**Project Report**

**On**

**Scrummy**

**CSC402 Software Engineering**

**Submitted by**

**Forbes, Kamile A(0888590)**

**Schaefer, Zachary S(0881121)**

**Dinesh Gonuguntla(0918700)**

**Table of Contents**

**1.Introduction to framework**

**2.Team structure**

1. **Scrum Master**
2. **Product Owner**
3. **Development Team**
4. **Testing team**

**3.Process Overview**

**4.Planning Phase**

1. **Define the project**
2. **Define architecture**
3. **Build product backlog**
4. **Move items from product backlog to sprint backlog**

**5.Sprint Phase**

1. **Daily meetings**
2. **Development work**
3. **Sprint review**
4. **Sprint retrospective meeting**

**6. Security Requirements**

**7. QA**

**8. Testing**

**9. Improving and adapting the framework**

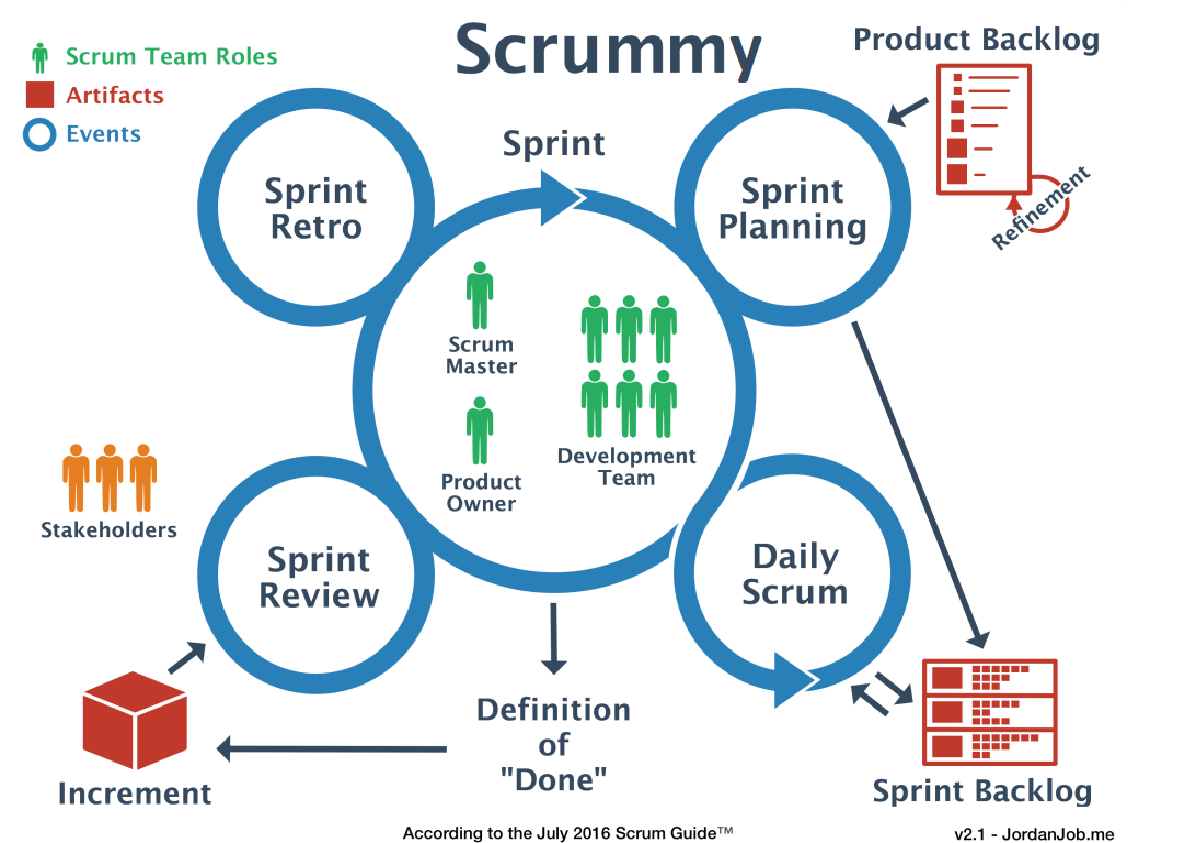
**Introduction to the Framework:**

Scrummy was designed to aid organizations in quickly developing projects without sacrificing quality standards. Scrummy is based off of Scrum, and as such it employs an iterative approach to the development cycle. It also has a focus on interactions between the team and the customer to ensure that the product successfully fulfills the evolving needs of the customer.

The framework utilizes short iterative cycles to not only guarantee that employees are working efficiently but also to allot the ideal amount of time to each module. The iterative approach can be a massive benefit, as it is not uncommon for product goals and requirements to change during development. Scrummy is easy to use but difficult to master, which is beneficial because it is not hard to adopt Scrummy yet affords individuals the opportunity to continually improve their skills as they become more proficient in using it. However, like Scrum, Scrummy is a very teamwork-oriented framework, meaning that teamwork is completely essential to successfully adopting and incorporating the Scrummy framework into an organization.

The cycle employed by the framework inherits phases from Scrum, such as the Planning and Sprint phases, which means the workflow is similar to that of Scrum. However, the framework also possesses teams that perform testing as well as QA, which further enhance the organization's abilities to find and squash bugs as well as meet security and efficiency needs. The testing team is present full time and reviews code (both code that is in development and code for completed modules) during the entire development process, which avoids taking too much time away from the developers and allows for increased efficiency. The dedicated testing team works out as many bugs as possible so that the developers can spend more time working on the project. The testing team views and tests the modules to ensure that they meet the quality and efficiency needs of the customer. At the end of each iteration, a team meeting is held, which is another concept inherited from Scrum. The customer is brought on board following major development milestones to showcase the recently completed work, affording the customer the ability to test the program to see whether or not it successfully meets their needs. If it does not meet their needs, their input is recorded and will be incorporated into the next development cycle.

Scrummy is a framework that encourages teams to utilize shorter sprints so that they can gather more information about their development cycles, allowing them to improve their processes more quickly than in other frameworks. Scrummy provides numerous other benefits, such as aiding in the understanding of viable markets and product capabilities. Another major benefit is that it enables organizations to develop and sustain environments for their products by following the Scrummy processes.



**Agile Methodology- Scrum Framework**

**Introduction to Agile Methodology: -**

To Overcome the slowness and adapting to frequent changes in developing a software a new methodology came to existence and named as Agile. In simple terms, Agile methodology deals with the below points.

* Communication or interactions over the developing software or people involved through various processes and tools.
* Gives more importance on the actual software or product going to be delivered instead of documenting the procedures followed.
* Adapting to the frequenting changes that are proposed by the customer through planning on daily basis.
* It works on value driven approach providing incremental delivery in weekly basis and responsive to the frequent changes which leads to customer satisfactions.

**Advantages of Agile Methodology:**

* High Revenue income due to faster and high-quality delivery.
* Mitigation of risk amount is less due to frequent interactions
* Informal inspection at various intervals leads to proper shape and idea about the quality.
* Product delivery in incremental way leads to more customer satisfaction.

**Flavors of Agile Methodology:**

Agile generally doesn’t follow any concrete steps or methods to build a framework or project. To overcome this failure, it borrowed many principles form lean manufactures and created an Agile manifesto.

Few methods generally an organization chooses to carry forward the Agile Movement includes

* **Scrum**

Light weight with specific set of guidance, flexible and easy to cope up at any point of time due to high level of transparency.

* **Kanban**

Complete organization of workflow structure like maintaining visualization, setting up various limits, and providing proper details related to workflow. Its works mainly on workflow structure.

* **Extreme Programming**

Generally named as XP, it works with specific engineering practices with proper coding, high level logic and mainly used in extreme projects as name refers.

* **Crystal Clear**

As name indicates it mainly focus on people, artifacts, easy access through frequent integration.

