Week 1

CS2212
Introduction to Software Engineering

Process Models Part 1

Some images and content from Software Engineering: A Practitioner's Approach 9th Edition By Roger Pressman and Bruce Maxim

Recall From Last Class

Process: a collection of **activities**, **actions**, and **tasks** that are performed when some work product is to be created.

Activity: strives to achieve a broad objective (e.g. communication with stakeholders) and is applied regardless of the application domain, size, complexity, etc.

Action: encompasses a set of **tasks** (e.g. architectural design) that produce a major work product (e.g. an architectural model).

Tasks: focuses on small, but well-defined objectives (e.g. conducting a unit test) that produces a tangible outcome.

There are five key activities applicable to all software projects:

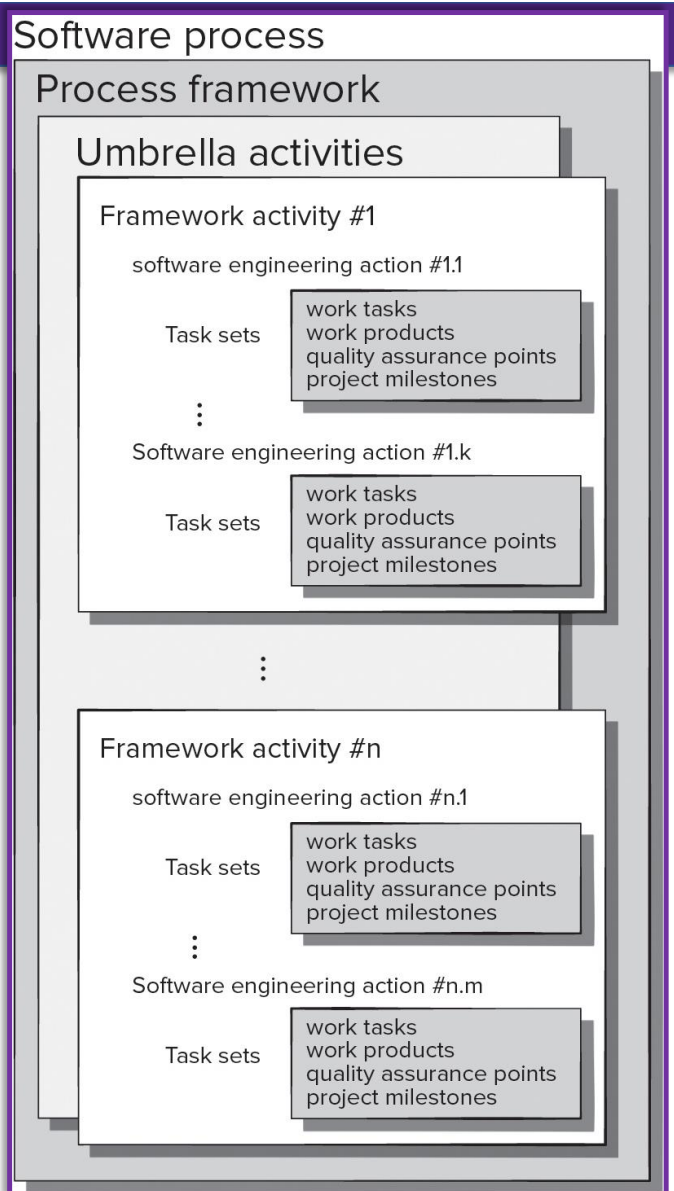
- Communication
- Planning
- Modeling
- Construction
- Deployment

Software Process

- A **software process** is a framework for the **activities**, **actions**, and **tasks** that are required to build high-quality software *(high-quality software is the work product in this case).*
- A **software process** defines the **approach taken** as software is developed and engineered.
- **Software processes** entail a series of **predictable steps** that, if followed, should produce a timely and quality result.

Software Process Framework

- Each **activity** is populated by a set of **actions**.
- Each **action** is defined by a **task set** that identifies the work tasks, the work products, the quality assurance points, and milestones.
- **Umbrella activities** complement **framework activities** and are done throughout the process.



Software process diagram is Figure 2.1 from our textbook.

Process Flows

- It is important to note that five key **activities** are **not** necessarily performed in this order or linearly:
  1. Communication
  2. Planning
  3. Modeling
  4. Construction
  5. Deployment
- There are many different ***process flows*** for these **activities** that determine how these **activities** are organized and sequenced over time.

