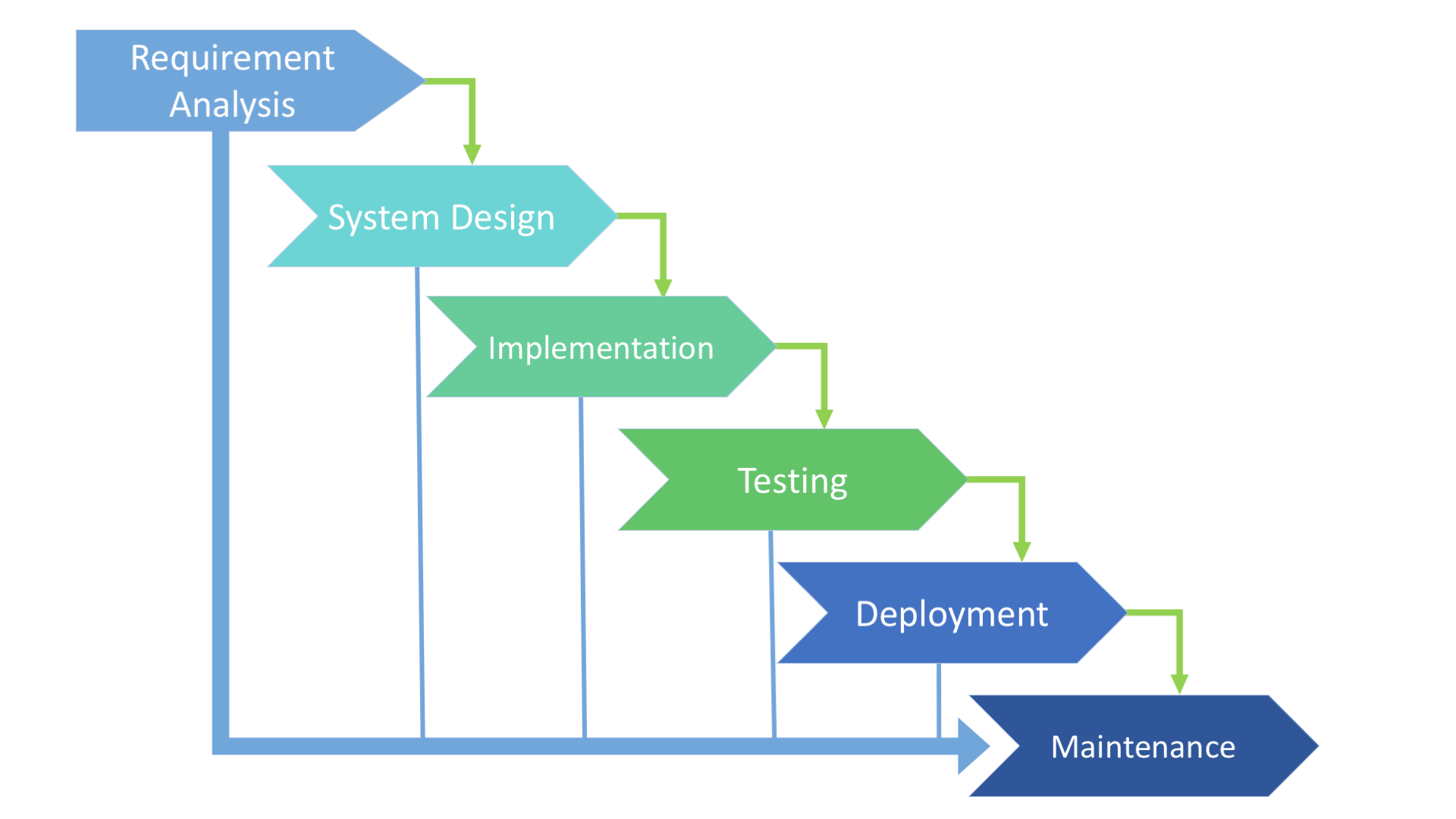
**Project Type**

**Sequential development models**

**Waterfall SDLC**

Waterfall – is a cascade SDLC model, in which the development process looks like the flow, moving step by step through the phases of analysis, projecting, realization, testing, implementation, and support. This SDLC model includes gradual execution of every stage completely. This process is strictly documented and predefined with features expected to every phase of this software development life cycle model.