**Scrum**

Scrum is a framework that helps teams work together. Much like a rugby team (where it gets its name)

training for the big game, Scrum encourages teams to learn through experiences, self-organize while

working on a problem and reflect on their wins and losses to continuously improve.

While the Scrum I’m talking about is most frequently used by software development teams, its

principles and lessons can be applied to all kinds of teamwork. This is one of the reasons Scrum is so

popular. Often thought of as an agile project management framework, Scrum describes a set of

meetings, tools, and roles that work in concert to help teams’ structure and manage their work.

Scrum specifies three major roles that play a part in the Scrum Team: Product Owner, Scrum Master, and Development team member. Besides these roles, you should also expect to have Stakeholders. In larger enterprises, there are usually (several) Business Analysts (BA) involved in a software implementation. I’ll describe how they all fit into Scrum. Additionally, I’ll introduce the concept of Subject Matter Experts (SME), those people who hold the key to the Scrum team’s success.

**The Scrum Team**

The Scrum team includes the: Product Owner, Scrum Master, and Development team. These individuals share different tasks and responsibilities related to the delivery of the product. Scrum describes this as a self-organizing and cross-functional team. But how can you facilitate this concept within your team? Here’s a list of key characteristics that are important to consider when creating your Scrum team. I’ve described these characteristics based on the ideal scenario:

1. **Self-managing:** Each Scrum team decides how the group will work. Within this team, each member is equally important (no-hierarchy), but responsibilities are clearly defined. This means that each team member should get equal opportunity to voice his or her opinion. Together, they can then form a solution. Ultimately, the Product Owner gets the final say about prioritization, but all other discussions are guided by the Scrum Master to a solution everybody agrees with.
2. **Cross-functional:** The team should possess all knowledge required to deliver a working product. This does not include business knowledge but does include sufficient knowledge about: QA, UX, integrations, etc. This does not mean that each team member should be the perfect developer and have all of this knowledge, but this knowledge needs to be spread across team members.
3. **Co-located:** Scrum is all about close collaboration. Ideally, the entire team would be sitting in the same room so that there are no barriers (no matter how small) to communication. When team members are spread out over different rooms, locations, or time zones, it is normal for people to postpone their interaction.
4. **Dedicated:** Every member of the Team should be assigned to the project full-time as any distraction will just delay work. Focused work is far more effective than switching between assignments or dividing your attention between two projects. Being dedicated to a single project is also the best way to take up ownership and responsibility, (allowing for better self-management).
5. **Long-lived:** Try to keep teams together. Don’t make frequent changes in teams. New Scrum teams take time to learn to work together; therefore, making frequent changes (even between projects) requires time for the team to learn to work together.

**What do I mean by cross-functional?**

In an ideal world, your Scrum team is cross-functional and all knowledge for all aspects is available within the team. This includes knowledge about, Testing (QA), styling and User Experience, but also sufficient knowledge about all integrations that might be necessary. However, I realize that it isn’t always realistic for Scrum team members to know especially detailed knowledge about integration with other systems. In this instance, you need to make sure that your team has access to all the knowledge they need. Depending on the size and complexity of the project, it might make sense to include the Integration expert, QA-er, UX-er as a full-time team member. But if you’re working on a small Scrum team, introducing those experts would create too much overhead. In those instances, you need a Subject Matter Expert (SME).

**Introduction of a Subject Matter Expert**

When talking about Subject Matter Experts, we’re usually talking about an experienced business user whom the Scrum team engages with for questions. But from the Scrum team’s perspective, I would describe an SME as: every person that possesses knowledge the team needs for a successful product delivery. For further reference, all SMEs are also considered Stakeholders for this product (but not all Stakeholders have to be SMEs).

This allows us to classify several roles as SMEs, such as a Systems Admin (Infra SME) and UX expert (UX SME). But we can even classify developers, Scrum Masters, or Project Managers from other teams as SMEs for their respective projects. A team member of this Scrum Team can even be an SME for another Scrum Team.

But remember, an SME is not part of the Scrum Team. And as such, this person cannot be held responsible nor accountable for any work they do for the team or Product Owner. This creates some challenges, and you should not resolve this by making an SME responsible for his work. Making an SME responsible could make team members responsible for work done by multiple teams which introduces a much bigger problem because it conflicts with the earlier described team characteristics (numb 1, 2 & 3).

The Scrum team might need the SMEs for different purposes and at different times, but they are all expected to answer questions and perform tasks to improve the product. During the planning meetings, you can identify when information or actions are expected from the SMEs, and a Scrum Team Member can follow up with the SME to complete the action on time to prevent delays to this product delivery.

By using this broad definition of SME, we are setting clear expectations. If the Scrum Team needs a person to help on an activity for this product, that person is seen as an SME and knows what the team expects.

**Scrum Master**

Scrum masters are the facilitators of scrum, the lightweight agile framework with a focus on time-boxed iterations called sprints. As facilitators, scrum masters act as coaches to the rest of the team. “Servant leaders” as the Scrum Guide puts it. Good scrum masters are committed to the scrum foundation and values but remain flexible and open to opportunities for the team to improve their workflow.



**Scrum Master Responsibilities**

In the ideal agile world, the team would manage their own process and tool. Yet we’ve found that many teams making the leap to agile often rely on the scrum master as the owner of their process. It takes time for responsibility and authority to diffuse through a team. In this transformative context, the role can be as lightweight as scheduling the scrum ceremonies or as involved as any other scrum team member. Although the Scrum Guide lists how the scrum master serves other scrum roles, this is not an exhaustive list of responsibilities. Indeed, we find scrum masters often perform some or all of the following, not all of which are defined by scrum:

Standups - Facilitate daily standups (or the daily scrum) as needed.

1. **Iteration/sprint planning meetings –** Protect the team from over-committing and scope creep. Aid in estimation and sub task creation.
2. **Sprint reviews**– Participate in the meeting and capture feedback.
3. **Retrospectives**– Note areas for improvement and action items for future sprints.
4. **Board administration** – Work as the administrator of the [scrum board](https://www.atlassian.com/agile/tutorials/creating-your-agile-board). Ensure that cards are up to date and the scrum tool, [Jira software](https://www.atlassian.com/software/jira) or otherwise, is working well.
5. **1 on 1s** – Meet individually with team members and stakeholders as needed. Iron out team disagreements about process and work styles. While many scrum practitioners are anti-1on1, as they believe these communications should happen during standups, some teams, particularly for new teams, prefer to have these regular face-to-face interactions with specific team members. The scrum master may decide that these individual interactions are crucial for team development and getting to know one another.
6. **Internal Consulting –** Scrum masters should be prepared to consult with team members and internal stakeholders on how best to work with the scrum team.
7. **Reporting**– Regular analysis of [burndown charts](https://www.atlassian.com/agile/tutorials/burndown-charts) and other portfolio planning tools to understand what gets built and at what cadence.
8. **Blockers**– The scrum master aids the team by eliminating external blockers and managing internal roadblocks through process or workflow improvements.
9. **Busy work** – If the scrum team isn’t humming, that’s the scrum master’s problem. Maybe that means fixing broken computers, moving desks around, or even adjusting the thermostat. Scrum masters should be comfortable doing just about anything to help their team and should be not slink away from grabbing coffees or some snacks if that’s what the team really needs.



**Do I need a scrum master?**

Any scrum trainer will teach that a scrum team must have a scrum master. Without one, you are doing something just shy of true scrum, often called scrum-but. When starting out with scrum, it can be a huge help to have someone in the role who has seen scrum working before. Better yet, has seen many examples of it working. For this reason, scrum masters are often hired as consultants, rather than as full-time employees. But every scrum team is different. Many experienced teams handle the responsibilities listed above as a unit and take pride and enjoyment in a shared management of the process. The role of scrum master rotates through the team, with team members facilitating standups and retros in turn. And for some teams, the right thing is just to have the same person play the role every day. Unfortunately, misunderstanding of the scrum master role often leads existing managers to assume it is their role. To better understand why this can be a problem, let’s compare the scrum master to non-scrum roles you may already have in your organization, and why it's important to keep the role separate.

