Software process model: Waterfall model, Iterative enhancement model, Prototyping model, Spiral model, Time boxing, V Shaped Models RAD model, 4th Generation model, Formal methods, Agile development.

Software processes are the activities for designing, implementing, and testing a software system. The software development process is complicated and involves a lot more than technical knowledge.

That's where software process models come in handy. A software process model is an **abstract representation** of the development process.

In this article, we will introduce you to the top seven software process models and discuss when to use them.

What is a software process model?

A software process model is an abstraction of the software development process. The models specify the stages and order of a process. So, think of this as a representation of the **order of activities** of the process and the **sequence** in which they are performed.

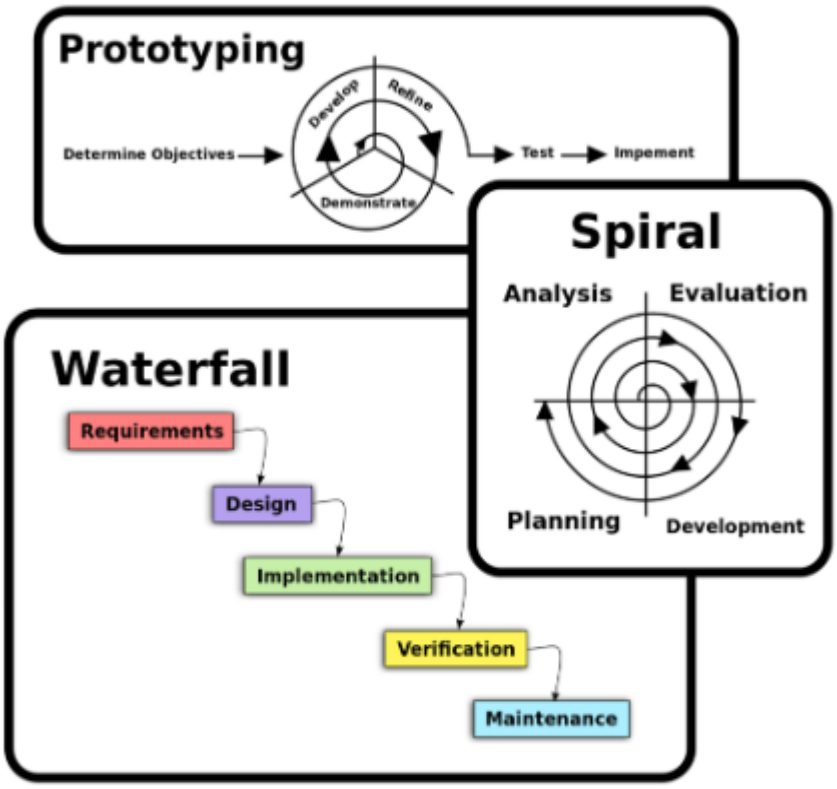
A model will define the following:

- The tasks to be performed
- The input and output of each task
- The pre and post-conditions for each task
- The flow and sequence of each task

The goal of a software process model is to provide guidance for controlling and coordinating the tasks to achieve the end product and objectives as effectively as possible.

There are many kinds of process models for meeting different requirements. We refer to these as **SDLC models** (Software Development Life Cycle models). The most popular and important SDLC models are as follows:

- Waterfall model
- V model
- Incremental model
- RAD model
- Agile model
- Iterative model
- Prototype model



Source: Omar Elgabry

- Spiral model

Factors in choosing a software process

Choosing the right software process model for your project can be difficult. If you know your requirements well, it will be easier to select a model that best matches your needs. You need to keep the following factors in mind when selecting your software process model:

Project requirements

Before you choose a model, take some time to go through the project requirements and clarify them alongside your organizations or team's expectations. Will the user need to specify requirements in detail after each iterative session? Will the requirements *change* during the development process?

Project size

Consider the size of the project you will be working on. Larger projects mean bigger teams, so you'll need more extensive and elaborate project management plans.

Project complexity

Complex projects may not have clear requirements. The requirements may change often, and the cost of delay is high. Ask yourself if the project requires constant monitoring or feedback from the client.

Cost of delay

Is the project highly time-bound with a huge cost of delay, or are the timelines flexible?

Customer involvement

Do you need to consult the customers during the process? Does the user need to participate in all phases?

Familiarity with technology

This involves the developers' knowledge and experience with the project domain, software tools, language, and methods needed for development.