Linear Process Flow

Each of the process activities are executed in sequence.

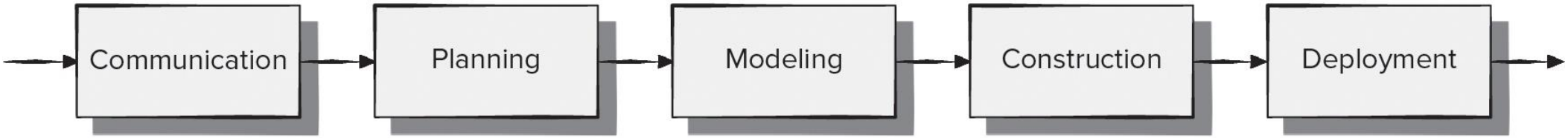


Figure 2.2 a) from our textbook.

Iterative Process Flow

One or more of the activities are repeated before proceeding to the next.

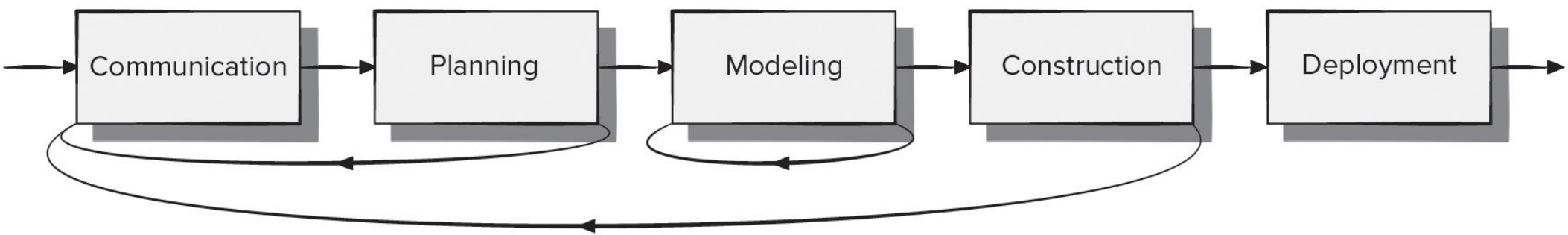
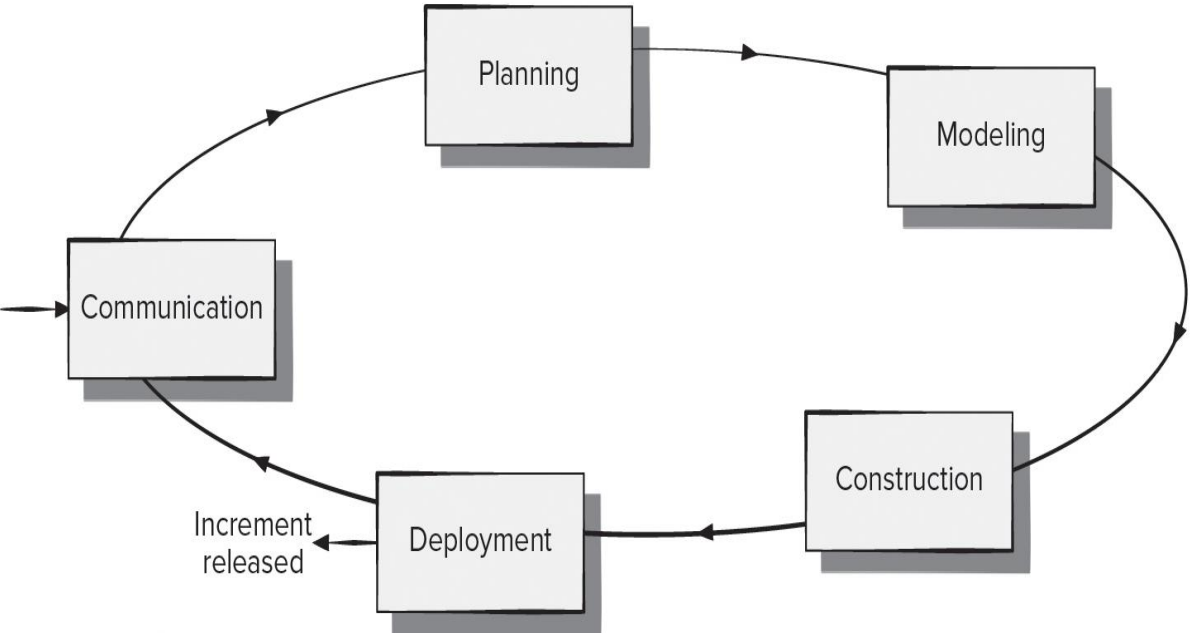


Figure 2.2 b) from our textbook.