**Scrum Master vs Product Manager**

As we advocate in our [Agile Product Management overview](https://www.atlassian.com/agile/product-management), the more involved that a product manager is with the development team, the better. That involvement should be along the lines of a product owner who champions customer needs, the "why" of the product. When the involvement blurs into tasking, the "how" for a team, then there is a problem. Even with the best of intentions, this kind of utilization mindset tends to hide problems: defects, hand-offs, and unknowns. Interleaving scope and process tends toward locking scope, schedule, and quality. That's a recipe for failure.

That's why the scrum master and product owner fill two different needs on a scrum team, that are often combined with traditional software management. And it's tempting in small teams to avoid the perceived overhead of another role. However, when roadblocks crop up, or changes arise, a clear division between process management and product direction is required.

**Scrum Master vs Project Manager**

The scrum master’s non-technical (or non-agile) counterpart is the project manager. Both of these roles focus on the “how” of getting work done and solve workflow problems through process and facilitation. So, do you need both? Likely not.

Both a traditional project manager and a scrum master are responsible for helping their teams get work done, but their approaches are vastly different. The project manager sets and tracks timeframes and milestones, reports on progress, and coordinates team communication. However, they do so from a place of control, in a more traditional management role.

The scrum master helps the team enhance and streamline the processes by which they achieve their goals. They do so as a team member, or collaborator, ideally not as someone in control. The best scrum teams are self-organizing, and therefore don’t react well to top-down management.

These are just a few of the possible configurations of [scrum team management](https://www.atlassian.com/agile/teams). Some organizations make due with all of these roles, some have one or none.

**Scrum masters and the greater org**

There is one consideration that rises above the rest when thinking about hiring a scrum master: Only do so if your organization is committed to scrum and invested in the process. All the above roles can manage a development team in a myriad of ways, but a scrum master can only be effective with 100% buy-in on scrum. Full stop.

With a scrum master helping every team manage their process, [your entire organization](https://www.atlassian.com/agile/agile-at-scale) can realize some serious gains. On top of shipping value to your customers on a regular basis (the main goal of scrum), teammates and managers will be free to focus on what they do best. Product managers can focus on strategy, developers can write their best code, and Kyle from sales can ring that damn bell. What does that all sound like? It sounds like high-functioning scrum, music to our ears.

**Product Owner**

Product owner is a person who represents the customer, business or user community and is responsible for working with the team to determine what features will be included in the product release. As a liaison between the development team and customers, the product owner must collaborate closely with both groups to ensure there is a clear understanding of what features are needed in the product or application. Because there may be a variety of types of customers and users, the product owner must have a firm understanding of the business domain and the varying needs of different types of users.

The product owner role requires an individual with certain skills and traits, including availability, business savvy and communication skills. First, the Scrum product owner should remain available to the team. The best product owners show commitment by doing whatever is necessary to build the best product possible – and that means being actively engaged with their teams.

Business savvy is important for the agile product owner because he or she is the decision maker regarding what features the product will have. That means, the agile PO should understand the market, the customer and the business in order to make sound decisions.

Finally, communication is a large part of the product owner responsibilities. The product owner role requires working closely with key stakeholders throughout the organization and beyond, so he or she must be able to communicate different messages to different people about the project at any given time.

The Product Owner is the visionary of the project and is responsible for:

* Gathering requirements
* Managing and prioritizing the Product Backlog
* Software acceptance
* Planning the release
* Understand the value of the project

**Skills of Product Owner**

A product owner is usually a CEO, a Domain Specialist, a Project Manager or even a Business Analyst with technical skills. A Product Owner should ideally have a good balance of following skills:

* Domain expertise
* Good technical knowledge
* A decision maker
* Easily available to the team

**Responsibilities of the Product Owner**

The Product Owner decides what will be built and in which order. He or she will:

**Manage Product Backlog and Release Planning**

* Define the features of the product or desired outcomes of the project.
* Adjust features/outcomes and priority as needed to ensure ROI.
* Fully elaborates Acceptance Criteria for User Stories.
* Prioritize User stories according to business value.
* Perform Release planning and ensures that release backlog is aligned with the Vision and Roadmap.
* Reviews backlog in depth with the Scrum Master in preparation for future Release Planning activities.
* Write new User Stories needed in order to have a complete and comprehensive backlog.

**Work closely with the Scrum Team**

* Collaborates with the team to ensure User Stories are accurately elaborated and understood.
* Ensures that everyone in the Scrum Team understands what is required.
* Be available to answer any question arise during Sprint execution and support them.

**Sprint Planning and Execution**

* Attends Sprint Planning in order to answer any remaining questions.
* Attends Daily Stand-ups to remain engaged and up-to-date on the team’s progress.
* Completes incremental reviews of stories as they are completed.
* Adjust priorities of stories mid-sprint based on impediments or dependencies.
* Attend the Demo to provide feedback and acceptance of stories.
* Accept work results.

**Stakeholder Management**

* Collaborates with Stakeholders outside of the team to review progress.
* Collect and discuss required functionalities with the different Stakeholders. These requirements are then combined and filtered before giving it to the team in the form of prioritized Scrum Product Backlog Items.

**5 Qualities of product owner**

**1. Leadership**

As every team member is responsible for the product success, it is must for the product owner to provide guidance and direction to everyone involved in the development effort and ensures that project goes well.

**2. Decision making**

He or she should be good decision maker especially in deciding which product features will bring highest ROI.

**3. Communication**

He/she must be able to communicate different messages to different people about the project at any given time.

**4. Serve**

Adopt a “customer service” mentality, make yourself available in-person whenever possible, and be open to questions.

**5. Let Go**

Allowing the Scrum team to oversee their own task-execution will save time and make everyone’s life easier

**Scrum Development Team**

A Scrum Team is a collection of individuals working together to deliver the requested and committed product increments. It comprises of cross-functional members who are capable of achieving the sprint goals. This could include software engineers, architects, programmers, analysts, system admins, QA experts, testers, UI designers, etc. The most effective scrum teams are tight-knit, co-located, and usually 5 to 7 members. Team members have differing skill sets, and cross-train each other so no one person becomes a bottleneck in the delivery of work. Strong scrum teams approach their project with a clear "we" attitude. All members of the team help one another to ensure a successful sprint completion.

**Responsibilities of the Scrum Development Team**

**1. Core responsibilities**

* Embraces the (5) Scrum Values – Focus, Commitment, Openness, Respect and Courage.
* Delivers working product on-time, on scope and on quality.
* Shares development responsibilities, assists, trains and mentors Team members to meet “Sprint” challenges.
* Strives to value the individual and increase team recognition over self-recognition.
* Builds consensus and applies sound judgment in a Team centric environment.
* Provides open and honest feedback within a “Scrum Team” environment.
* Always is prepared to challenge self and the Scrum Team in (DoD) “Definition of Done.”

**2. Sprint creation and implementation**

* Select the tasks of highest need and complete them as quickly as possible.
* Request clarification from the product owner when you are unclear about a user story.
* Collaborate with other development team members to design the approach to a specific user story, seek help when you need it, and provide help when another development team member needs it.
* Conduct peer reviews on one another’s work.
* Take on tasks beyond your normal role as the sprint demands.
* Fully develop functionality as agreed to in the definition of done.
* Report daily on your progress.
* Alert the scrum master to any roadblocks you can’t effectively remove on your own.
* Achieve the sprint goal you committed to during sprint planning.