*ADVANTAGES:*

Simple to use and understand

Management simplicity thanks to its rigidity: every phase has a defined result and process review

Development stages go one by one

Perfect for the small or mid-sized projects where requirements are clear and not equivocal

Easy to determine the key points in the development cycle

Easy to classify and prioritize tasks

*DISADVANTAGES:*

The software is ready only after the last stage is over

High risks and uncertainty

Not the best choice for complex and object-oriented projects

Inappropriate for the long-term projects

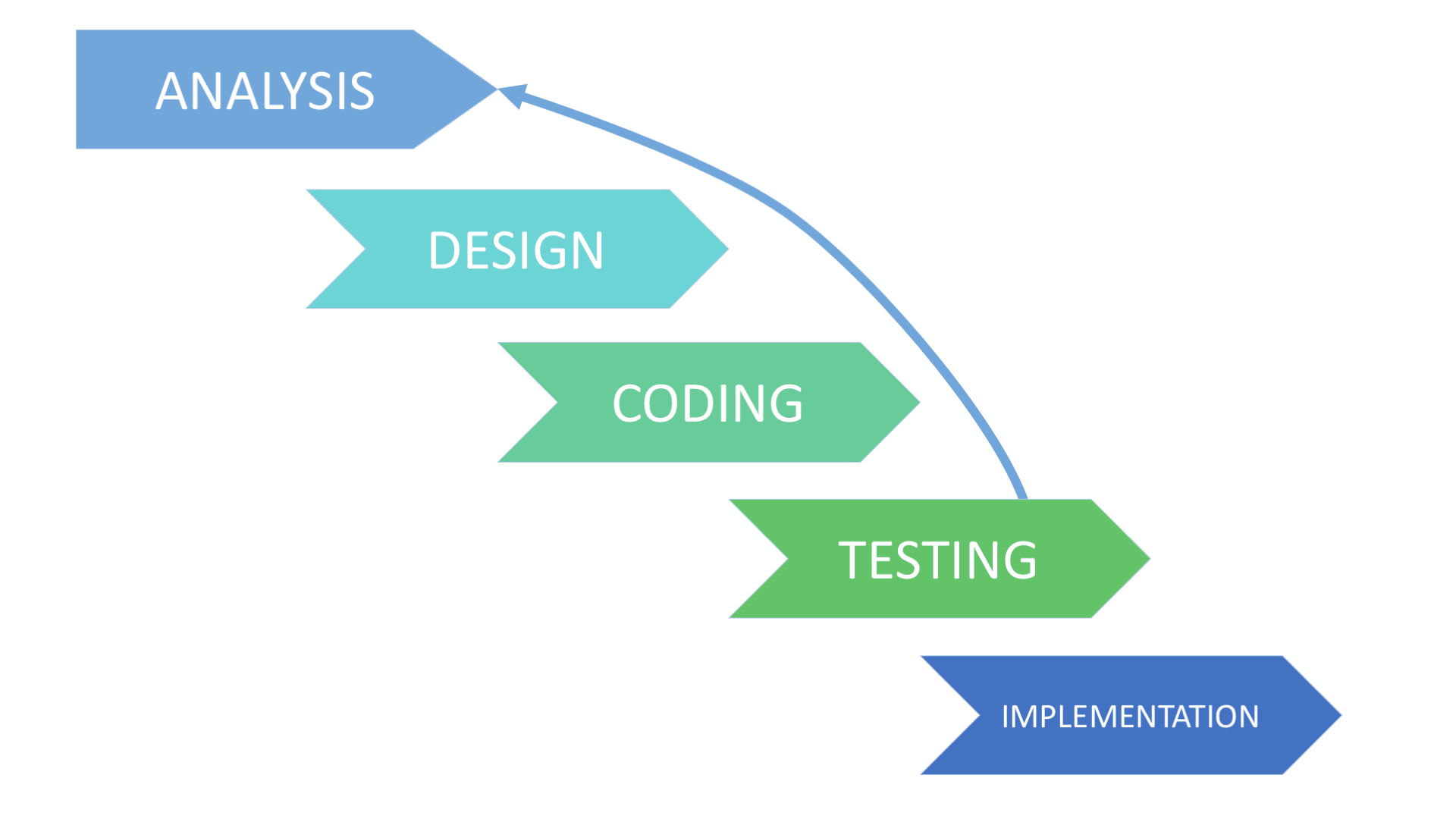
The progress of the stage is hard to measure while it is still in the development