Evolutionary Process Flow



Activities are executed in a "circular" manner, with each complete circuit leading to a more complete version of the software.

Figure 2.2 c) from our textbook.

Parallel Process Flow

One or more activities are executed in parallel with other activities (for example, modeling for one aspect might be executed at the same time that another aspect is under construction).

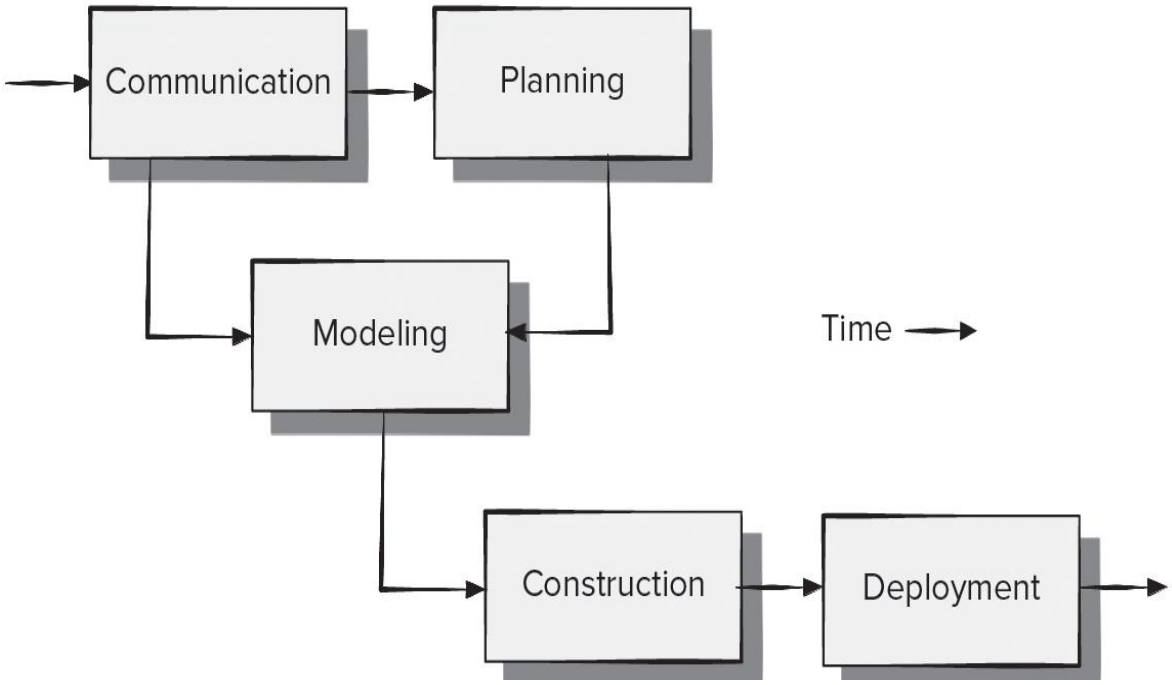


Figure 2.2 d) from our textbook.

Defining Framework Activities

- The key **activities** and definitions we have discussed so far are not enough information to properly execute them.
- Must answer questions such as:
  - *What **actions** are appropriate for each **framework activity** given our problem, team members, resources, and stakeholders?*
- **Example:** ***Communication*** **activity** for a one-man team
  - Only **action** might just be a phone call to the one **stakeholder**.
  - **Task set** would contain:
    1. Make contact with stakeholder via telephone.
    2. Discuss requirements and develop notes.
    3. Organize notes into a brief written statement of requirements.
    4. E-Mail requirements to stakeholder fore review.

Identifying a Task Set

- A **task set** defines the **actual work** to be done to accomplish the objectives of a software engineering action.
- Different projects require different task sets.
- Must choose a **task set** that accommodates the **needs of the project** and the **characteristics of the team**.
- A **task set** is defined by creating several lists:
  - A list of the tasks to be accomplished.
  - A list of the work products to be produced.
  - A list of the quality assurance filters to be applied.

