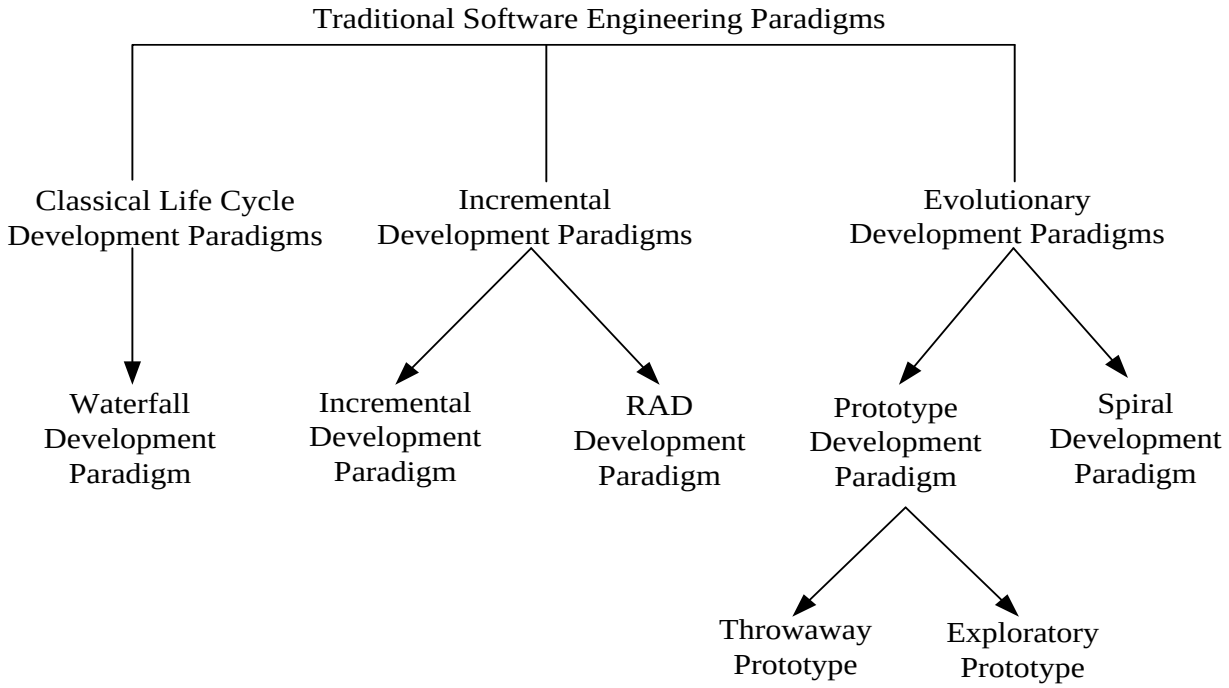
SOFTWARE DEVELOPMENT LIFE CYCLE MODELS

- A **software development process**, also known as a **software development life cycle (SDLC)**, is a structure imposed on the development of a software product.
- Software Development Life Cycle or SDLC is a model of a detailed plan on how to create, develop, implement and eventually fold the software.
- It's a complete plan outlining how the software will be born, raised and eventually be retired from its function.
- There are several models for such processes, each describing approaches to a variety of tasks or activities that take place during the process.
- Some people consider a lifecycle model a more general term and a software development process a more specific term.
- It aims to be the standard that defines all the tasks required for developing and maintaining software.

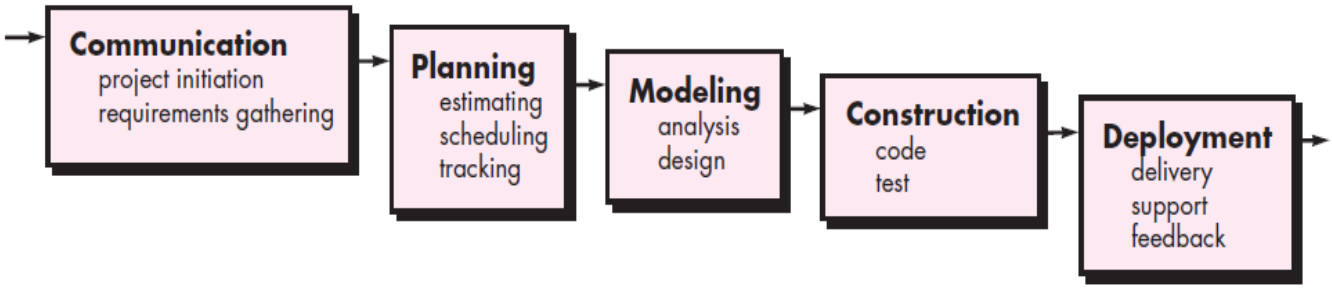


Classic life cycle paradigm:

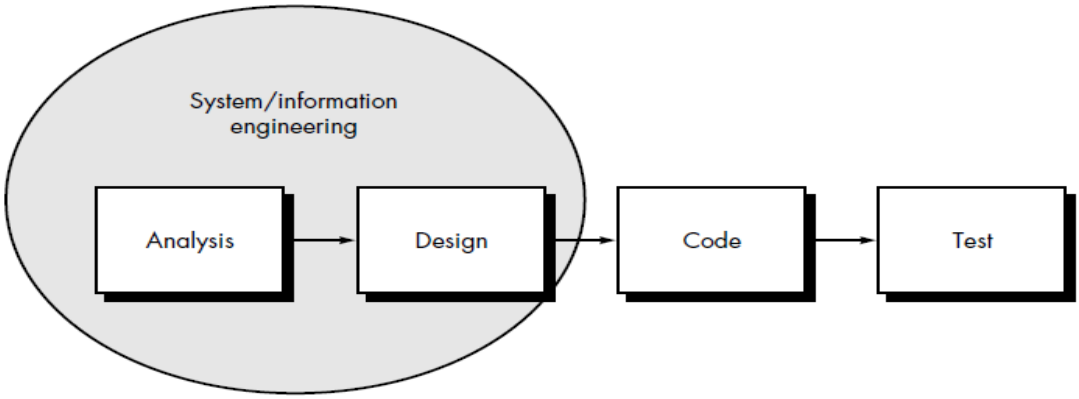
- The classic life cycle paradigm has a definite and important place in software engineering work.
- It provides a template into which methods for analysis, design, coding, testing, and support can be placed.
- The classic life cycle remains a widely used procedural model for software engineering.
- While it does have weaknesses, it is significantly better than a haphazard approach to software development.

1. Waterfall Model

- Waterfall model sometimes called the *classic life cycle* or the *waterfall model*, the *linear sequential model* suggests a systematic, sequential approach to software development that begins at the system level and progresses through analysis, design, coding, testing, and support.
- Following figure illustrates the linear sequential model for software engineering.
- Modeled after a conventional engineering cycle, the linear sequential model encompasses the following activities:



Or in abstracted form:



1. **System/information engineering and modeling:**

- Because software is always part of a larger system (or business), work begins by establishing requirements for all system elements and then allocating some subset of these requirements to software.
- This system view is essential when software must interact with other elements such as hardware, people, and databases.
- System engineering and analysis encompass requirements gathering at the system level with a small amount of top level design and analysis.
- Information engineering encompasses requirements gathering at the strategic business level and at the business area level.

2. **Software requirements analysis**

- The requirements gathering process is intensified and focused specifically on software.
- To understand the nature of the program(s) to be built, the software engineer ("analyst") must understand the information domain for the software, as well as required function, behavior, performance, and interface.
- Requirements for both the system and the software are documented and reviewed with the customer.

3. **Design**

- Software design is actually a multistep process that focuses on four distinct attributes of a program: data structure, software architecture, interface representations, and procedural (algorithmic) detail.
- The design process translates requirements into a representation of the software that can be assessed for quality before coding begins.
- Like requirements, the design is documented and becomes part of the software configuration.

4. **Code generation.**

- The design must be translated into a machine-readable form.
- The code generation step performs this task. If design is performed in a detailed manner, code generation can be accomplished mechanistically.

5. **Testing**

- Once code has been generated, program testing begins.
- The testing process focuses on the logical internals of the software, ensuring that all statements have been tested, and on the functional externals; that is, conducting tests to uncover errors and ensure that defined input will produce actual results that agree with required results.

6. Support

- Software will undoubtedly undergo change after it is delivered to the customer (a possible exception is embedded software).
- Change will occur because errors have been encountered, because the software must be adapted to accommodate changes in its external environment (e.g., a change required because of a new operating system or peripheral device), or because the customer requires functional or performance enhancements.
- Software support/maintenance reapplies each of the preceding phases to an existing program rather than a new one.

Advantage of Waterfall model:

- The waterfall model is the oldest and most widely used model in the field of software development.
- There are certain advantages of the waterfall model, which causes it to be the most widely used model as yet.

Some of them can be listed as under.

1. Needless to mention, it is a linear model and of course, linear models are the most simple to be implemented.
2. The amount of resources required to implement this model is minimal.
3. One great advantage of the waterfall model is that documentation is produced at every stage of the waterfall model development. This makes the understanding of the product designing procedure simpler.
4. After every major stage of software coding, testing is done to check the correct running of the code.

Disadvantages of Waterfall model:

Among the problems that are sometimes encountered when the linear sequential model is applied are:

1. Real projects rarely follow the sequential flow that the model proposes. Although the linear model can accommodate iteration, it does so indirectly. As a result, changes can cause confusion as the project team proceeds.
2. It is often difficult for the customer to state all requirements explicitly. The linear sequential model requires this and has difficulty accommodating the natural uncertainty that exists at the beginning of many projects.
3. The customer must have patience. A working version of the program(s) will not be available until late in the project time-span. A major blunder, if undetected until the working program is reviewed, can be disastrous.
4. The linear nature of the classic life cycle leads to “blocking states” in which some project team members must wait for other members of the team to complete dependent tasks. In fact, the time spent waiting can exceed the time spent on productive work! The blocking state tends to be more prevalent at the beginning and end of a linear sequential process. Each of these problems is real.