**6 Qualities of Scrum Development Team**

Key qualities in an ideal scrum team should be:

**1. Pair Programmer:** The programmer should be able to work effectively with another one at one workstation. While one programmer writes the code - the driver - the other one observes - the navigator - and reviews each line of code as it is typed in. The two programmers switch roles frequently.

**2. Understands TDD, BDD, etc.:**The team member should be familiar with advanced technique of using automated unit tests to drive the design of software and force decoupling of dependencies.

**3. Understands Refactoring:** Code Refactoring is the process of clarifying and simplifying the design of existing code, without changing its behavior. Scrum teams should maintain and extend their code a lot from iteration to iteration and support continuous refactoring.

**4. Continuous Integration:** Continuous Integration (CI) involves producing a clean build of the system several times per day, usually with a tool like Cruise Control, which uses Ant and various source-control systems. Scrum teams typically configure CI to include automated compilation, unit test execution, and source control integration.

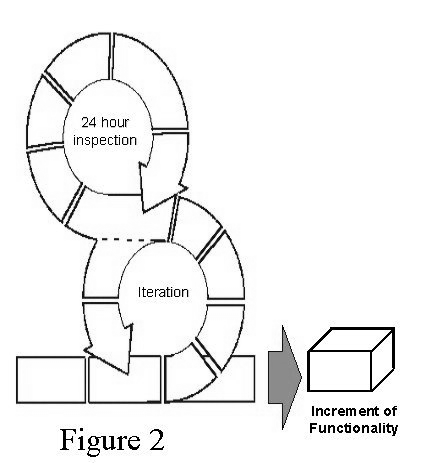
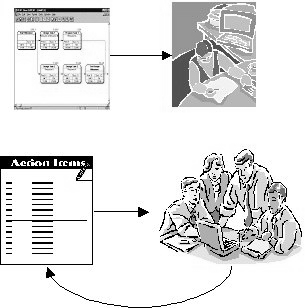
**5. Self-motivated and organized:** Scrum suggests self-organizing and self-managing teams. The Scrum team should remain accountable for the work it does and own the projection behalf of the client. Self-motivation is an important Scrum virtue since all Scrum roles are empowered to function on their own and there is no senior-junior hierarchy within the team. The entire team is accountable for itself.

**6. Team player:** Scrum supports team work and not just individual efforts. It’s important to function as a single cohesive unit and achieve the goals by working together. Each team member should be a team player.

**Processes Overview**

The result of all of this is that agile methods have some significant changes in emphasis from heavyweight methods. The most immediate difference is that they are less document-oriented, usually emphasizing a smaller amount of documentation for a given task. In many ways they are rather code-oriented, following a route that says that the key part of documentation is source code. However, I don’t think this is the key point about agile methods. Lack of documentation is a symptom of two much deeper differences, Agile methods are adaptive rather than predictive. Heavy methods tend to try to plan out a large part of the software process in great detail for a long span of time, this works well until things change. So, their nature is to resist change. The agile methods, however, welcome change. They try to be processes that adapt and thrive on change, even to the point of changing themselves. Agile methods are people-oriented rather than process-oriented. They explicitly make a point of trying to work with peoples’ nature rather than against them and to emphasize that software development should be an enjoyable activity.

Agile processes use empirical, adaptive processes for rendering emergent requirements into a working system. An agile process has a strong core. The framework on which the rest of the process works is part of this core. It supports the iterative, incremental nature of all agile processes. Its backbone is the Sprint. While Scrum uses 30-day Sprints, other agile processes use shorter or slightly longer iterations, but they rarely exceed 60 days. Iteration length is limited so that the team is forced to deliver a potentially shippable product increment with regularity. At the end of every Sprint, the increment is inspected, which dictates the project’s future progress. If the increment is appropriate and useful, the project progresses with only minor changes. If the increment is unsatisfactory, the cause is to determine, and adjustments made before the next Sprint begins. Anything unexpected can be detected and adjusted to at the end of a Sprint, based on the inspection of the increment. The lower circle represents a Sprint, with the output from each Sprint a potentially shippable product increment. The upper circle represents the daily inspection that occurs during the Sprint, where status is reported, and progress inspected and adapted to. This cycle repeats until the project is no longer funded. Agile processes try to have all analysis, design, coding and testing work done within a Sprint. Teams perform this work. The optimal team is a small group at the same site.



The framework operates at the start of a Sprint, the team reviews what it must do. Then, it selects what it believes it can turn into an increment of potentially shippable functionality by the end of the Sprint. The team is then left alone to make its best effort for the rest of the Sprint. At the end of the Sprint, the team presents the increment of functionality that it built. The heart of agile processes occurs within the Sprint. The team takes a look at the requirements, the technology, and evaluates each other’s skills and capabilities. It then devises the best way it knows to build the functionality, modifying the approach daily as it encounters new complexities, difficulties, and surprises. The team figures out what needs to be done and determines the best way to do it. This creative process is the heart of the extreme productivity that’s found in agile processes. In more traditional project management processes, management devises plan to which teams follow. The creative process is largely performed outside the team, as management tries to predict the best way for the team to build functionality. There is very little room left for team creativity and flexibility. The customer doesn’t value it. Although the development organization may know it’s required, the customer views it as overhead, this preliminary work is usually performed by specialized groups that aren’t responsible for delivering the increments of functionality to the customers and, it delays the delivery of valuable functionality and can allow the competition to catch up or gain an advantage. To avoid these problems, development teams do all this preliminary work in agile processes while they’re building functionality. If the development teams require experts to do the work, they’re included on the teams for as many iterations as it takes. However, this preliminary work is done in parallel with the development of working functionality that can be demonstrated to the customer. This functionality works from the user interface, through all intermediate levels to the persistent data stores and back. This “tracer bullet” demonstrates that the development environment and operational environment work, allowing the team and users to refine their aim in future iterations.

**Planning Phase:**

Scrummy also inherits the planning phase from Scrum, but it aims for shorter meetings due to the encouraged shorter sprint duration. In Scrum, the maximum planning time for a sprint is 8 hours, although most projects will not require a planning meeting to last that long. Scrummy aims to keep the duration of these meetings well under 8 hours. A massive benefit of shorter sprint duration is the fact that planning for shorter sprints tends to be more accurate, greatly reducing the chance of missing deadlines.

The planning phase has three goals:

1. Define the project
2. Build product backlog
3. Move items from product backlog to sprint backlog

Each of these goals, along with the ways they are accomplished, will be discussed in depth in the following sections.



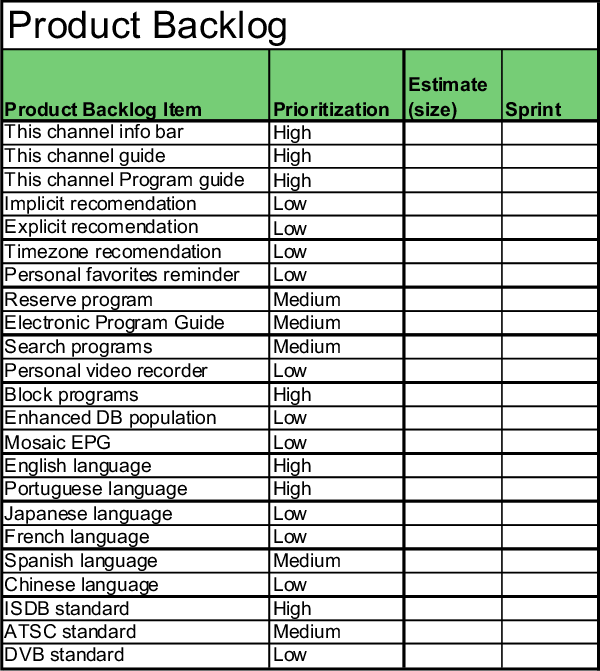
1. **Define the Project**

The first goal of the planning phase is to define the project. If it is the first cycle of a project, the overview of the project as well as the current known requirements are laid out and discussed so that all team members understand the customer's vision and realize what they are working towards. The vital functions and efficiency goals are mentioned and recorded so that the team can start to break the project down into pieces and gauge approximately how long the project should take to complete. Although the requirements are likely to change over the life of the project, it is helpful to have a ballpark figure of the project's timeline so that an estimated pace can be set and adhered to.