Integration is done at the very end, which does not give the option of identifying the problem in advance

**Iterative SDLC model**

Iterative development is a type of software development process that makes progress through successive refinement through Product Increments. The Development Team creates the product although knowing that many parts are not completed. The team works on the parts that are not completed and enhances the product till the product is finished and satisfactory. **After each iteration, the customer’s feedback is taken into consideration and the software is improved as they add more details to the product. Taking an example of building a search function for a website, at first, the website would contain basic search criteria which could be later enhanced by adding more options in the next iteration.** In iterative processes, the developer first makes a minimally viable product, and later uses customer feedback to enhance the product, and continues the process through multiple iterations till the customer is satisfied.

The iterative development process is often used synonymously with Agile development. However, one has to know that even traditional project methodologies can utilize iterative development without using any other components of Agile. When traditional project methodologies use the iterative development method, they take the sequential approach to process further product development. The requirements of the project are determined when the project is started and are constantly developed and tested by the team. Once the development process is completed, and testing is completed, they are repeated based on the feedback of the customers throughout the project till the customer is satisfied. There are gaps given for the team members to rework and improve, however, the customers cannot add the items of requirements, only the development and testing is done repeatedly without overlapping. The timeframes are not specifically considered in an iterative process as it can last up to days, weeks, or even months; there is no real limit. The project is completed only when all the iterations are completed. In iterative development, customers are involved in the initial phase of the product, but they don’t follow up during the project. The requirements are established at the beginning, and the product is provided once for all after its completion.





*Advantages:*

Some functions can be quickly developed at the beginning of the development lifecycle;

The paralleled development can be applied

The progress is easy measurable

The shorter iteration is - the easier testing and debugging stages are

It is easier to control the risks as high-risk tasks are completed first

Problems and risks defined within one iteration can be prevented in the next sprints

Flexibility and readiness to the changes in the requirements

*Disadvantages:*

Iterative model requires more resources than the waterfall model

Constant management is required

Issues with architecture or design may occur because not all the requirements are foreseen during the short planning stage

Bad choice for the small projects

The process is difficult to manage

The risks may not be completely determined even at the final stage of the project

Risks analysis requires involvement of the highly-qualified specialists

# Incremental Model

Incremental Model is a process of software development where requirements are divided into multiple standalone modules of the software development cycle. In this model, each module goes through the requirements, design, implementation and testing phases. Every subsequent release of the module adds function to the previous release. In the incremental model the whole requirement is divided into various builds. Multiple development cycles take place here, making the life cycle a [**“multi-waterfall” cycle**](http://tryqa.com/what-is-waterfall-model-advantages-disadvantages-and-when-to-use-it/).

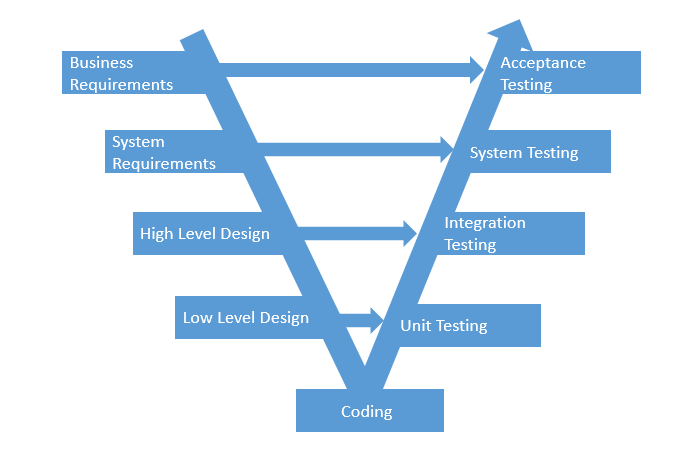




**V-shaped SDLC Model**

The V-model is an SDLC model where execution of processes happens in a sequential manner in a V-shape. It is also known as Verification and Validation model.