Project resources

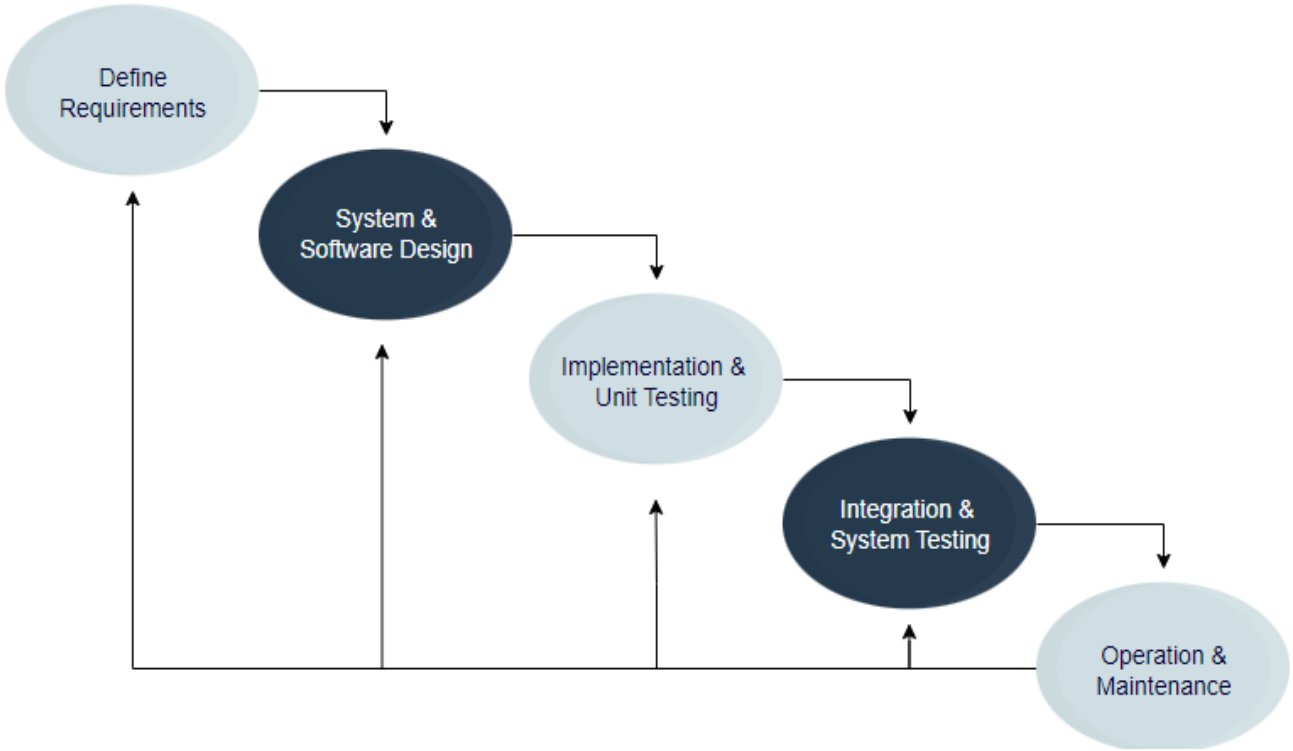
This involves the amount and availability of funds, staff, and other resources.

types of software process models

As we mentioned before, there are multiple kinds of software process models that each meet different requirements. Below, we will look at the top seven types of software process models that you should know.

Waterfall Model

The waterfall model is a **sequential, plan driven-process** where you must plan and schedule all your activities before starting the project. Each activity in the waterfall model is represented as a separate phase arranged in linear order.



It has the following phases:

- Requirements
- Design
- Implementation
- Testing
- Deployment
- Maintenance

Each of these phases produces one or more documents that need to be approved before the next phase begins. However, in practice, these phases are very likely to overlap and may feed information to one another.

The software process **isn’t linear**, so the documents produced may need to be modified to reflect changes.

The waterfall model is easy to understand and follow. It doesn’t require a lot of customer involvement after the specification is done. Since it’s inflexible, it can’t adapt to changes. There is no way to see or try the software until the last phase.