If it is not the first cycle of a project, then the current state of the project is assessed and the team discusses what they learned during the previous iteration to improve the efficiency and quality of the next iteration.

1. **Build and Manage the Product Backlog**

During this phase, the product owner discusses the product backlog with the rest of the team. If it is the first cycle of a project, the PO builds the product backlog using the requirements presented by the customer as well as other features laid out during the process of defining the project. The PO may also choose to assign priority levels to items on the list, highlighting the modules that would be most beneficial to implement next. After the product backlog is built, the PO continues to communicate with the customer to discover if any requirements have changed or if the customer realizes that extra functionalities are required. This feedback is then used to manage the product backlog, adding new items to it or refining existing ones.



1. **Build and Manage Sprint Backlog from the Product Backlog**

After the product backlog is built and refined for the iteration, the development team discusses which features from the product backlog will be developed and added into the program during the next cycle. The development team makes the decisions in this phase, as they choose not only what features to implement but also how many they wish to incorporate during the next development cycle. The team also chooses how long the next cycle will be, which can be anywhere from 1 to 4 weeks. However, cycles closer to 1 weeks are preferred over longer cycles, as the end of each cycle provides incredibly valuable data to the team regarding efficiency. This data shows crucial information, such as how long the team spent working on a specific module. This data can then be analyzed to compare how much time was used developing to how much time was allotted to the cycle. Using this information, the team is able to more accurately plan for the next iteration. As the number of cycles in a project increases, the amount of data the team gets increases as well, allowing for the planning of each phase to become more accurate.

Essentially, a sprint planning meeting results in the development team choosing what will get done during that sprint and how it will get done.

**Sprint meeting**

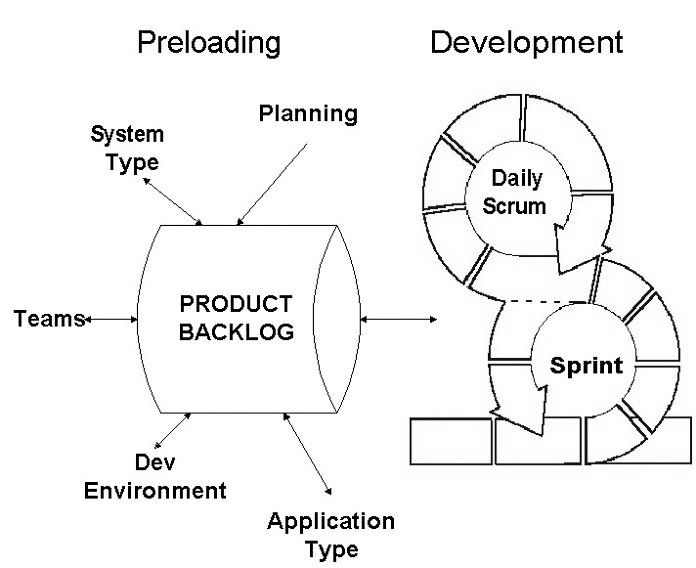
The product owner and team collaborate about the meaning and details of these backlog items until the team has enough grasp of the intentions and details to arrive at their own estimate of how long it will take to turn the item into product functionality. This estimate supersedes the previous estimate for each item. The reason for this is that the team now has a better grasp of the technology and business domain than it did at the start of the last Sprint. To the extent that the product owner has been working with the team, frequently refining the estimates, this variation is minimal. Otherwise these variations may be a surprise.

The collaboration between team and product owner includes any input that any other participant in the Sprint planning meeting provides. The Sprint planning meeting is the one completely open meeting in Scrum, where everyone has a say. To the extent that the product owner has worked with everyone who has a stake in the product or system prior to the meeting, this input has already been gathered. To the extent that the organization is at odds about what should be done next, the Sprint planning meeting may have difficulty concluding. This is appropriate, ensuring that development is not initiated until organizational agreement is reached on funding the next Sprint. Since this agreement is required to start a Sprint, Scrum provides a 30-day checkpoint for how to best spend an organization’s money. Having selected the Product Backlog, a Sprint Goal is crafted. The Sprint Goal is the purpose of the selected functionality; that is, what should the selected backlog be able to demonstrate once it is built? The Sprint Goal is an objective that will be met through the implementation of the Product Backlog.

The Sprint goal is to provide a standardized middleware mechanism for the identified customer service transactions to access backend databases. The Sprint Goal gives the team some wiggle room regarding the functionality. For example, the goal for the above Sprint could also be: “Automate the client account modification functionality through a secure, recoverable transaction middleware capability.” As the team works, it keeps this goal in mind. In order to satisfy the goal, it implements the functionality and technology. If the work turns out to be harder than the team had expected, then the team might only partially implement the functionality. At the Sprint Review meeting, management, customers, and the Product Owner review how and to what degree the functionality has been implemented. They review how the Sprint Goal has been met. If they are dissatisfied, they can then make decisions about requirements, technology or team composition. During the Sprint, though, the team alone determines how to meet the Sprint Goal. At the end of the Sprint, any incomplete work returns to the Product Backlog. The team then devises the individual tasks that must be performed to build the product increment.

**Developing**

All actual product development is done in the Developing Phase. This phase consists of multiple Sprints to develop increments of product functionality. Each Sprint starts with a Sprint Planning Meeting and concludes with a Sprint Review.



**Conduct Review**

The Sprint Review meeting is a four-hour informational meeting. During this meeting, the team presents to management, customers, users, and the Product Owner the product increment that it has built during the Sprint. The ScrumMaster is responsible for scheduling, setting up, and ensuring that all of the agenda items are covered. The ScrumMaster is responsible for coordinating and conducting the Sprint Review meeting. The ScrumMaster meets with the team to establish the agenda and discuss how the Sprint results will be presented and by whom. The ScrumMaster sends all attendees a reminder a week before the meeting, confirming the time, date, location, attendees, and agenda.

**AGENDA:**

1. Scrum Master provides review of Sprint goals, functionality chose, and functionality developed and to be demonstrated.
2. Team provides overview of the Sprint and functionality.
3. Team demonstrates the functionality on various workstations. Sometimes, if the audience is large, various team members demonstrate different pieces of functionality simultaneously until everyone has seen everything.
4. Meeting reconvenes and Scrum Master facilitates a discussion of impressions and observations on what was just demonstrated.
5. Scrum Master facilitates Sprint retrospective, a discussion of what went right and what went wrong in the Sprint, and what can be done to improve the next Sprint.
6. Scrum Master facilitates a discussion of the impact and implications of the demonstrated functionality of the project schedule and product backlog.
7. Scrum Master announces time and place of the Sprint Planning meeting that will initiate the next Sprint.

**End of Sprint Review**

The end of Sprint review provides an inspection of project progress at the end of a Sprint, every 30 calendar days. Based on the inspection, adaptations can be made to the project. The team has estimated where it will be at the end of the Sprint and set its course accordingly. At the end of the Sprint, the team presents the product increment that it has been able to build. Management, customers, users, and the Product Owner assess the product increment. They listen to the tales the team has to tell about its journey during the Sprint. They hear what went wrong and what went right. They take a fix on where they really are on their voyage of building the product and system. After all of this, they are able to make an informed decision about what to do next. In other words, they determine the best course to take in order reach their intended destination. every thirty days, and the team builds product during the other twenty- nine days.

