regression Test results