The V-Model is an extension of the waterfall model and is based on the association of a testing phase for each corresponding development stage. This means that for every single phase in the development cycle, there is a directly associated testing phase. This is a highly-disciplined model and the next phase starts only after completion of the previous phase.



Advantages:

Every stage of V-shaped model has strict results so it’s easy to control

Testing and verification take place in the early stages

Good for the small projects, where requirements are static and clear

Disadvantages:

Lack of the flexibility

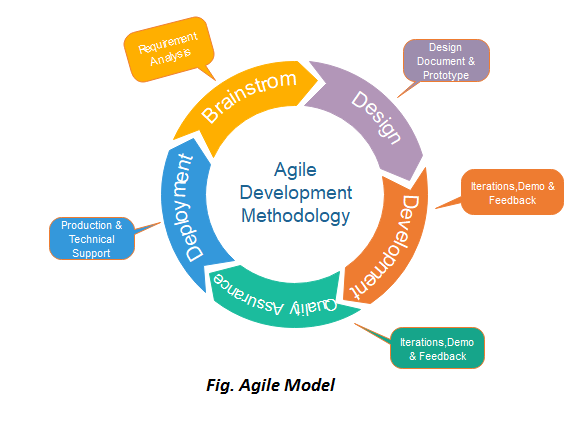
Bad choice for big projects

Relatively big risks

**Agile SDLC Model**

Agile methodology is a software development model that encourages the continuous iteration of development and testing in the entire software development lifecycle of the project.

In the agile methodology after every development iteration, the customer is able to see the result and understand if he is satisfied with it or he is not. This is one of the advantages of the agile software development life cycle model. One of its disadvantages is that with the absence of defined requirements it is difficult to estimate the resources and development cost. Extreme programming is one of the practical use of the agile model. The basis of such model consists of short weekly meetings – Sprints which are the part of the Scrum approach.



*Phases of Agile Model:*

Following are the phases in the Agile model are as follows:

Requirements gathering

Design the requirements

Construction/ iteration

Testing/ Quality assurance

Deployment

Feedback

1. *Requirements gathering* - In this phase, you must define the requirements. You should explain business opportunities and plan the time and effort needed to build the project. Based on this information, you can evaluate technical and economic feasibility.

2. *Design the requirements* - When you have identified the project, work with stakeholders to define requirements. You can use the user flow diagram or the high-level UML diagram to show the work of new features and show how it will apply to your existing system.

3. *Construction/ iteration* - When the team defines the requirements, the work begins. Designers and developers start working on their project, which aims to deploy a working product. The product will undergo various stages of improvement, so it includes simple, minimal functionality.

4. *Testing* - In this phase, the Quality Assurance team examines the product's performance and looks for the bug.

5. *Deployment* - In this phase, the team issues a product for the user's work environment.

6. Feedback - After releasing the product, the last step is feedback. In this, the team receives feedback about the product and works through the feedback.

*Advantages:*

Corrections of functional requirements are implemented into the development process to provide the competitiveness