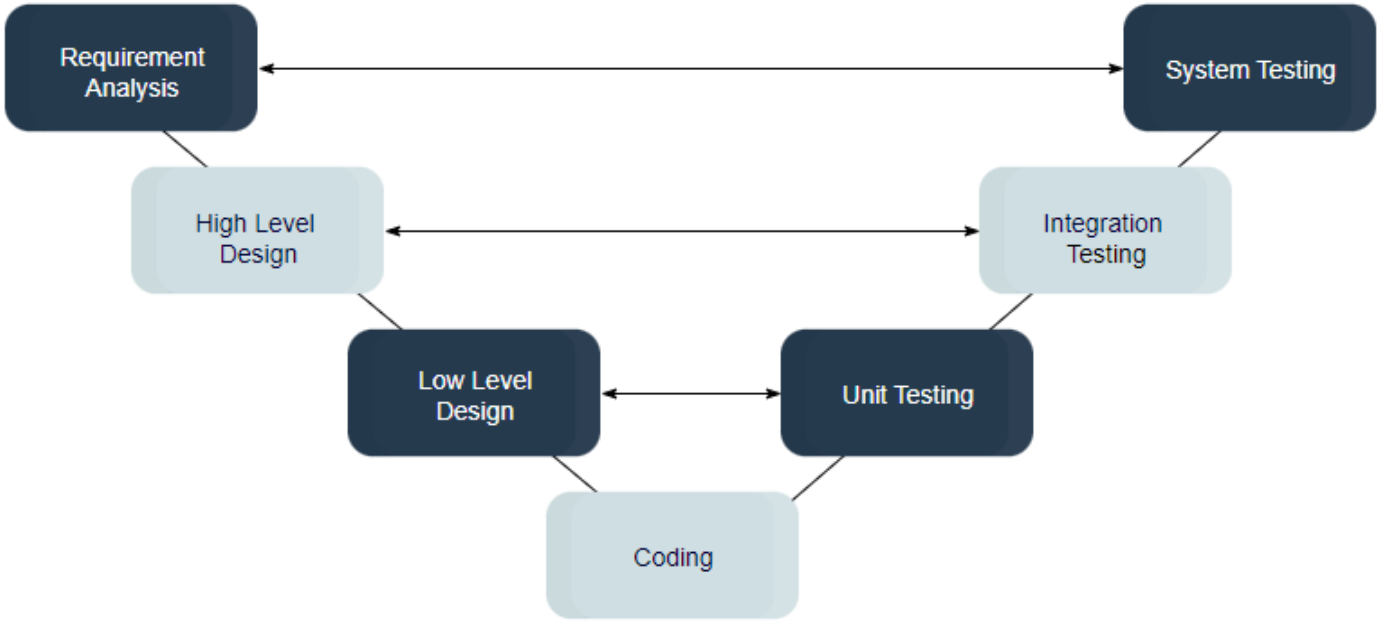
The waterfall model has a rigid structure, so it should be used in cases where the requirements are understood completely and unlikely to radically change.

V Model

The V model (Verification and Validation model) is an extension of the waterfall model. All the requirements are gathered at the start and cannot be changed. You have a corresponding testing activity for each stage. For every phase in the development cycle, there is an **associated testing phase.**

The corresponding testing phase of the development phase is planned in parallel, as you can see above.

The V model is highly disciplined, easy to understand, and makes project management easier. But it isn't good for complex projects or projects that have unclear or changing requirements. This makes the V model a good choice for software where downtimes and failures are unacceptable.



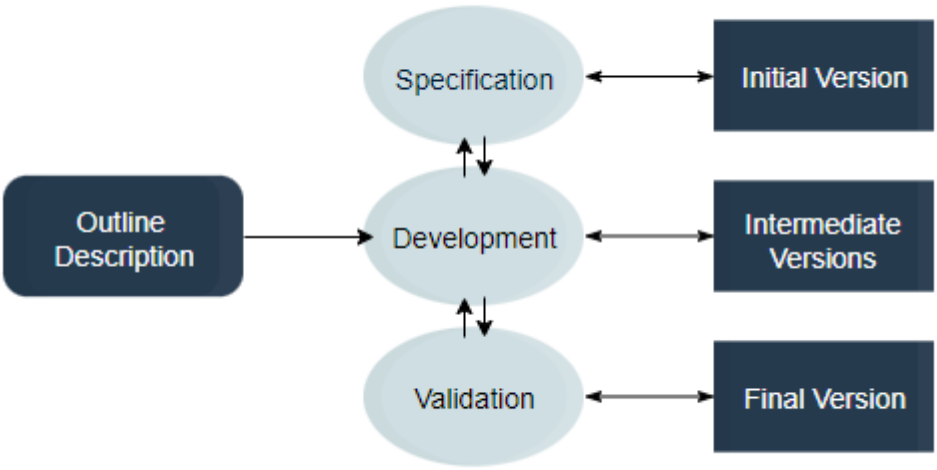
Incremental Model

The incremental model divides the system's functionality into **small increments** that are delivered one after the other in quick succession. The most important functionality is implemented in the initial increments.

The subsequent increments expand on the previous ones until everything has been updated and implemented.

Incremental development is based on developing an initial implementation, exposing it to user feedback, and evolving it through new versions. The process' activities are interwoven by feedback.