Different groups of people attend the Sprint Review. Management comes to the Sprint Review to see what the team has been able to build with the resources that it has provided. Customers come to the Sprint Review to see if they like what the team has built. The Product Owner comes to the Sprint Review to see how much functionality has been built. Other engineers and developers come to the meeting to see what the team was able to do with the technology. Everyone wants to see what the team has built, what the Sprint was like, how the technology worked, what short- cuts had to be taken, what things it was able to add, and its ideas there are as to what can be done next. During the meeting, everyone visualizes the demonstrated product functionality working in the customer or user environment. As this is visualized, consider what functionality might be added in the next Sprint. The product increment is the focal point for brainstorming no one should prepare extensively for the meeting. In order to enforce this rule, PowerPoint presentations are forbidden. If the team feels that it has to spend more than two hours preparing for the meeting, then it is usually has less to show for the Sprint than it had hoped, and it is trying to obscure this fact with a fancy presentation. Sprint Review Meetings are very informal. At these meetings, what matters is the product the team has been able to create. The Sprint Review is a working meeting. Questions, observations, discussions and suggestions are allowed, and even encouraged. If a lot of give and take is needed, it should happen. Remember, though, that the meeting is informational, not critical or action-oriented. Everyone should get an understanding of the product increment, as this is the knowledge that they will need for the Sprint Planning meeting.



**Sprint Retrospective**

The Scrum process works out of the box. However, as organizations understand it, they often want to make modifications to make it suit their environment better. Also, when the project was formulated, staffing, tools, and environments were selected. As the project progresses, these also may require adjusting. After every Sprint, everyone on the project takes time to reflect. What went well in the Sprint? What went not so well? What could be improved? What did we do that we want to continue doing? When the project is started, the inputs are the Scrum process, the ScrumMaster, the Product Owner, the business domain experts, the teams, the development and QA environments, the engineering practices and standards, and the technologies. The basis of Scrum is to inspect and adapt. This Sprint Retrospective is the time to inspect all of the above items, and anything else that affects the project, and to make adjustments so that the team can do better.

**Security Requirements**

**Why do you need software security requirements?**

Traditionally, requirements are about defining what something can do or be. A hammer has to be capable of driving nails. A door lock needs to keep a door closed until it’s unlocked with a specific key. A car needs to move travelers from point A to point B along the nation’s roads. It also needs to work with the modern gasoline formulation. These types of requirements work fine for physical objects but fall short when designing software.

Additionally, these objects can be used for more than just their intended purpose, and their purposes can be circumvented to suit the user. For instance, a hammer can be used to break a window, a door lock can be picked, and a car can be used to transport stolen goods. Similarly, software can be abused or made vulnerable. The key difference is that GM isn’t liable when their cars are used as getaway vehicles. However, when your software’s capabilities and permissions are hacked, you (as the software owner) are the one who suffers.

Security vulnerabilities allow software to be abused in ways that the developers never intended. Imagine being able to design a hammer that can only hammer nails and nothing else. By building robust software security requirements, you can lock down what your software does so that it can *only* be used as intended.

Fortunately, building software that is immune to the [OWASP Top 10](https://www.synopsys.com/software-integrity/resources/knowledge-database/owasp-top-10.html) is easier than building a hammer that turns to marshmallows when used to hit anything but nails.

**What is a security requirement?**

A security requirement is a goal set out for an application at its inception. Every application fits a need or a requirement. Some applications allow customers to perform actions without needing help from a company representative. Just as those actions and outcomes are laid out as goals for the final application, the security goals must also be included. A security requirement is not a magic wand that you can wave at an application and say “Thou shalt not be compromised by hackers” any more than a New Year’s resolution is a magic wand that you can wave at yourself to lose weight. Just like a resolution to lose weight, being vague is a recipe for failure. How much weight? How will you lose it? Will you exercise, diet, or both? What milestones will you put out there? In security, the same types of questions exist. What kinds of vulnerabilities are you looking to prevent? How will you measure whether your requirement is met? What preventative measures will you take to ensure that vulnerabilities aren’t built into the code itself?

When building a security requirement, be specific about the kind of vulnerabilities to prevent. Take this requirement example: “[Application X] shall not execute commands embedded in data provided by users that forces the application to manipulate the database tables in unintended ways.” This is a fancy way of saying that the application should not be vulnerable to [SQL injection attacks](https://www.synopsys.com/blogs/software-security/ramp-up-your-web-application-security/). This can be tested with specific kinds of tests, both on the source code itself and on the compiled application. These attacks are preventable with a combination of rejecting or scrubbing bad input from the user, using a carefully crafted type of database query that flags data as data and not as commands to be acted upon, and modifying the output of the database calls to prevent bad data from attacking functionality down the line.

**Requirements for your requirements.**

In order to build good requirements, you should make sure that you are answering questions about your requirements. A security requirement should be built much like a functionality requirement; it shouldn’t be vague or unattainable. Anticipate the questions that the developers will have an answer them ahead of time. Here’s how:

1. **Is this testable?**Can this requirement be tested in the final application? “Being secure” is not testable; however, encoding all user supplied output is.
2. **Is this measurable?**When we test for this, can we determine coverage and effectiveness?
3. **Is this complete?**Are we forgetting something in this? Are we mandating checks for user supplied data to databases but not logs?
4. **Is this clear?**Will the people responsible for designing, implementing, testing, and delivering on this requirement understand the intent of the requirement?
5. **Is this unambiguous?**Are there any other ways that this requirement can be interpreted?
6. **Are these requirements consistent?**Are we approaching each security requirement in the same way to ensure that the security measures are applied consistently across the board?

When building a requirement, remember that it is a goal that someone must achieve. By creating specific and achievable requirements, the designers and developers can meet the security goals for an application.

**Types of security requirements.**

If you are entrenched in the requirements or contracting world, you are already aware of the basic kinds of requirements: functional, non-functional, and derived. Security requirements fall into the same categories, but just like performance requirements define what a system has to do and has to be in order to perform according to specifications, security requirements define what a system has to do and be in order to perform securely.

When defining functional non-security requirements, you see things like “If the scan button is pressed, the lasers shall activate and scan for a barcode.” This is what a barcode scanner needs to do. When a security requirement is written, it talks about the things that a system has to do to enforce security like “The cashier must log in with a magnetic stripe card and pin number before the cash register is ready to process sales.”

A functional security requirement is something that describes functional behavior that enforces security. It can be directly tested and observed. Requirements that have things to do with access control, data integrity, authentication, and wrong password lockouts fall under functional requirements.

Non-functional requirements describe what a system has to be. These are statements that support auditability and uptime. Non-functional security requirements are statements like “Audit logs shall be verbose enough to support forensics.” Supporting auditability is not a direct functionality requirement, but it supports auditability requirements from regulations that may apply.

Derived requirements are inspired by the functional and non-functional requirements. When a system has a user ID and PIN functional requirement, a derived requirement may define the number of PIN guesses before an account is locked out. For audit logs, a derived requirement may support the integrity of the logs, such as log injection prevention.

Derived requirements are tricky because these stem from [abuse cases](https://www.synopsys.com/blogs/software-security/abuse-cases-can-drive-security-requirements/). Not only must the requirements designer think like a user and a customer, but they also have to think like an attacker. For every bit of functionality that is given to the user, that functionality could be abused by an attacker. Login functionality can become password guessing attempts, uploading files can open a system up to hosting malware, and accepting text can open the door to [cross-site scripting](https://www.synopsys.com/software-integrity/resources/knowledge-database/cross-site-scripting.html) or [SQL injection](https://www.synopsys.com/software-integrity/resources/knowledge-database/sql-injection.html).

**Making requirements.**

Security requirements can come from many sources along the requirements and early design phases. When defining functionality, that functionality must be defined securely or have supporting requirements to ensure that the business logic is secure. Generic guidance from industry best practices, regulatory requirements, and guidance must be tailored to meet specific application requirements.