Project is divided by short and transparent iterations

Risks are minimized thanks to the flexible change process

Fast release of the first product version

*Disadvantages:*

Difficulties with measuring the final cost because of permanent changes

The team should be highly professional and client-oriented

New requirements may conflict with the existing architecture

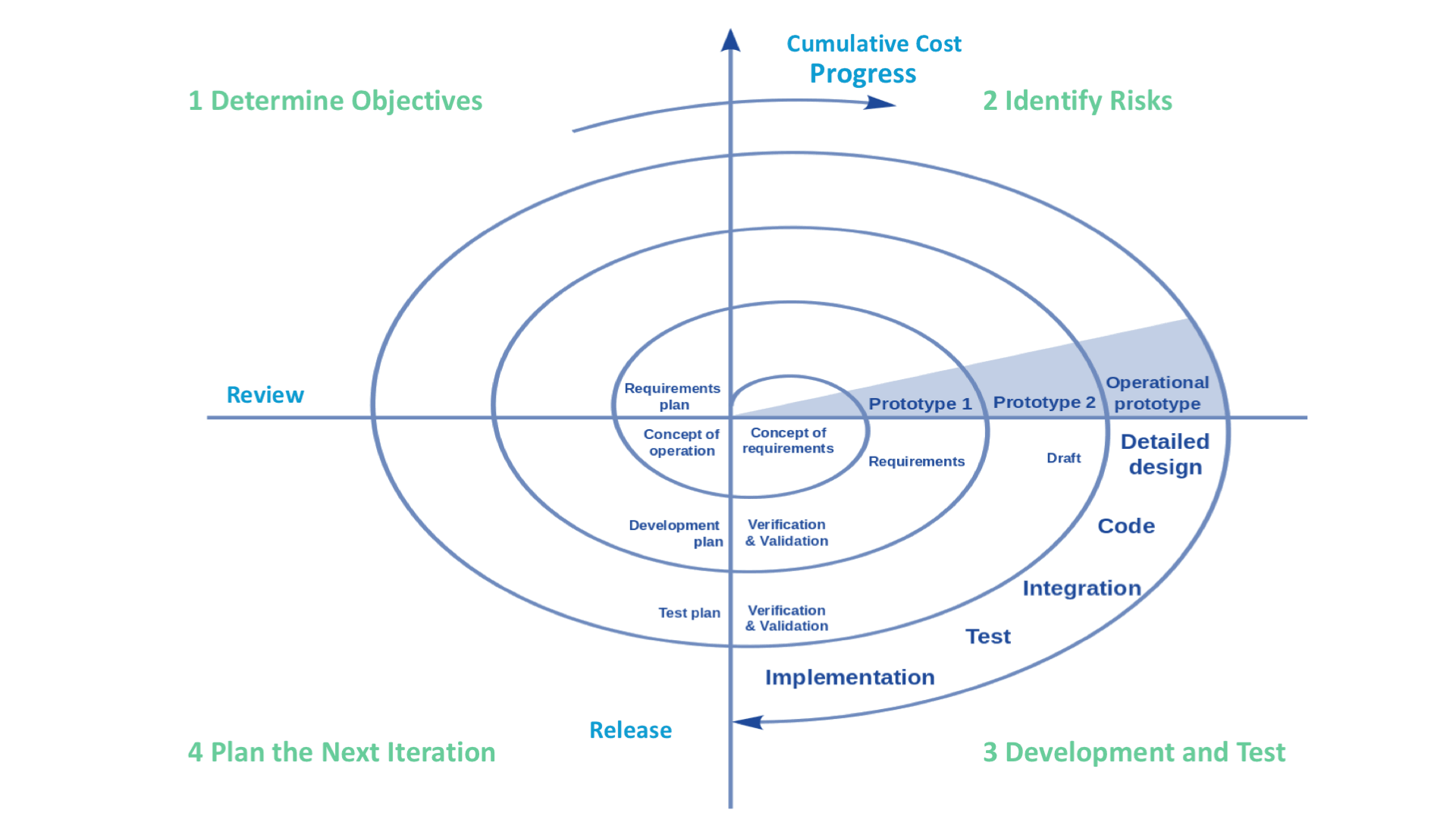
With all the corrections and changes there is possibility that the project will exceed expected time

**Iterative vs Agile**

The main difference between agile and iterative is that agile is a time-boxed and iterative approach to software delivery that builds and grows from the start of the project until delivering all at once near the end. But, iterative is a development model that allows taking a small set of requirements and enhancing it further until the entire product is ready to be deployed to the end-user or the customer.

**Spiral SDLC Model**

Spiral model – is SDLC model, which combines architecture and prototyping by stages. It looks like an Agile model with **a significant accent on risk analysis**. The main issue of the spiral model – is defining the right moment to make a step into the next stage. The preliminary set time frames are recommended as the solution to this issue. The shift to the next stage is done according to the plan, even if the work on the previous stage isn’t done yet. The plan is introduced based on the statistic data, received during the previous projects even from the personal developer’s experience.