Each iteration passes through the requirements, design, coding, and testing stages.



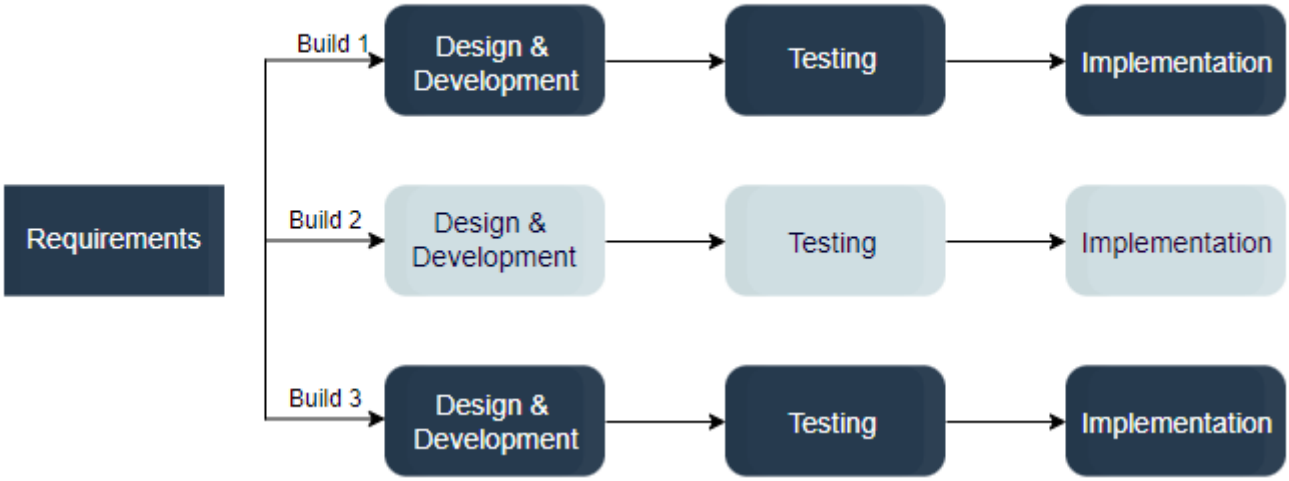
The incremental model lets stakeholders and developers see results with the first increment. If the stakeholders don't like anything, everyone finds out a lot sooner. It is efficient as the developers only focus on what is important and bugs are fixed as they arise, but you need a **clear and complete definition** of the whole system before you start.

The incremental model is great for projects that have loosely coupled parts and projects with complete and clear requirements.

Iterative Model

The iterative development model develops a system by **building small portions** of all the features. This helps to meet the initial scope quickly and release it for feedback.

In the iterative model, you start off by implementing a small set of software requirements. These are then **enhanced iteratively** in the evolving versions until the system is completed. This process model starts with part of the software, which is then implemented and reviewed to identify further requirements.



Like the incremental model, the iterative model allows you to see the results at the early stages of development. This makes it easy to identify and **fix any functional or design flaws**. It also makes it easier to manage risk and change requirements.

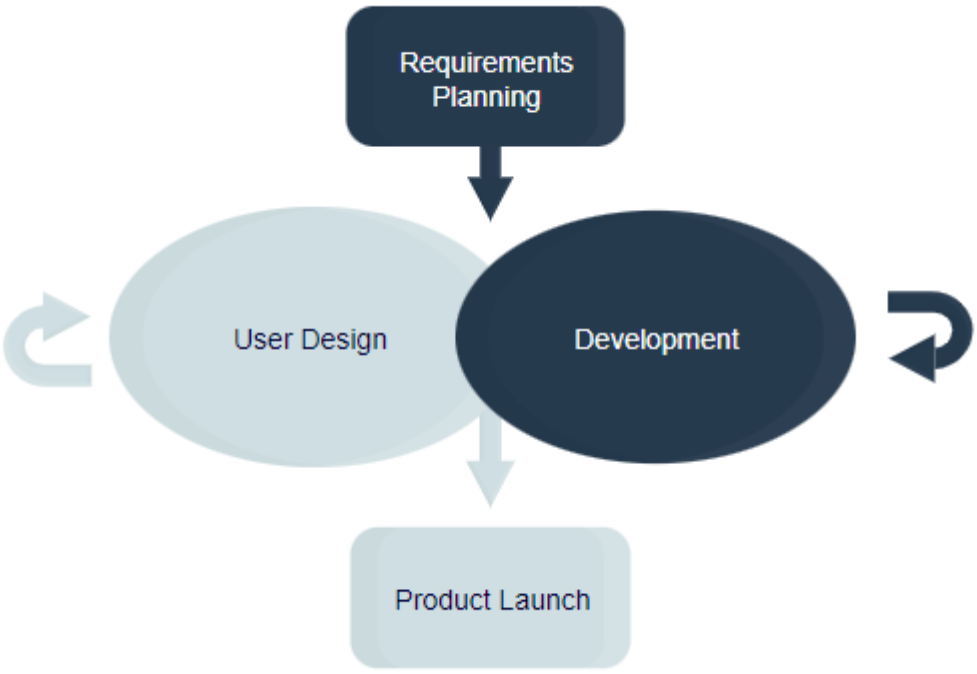
The deadline and budget may change throughout the development process, especially for large complex projects. The iterative model is a good choice for large software that can be **easily broken down into modules**.