Abuse cases are a way to think like an attacker. A use case is flipped on its head and designers analyze how the functionality can be abused. If a user is allowed to generate reports with sensitive data, how might an unauthorized user gain access to those reports and their sensitive data? Those abuse cases are often answered by industry best practices which can be used to build requirements for how the application handles access to privileged data.

Security requirements can also come from analysis of the design via [architecture risk analysis](https://www.synopsys.com/software-integrity/software-security-services/software-architecture-design/risk-analysis.html). If a web application uses a specific framework or language, industry knowledge of attack patterns and vulnerabilities can be applied. If a framework prevents cross-site scripting in some situations and not others, a requirement which speaks to how the developers will prevent cross-site scripting in insecure situations needs to be defined.

Every security requirement should address a specific security need and knowledge of the vulnerabilities that could exist in an application is essential. Generic guidance and knowledge are not enough. Specific security requirements will arise from the specific application requirements.

**What can requirements do for you?**

It doesn’t matter if you build software in-house or if you outsource your software to third-party vendors; building sound security requirements can benefit you. By defining your security requirements early, you can spare yourself from nasty surprises later. Sound security requirements will help internally as they provide a clear road map for developers. They can also help with external regulatory requirements. Implementing measures to keep software from getting hacked is a good strategy and security requirements are a fantastic start to not being unhappy with what you get. Build your software security requirements early and sit in the shade of securely built software later.

**Quality Assurance**

**What is Quality?**

Quality is extremely hard to define, and it is simply stated: "Fit for use or purpose." It is all about meeting the needs and expectations of customers with respect to functionality, design, reliability, durability, & price of the product.

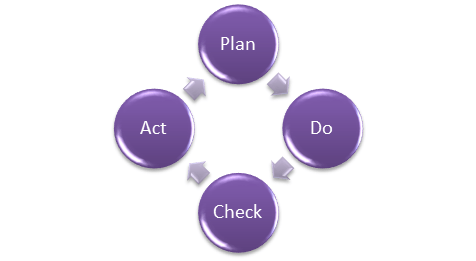
**What is Assurance?**

Assurance is nothing but a positive declaration on a product or service, which gives confidence. It is certainty of a product or a service, which it will work well. It provides a guarantee that the product will work without any problems as per the expectations or requirements.

**What is Quality Assurance?**

Quality Assurance is popularly known as QA Testing, is defined as an activity to ensure that an organization is providing the best possible product or service to customers. QA focuses on improving the processes to deliver Quality Products to the customer. An organization has to ensure, that processes are efficient and effective as per the quality standards defined for software products.

**How to do Quality Assurance: Complete Process**

[](https://www.guru99.com/images/Q2.png)Quality assurance has a defined cycle called PDCA cycle or Deming cycle. The phases of this cycle are:

* Plan
* Do
* Check
* Act

These above steps are repeated to ensure that processes followed in the organization are evaluated and improved on a periodic basis. Let's look into the above steps in detail -

* **Plan -** Organization should plan and establish the process related objectives and determine the processes that are required to deliver a high-Quality end product.
* **Do -** Development and testing of Processes and also "do" changes in the processes
* Check - Monitoring of processes, modify the processes, and check whether it meets the predetermined objectives
* **Act -** Implement actions that are necessary to achieve improvements in the processes

An organization must use Quality Assurance to ensure that the product is designed and implemented with correct procedures. This helps reduce problems and errors, in the final product.

**What is Quality Control?**

[](https://www.guru99.com/images/Q3.png)

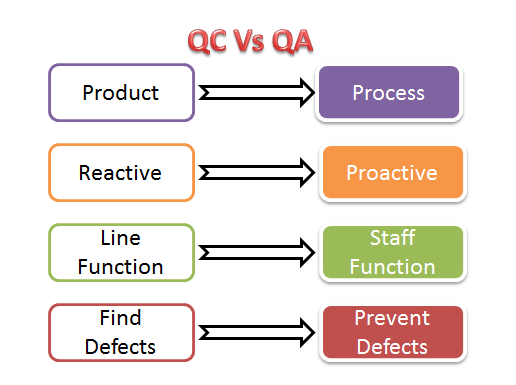
Quality control popularly abbreviated as QC. It is a Software Engineering process used to ensure quality in a product or a service. It does not deal with the processes used to create a product; rather it examines the quality of the "end products" and the final outcome.

The main aim of Quality control is to check whether the products meet the specifications and requirements of the customer. If an issue or problem is identified, it needs to be fixed before delivery to the customer.

QC also evaluates people on their quality level skill sets and imparts training and certifications. This evaluation is required for the service-based organization and helps provide "perfect" service to the customers.

**Difference between Quality Control and Quality Assurance?**

Sometimes, QC is confused with the QA. Quality control is to examine the product or service and check for the result. Quality assurance is to examine the processes and make changes to the processes which led to the end-product.

[](https://www.guru99.com/images/Q4.png)

**Examples of QC and QA activities are as follows:**

|  |  |
| --- | --- |
| **Quality Control Activities** | **Quality Assurance Activities** |
| Walkthrough | Quality Audit |
| Testing | Defining Process |
| Inspection | Tool Identification and selection |
| Checkpoint review | Training of Quality Standards and Processes |

**The above activities are concerned with QA and QC of any product and not essentially software.**

With respect to software

* QA becomes SQA (Software Quality Assurance)
* QC becomes Software Testing**.**

**Differences between SQA and Software Testing**

Following table explains on differences between SQA and Software Testing:

|  |  |
| --- | --- |
| **SQA** | **Software Testing** |
| Software Quality Assurance is about engineering process that ensures quality | Software Testing is to test a product for problems before the product goes live |
| Involves activities related to the implementation of processes, procedures, and standards. Example - Audits Training | Involves actives concerning verification of product Example - Review Testing |
| Process focused | Product focused |
| Preventive technique | Corrective technique |
| Proactive measure | Reactive measure |
| The scope of SQA applied to all products that will be created by the organization | The scope of Software Testing applies to a particular product being tested. |

**Best practices for Quality Assurance:**

* Create a Robust Testing Environment
* Select release criteria carefully
* Apply automated testing to high-risk areas to save money. It helps to fasten the entire process.
* Allocate Time Appropriately for each process
* It is important to prioritize bugs fixes based on software usage
* Form dedicated security and performance testing team
* Simulate customer accounts similar to a production environment

**Quality Assurance Functions:**

There are 5 primary Quality Assurance Functions:

1. **Technology transfer:** This function involves getting a product design document as well as trial and error data and its evaluation. The documents are distributed, checked and approved
2. **Validation:** Here validation master plan for the entire system is prepared. Approval of test criteria for validating product and process is set. Resource planning for execution of a validation plan is done.
3. **Documentation:** This function controls the distribution and archiving of documents. Any change in a document is made by adopting the proper change control procedure. Approval of all types of documents.
4. **Assuring Quality of products**
5. **Quality improvement plans**

**CMMI level**

The **Capability Maturity Model Integrated (CMMI)** is a process improvement approach developed specially for software process improvement. It is based on the process maturity framework and used as a general aid in business processes in the Software Industry. This model is highly regarded and widely used in Software Development Organizations.

CMMI has 5 levels. An organization is certified at CMMI level 1 to 5 based on the maturity of their Quality Assurance Mechanisms.

Level 1 - **Initial:** In this stage the quality environment is unstable. Simply, no processes have been followed or documented

Level 2 - **Repeatable:**Some processes are followed which are repeatable. This level ensures processes are followed at the project level.

Level 3 - **Defined:**Set of processes are defined and documented at the organizational level. Those defined processes are subject to some degree of improvement.

Level 4 - **Managed:** This level uses process metrics and effectively controls the processes that are followed.

Level 5 - **Optimizing:** This level focuses on the continuous improvements of the processes through learning & innovation.