*Advantages:*

Lifecycle is divided into small parts, and if the risk concentration is higher, the phase can be finished earlier to address the treats

The development process is precisely documented yet scalable to the changes

The scalability allows to make changes and add new functionality even at the relatively late stages

The earlier working prototype is done - sooner users can point out the flaws

*Disadvantages:*

Can be quite expensive

The risk control demands involvement of the highly-skilled professionals

Can be ineffective for the small projects

Big number of the intermediate stages requires excessive documentation

**Scrum framework**

SCRUM is an agile development process focused primarily on ways to manage tasks in team-based development conditions.

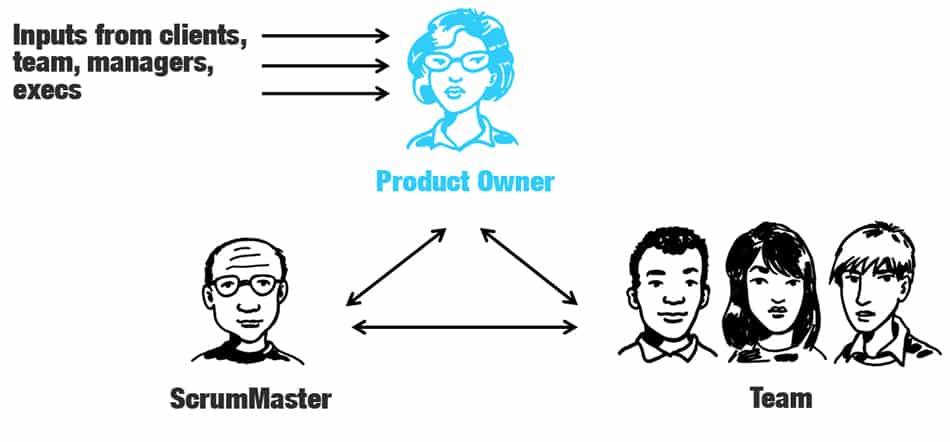
There are three roles in it, and their responsibilities are:

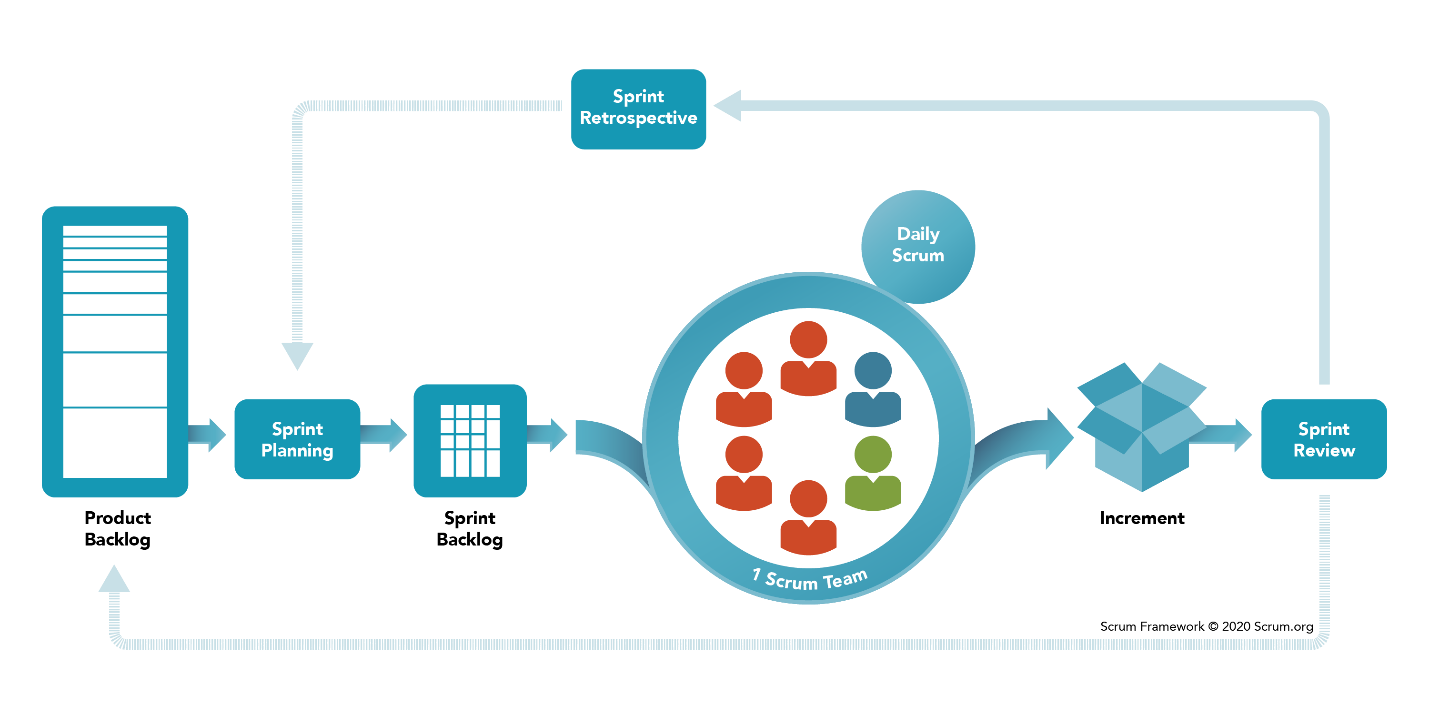
**Product owner (PO):** Is the representative of the stakeholders and customers who use the software. They focus on the business part and are responsible for the Return of Investment of the project. They translate the vision of the project to the team, validate the benefits in stories to be incorporated into the Product Backlog and prioritize them on a regular basis.