RAD Model

The Rapid Application Development (RAD model) is based on iterative development and prototyping with **little planning involved**. You develop functional modules in parallel for faster product delivery. It involves the following phases:

1. Business modeling
2. Data modeling
3. Process modeling
4. Application generation
5. Testing and turnover

The RAD concept focuses on gathering requirements using focus groups and workshops, reusing software components, and informal communication.



The RAD model accommodates changing requirements, reduces development time, and increases the reusability of components. But it can be complex to manage. Therefore, the RAD model is great for systems that need to be produced in a short time and have known requirements.

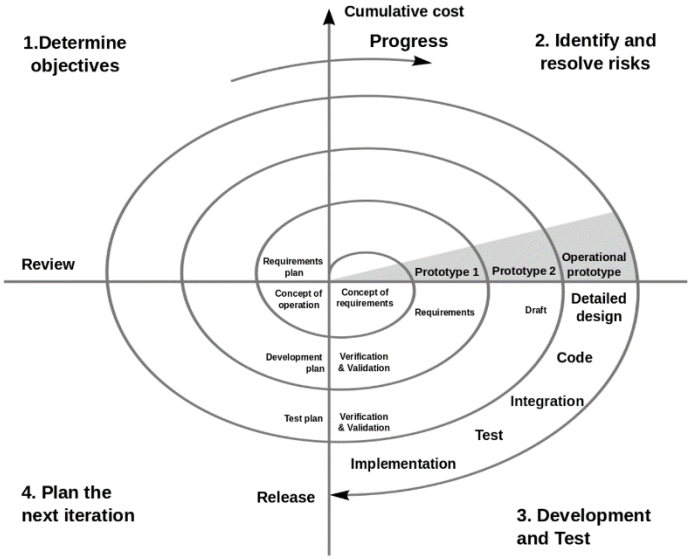
Spiral Model

The spiral model is a risk driven iterative software process model. The spiral model delivers projects in loops. Unlike other process models, its steps aren't activities but **phases** for addressing whatever problem has the greatest risk of causing a failure.

It was designed to include the best features from the waterfall and introduces risk-assessment.

You have the following phases for each cycle:

1. Address the highest-risk problem and determine the objective and alternate solutions
2. Evaluate the alternatives and identify the risks involved and possible solutions
3. Develop a solution and verify if it's acceptable
4. Plan for the next cycle



You develop the concept in the first few cycles, and then it evolves into an implementation. Though this model is great for managing uncertainty, it can be difficult to have stable documentation. The spiral model can be used for projects with **unclear needs** or projects still in research and development.

Agile model

The agile process model encourages **continuous iterations of development** and testing. Each incremental part is developed over an iteration, and each iteration is designed to be small and manageable so it can be completed within a few weeks.

Each iteration focuses on implementing a small set of features completely. It involves customers in the development process and minimizes documentation by using informal communication.

Agile development considers the following:

- Requirements are assumed to change
- The system evolves over a series of short iterations
- Customers are involved during each iteration
- Documentation is done only when needed

Though agile provides a very realistic approach to software development, it isn't great for complex projects. It can also present challenges during transfers as there is very little documentation. Agile is great for projects with **changing requirements**.

Some commonly used agile methodologies include:

Scrum: One of the most popular agile models, Scrum consists of iterations called sprints. Each sprint is between 2 to 4 weeks long and is preceded by planning. You cannot make changes after the sprint activities have been defined.

- **Extreme Programming (XP):** With Extreme Programming, an iteration can last between 1 to 2 weeks. XP uses pair programming, continuous integration, test-driven development and test automation, small releases, and simple software design.
- **Kanban:** Kanban focuses on visualizations, and if any iterations are used they are kept very short. You use the Kanban Board that has a clear representation of all project activities and their numbers, responsible people, and progress.