**Test Maturity Model (TMM):**

This model assesses the maturity of processes in a Testing Environment. Even this model has 5 levels, defined below-

Level 1 - **Initial**: There is no quality standard followed for testing processes and only ad-hoc methods are used at this level

Level 2 - **Definition:** Defined process. Preparation of test strategy, plans, test cases are done.

Level 3 - **Integration:** Testing is carried out throughout the software development lifecycle (SDLC) - which is nothing but integration with the development activities, E.g., V- Model.

Level 4 -**Management and Measurement:** Review of requirements and designs takes place at this level and criteria has been set for each level of testing

Level 5 - **Optimization:** Many preventive techniques are used for testing processes, and tool support (Automation) is used to improve the testing standards and processes.

Quality Assurance is to check whether the product developed is fit for use. For that, Organization should have processes and standards to be followed which need to be improved on a periodic basis. It concentrates mainly on the quality of product/service that we are providing to the customers during or after implementation of software.

**Software Testing**

**What is software testing?**

Software testing is an organizational process within software development in which business-critical software is verified for correctness, quality, and performance. Software testing is used to ensure that expected business systems and product features behave correctly as expected.

Software testing may either be a manual or an automated process.

* **Manual software testing** is led by a team or individual who will manually operate a software product and ensure it behaves as expected.
* **Automated software testing** is composed of many different tools which have varying capabilities, ranging from isolated code correctness checks to simulating a full human-driven manual testing experience.

There are many different types of testing that you can use to make sure that changes to your code are working as expected. Not all testing is equal, though, and we will see here how the main testing practices differ from each other.

**Manual vs Automated testing**

At a high level, we need to make the distinction between manual and automated tests. Manual testing is done in person, by clicking through the application or interacting with the software and APIs with the appropriate tooling. This is very expensive as it requires someone to set up an environment and execute the tests themselves, and it can be prone to human error as the tester might make typos or omit steps in the test script.

Automated tests, on the other hand, are performed by a machine that executes a test script that has been written in advance. These tests can vary a lot in complexity, from checking a single method in a class to making sure that performing a sequence of complex actions in the UI leads to the same results. It's much more robust and reliable than automated tests – but the quality of your automated tests depends on how well your test scripts have been written.

Automated testing is a key component of [continuous integration](https://www.atlassian.com/continuous-delivery/continuous-integration/how-to-get-to-continuous-integration) and [continuous delivery](https://www.atlassian.com/continuous-delivery/pipeline) and it's a great way to scale your QA process as you add new features to your application. But there's still value in doing some manual testing with what is called exploratory testing as we will see in this guide.

**The different types of tests**

**Unit tests**

Unit tests are very low level, close to the source of your application. They consist in testing individual methods and functions of the classes, components or modules used by your software. Unit tests are in general quite cheap to automate and can be run very quickly by a continuous integration server.

**Integration tests**

Integration tests verify that different modules or services used by your application work well together. For example, it can be testing the interaction with the database or making sure that microservices work together as expected. These types of tests are more expensive to run as they require multiple parts of the application to be up and running.

**Functional tests**

Functional tests focus on the business requirements of an application. They only verify the output of an action and do not check the intermediate states of the system when performing that action.

There is sometimes a confusion between integration tests and functional tests as they both require multiple components to interact with each other. The difference is that an integration test may simply verify that you can query the database while a functional test would expect to get a specific value from the database as defined by the product requirements.

**End-to-end tests**

End-to-end testing replicates a user behavior with the software in a complete application environment. It verifies that various user flows work as expected and can be as simple as loading a web page or logging in or much more complex scenarios verifying email notifications, online payments, etc...

End-to-end tests are very useful, but they're expensive to perform and can be hard to maintain when they're automated. It is recommended to have a few key end-to-end tests and rely more on lower level types of testing (unit and integration tests) to be able to quickly identify breaking changes.

**Acceptance testing**

Acceptance tests are formal tests executed to verify if a system satisfies its business requirements. They require the entire application to be up and running and focus on replicating user behaviors. But they can also go further and measure the performance of the system and reject changes if certain goals are not met.

**Performance testing**

Performance tests check the behaviors of the system when it is under significant load. These tests are non-functional and can have the various form to understand the reliability, stability, and availability of the platform. For instance, it can be observing response times when executing a high number of requests or seeing how the system behaves with a significant of data.

Performance tests are by their nature quite costly to implement and run, but they can help you understand if new changes are going to degrade your system.

**Smoke testing**

Smoke tests are basic tests that check basic functionality of the application. They are meant to be quick to execute, and their goal is to give you the assurance that the major features of your system are working as expected.

Smoke tests can be useful right after a new build is made to decide whether or not you can run more expensive tests, or right after a deployment to make sure that they application is running properly in the newly deployed environment.

**How to automate your tests**

An individual can execute all the tests mentioned above, but it will be very expensive and counter-productive to do so. As humans, we have limited capacity to perform a large number of actions in a repeatable and reliable way. But a machine can easily do that rapidly and will test that login/password combination works for the 100th time without complaining.

To automate your tests, you will first need to write them programmatically using a testing framework that suits your application. [PHPUnit](https://phpunit.de/), [Mocha](https://mochajs.org/), [RSpec](http://rspec.info/) are examples of testing frameworks that you can use for PHP, JavaScript, and Ruby respectively. There are [many options](https://en.wikipedia.org/wiki/List_of_unit_testing_frameworks) out there for each language so you might have to do some research and ask developer communities to find out what would be the best framework for you.

When your tests can be executed via script from your terminal, you can have them be automatically executed by a continuous integration server like Bamboo or use a cloud service like Bitbucket Pipelines. These tools will monitor your repositories and execute your test suite whenever new changes are pushed to the main repository.

**Exploratory testing**

The more features and improvements go into your code, the more you'll need to test to make sure that all your system works properly. And then for each bug you fix, it would be wise to check that they don't get back in newer releases. Automation is key to make this possible and writing tests sooner or later will become part of your development workflow.

So, the question is whether it is still worth doing manual testing? The short answer is yes, and it should be focused on what is called exploratory testing where the goal is to uncover non-obvious errors.

An exploratory testing session should not exceed two hours and need to have a clear scope to help testers focus on a specific area of the software. Once all testers have been briefed, is up to them to try various actions to check how the system behaves. This type of testing is expensive by nature but is quite helpful to uncover UI issues or verify complex user workflows. It's something especially worth doing whenever a significant new capability is added to your application to help understand how it behaves under edge cases.

**A note about testing**

It's important to talk about the goal of testing. While it's important to test that users can use your application (I can log in, I can save an object) it is equally important to test that your system doesn't break when bad data or unexpected actions are performed. You need to anticipate what would happen when a user makes a typo, tries to save an incomplete form or uses the wrong API. You need to check if someone can easily compromise data, get access to a resource they're not supposed to. A good testing suite should try to break your app and help understand its limit.

And finally, tests are code too! So, don't forget them during code review as they might be the final gate to production.

**Improving and adapting the framework:**

Just as requirements and scope change from project to project, this framework and the usage of it should also change from project to project. This is accomplished through the immense amount of data gathered by the team regarding efficiency. When an organization continues to employ and utilize this framework, the framework continues to become more efficient as the organization uses gathered data to improve it and adapt it to their usage. After each project is completed, the team learns more about numerous factors, including the efficiency of the time spent developing as well as the number of bugs that are produced during development but caught by testing teams. If a team allots either too much or too little time for a module, the data from the cycle will help them learn to plan time more efficiently in future cycles and projects. If a certain iteration produces a considerable number of bugs, then the developers can review the bug documentation and avoid utilizing the same techniques to reduce the likelihood of situations in which the testers need more time. This framework is easy to adopt but hard to master, meaning that organizations and their teams have ample room to grow alongside the framework as they use it. The more the framework is used, the more rewarding it will become.