The product owner role is an important one and should only be taken up by someone who fully understands what the end users expect to see. It could be someone from the marketing or product testing department, but no matter who it is, they should be able to clearly communicate what the users expect to see from the project team. It’s also important the product owner has an idea of future trends in the niche and what measures or features competitors are implementing as a means of gaining more ground in the industry. However, that is also contingent on what type of product is being developed (e.g. hardware or software).

**Scrum master:** The person who leads the team guiding them to comply with the rules and processes of the methodology. Scrum master manages the reduction of impediments of the project and works with the Product Owner. The Scrum Master is in charge of keeping Scrum up to date, providing coaching, mentoring and training to the teams in case it needs it. The Scrum Master is responsible for ensuring each member of the team understands Scrum and their roles in it

**Scrum Team:** The main work within the Scrum Framework is carried out by a dedicated Scrum team; this group of individuals work together to develop and deliver the product. Ideally, it is a small cross-functional team, consisting of about 6 people (+,- 3 people) including business analysts, product testers and developers.





**Product Backlog** is the master list of work that needs to get done maintained by the product owner or product manager. This is a dynamic list of features, requirements, enhancements, and fixes that acts as the input for the sprint backlog. It is, essentially, the team’s “To Do” list. The product backlog is constantly revisited, re-prioritized and maintained by the Product Owner because, as we learn more or as the market changes, items may no longer be relevant or problems may get solved in other ways.

**Sprint Planning**

**Sprint Backlog** is the list of items, [user stories](https://www.atlassian.com/agile/project-management/user-stories), or bug fixes, selected by the development team for implementation in the current sprint cycle. Before each sprint, in the sprint planning meeting (which we’ll discuss later in the article) the team chooses which items it will work on for the sprint from the product backlog. A sprint backlog may be flexible and can evolve during a sprint. However, the fundamental sprint goal – what the team wants to achieve from the current sprint – cannot be compromised.

**Increment** (or Sprint Goal) is the usable end-product from a sprint. Usually it is demonstrated during the **end-of-sprint demo**, where the team shows what was completed in the sprint. *You may not hear the word “increment” out in the world, as it’s often referred to as the team’s definition of “Done”,* a milestone, the sprint goal, or even a full version or a [shipped epic](https://www.atlassian.com/agile/project-management/epics). It just depends on how your team defines “Done” and how you define your sprint goals. For example, some teams choose to release something to their customers at the end of every sprint. So their definition of ‘done’ would be ‘shipped’. However, this may not be realistic of other types of teams. Say you work on a server-based product that can only ship to your customers every quarter. You may still choose to work in 2-week sprints, but your definition of ‘done’ may be finishing part of a larger version that you plan to ship together. But of course, the longer it takes to release software, the higher the risk that software will miss the mark.