Process Assessment and Improvement

- The existence of a software **process** is **no guarantee** that software will be delivered on time, or meet the customer's needs, or that it will exhibit long-term quality characteristics.
- Any software **process** can be assessed to ensure that it meets a set of basic **process** criteria that have been shown to be essential for successful software engineering.
- Software **processes** and **activities** should be assessed using numeric measures or software analytics *(metrics)*.

Prescriptive Process Models

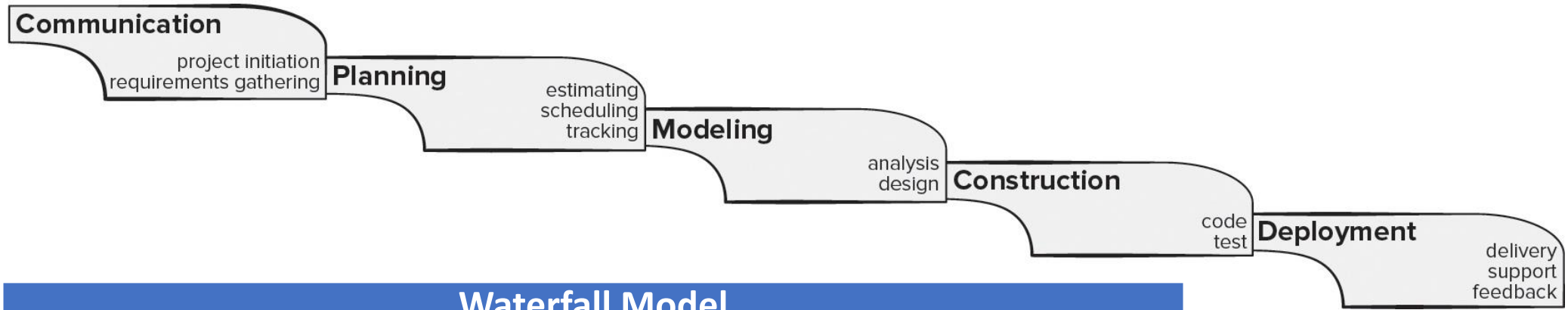
- Prescriptive process models advocate an **orderly** approach to software engineering.

- **That leads to two questions:**

  1. If prescriptive process models strive for structure and order, are they appropriate for a software world that thrives on change?

  2. If we reject traditional process models and replace them with something less structured, do we make it impossible to achieve coordination and coherence in software work?

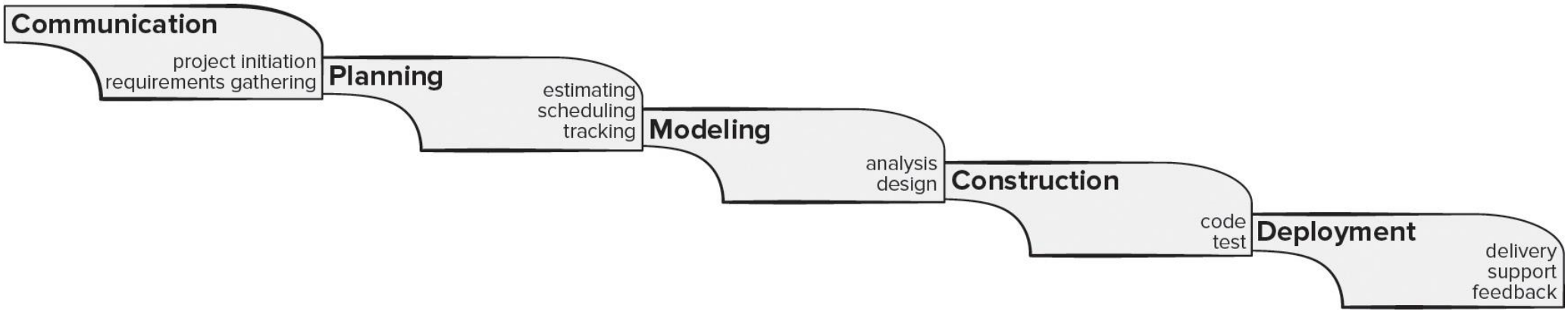
Waterfall Process Model



Waterfall Model

- The waterfall model, also called the “classic life cycle” is a systematic, sequential approach to development
- Customers specify requirements, after which planning, modeling, construction, and deployment of the software occurs
- Ongoing support and maintenance of the completed software proceeds from there
- It is the simplest and oldest paradigm for software engineering

Waterfall Process Model



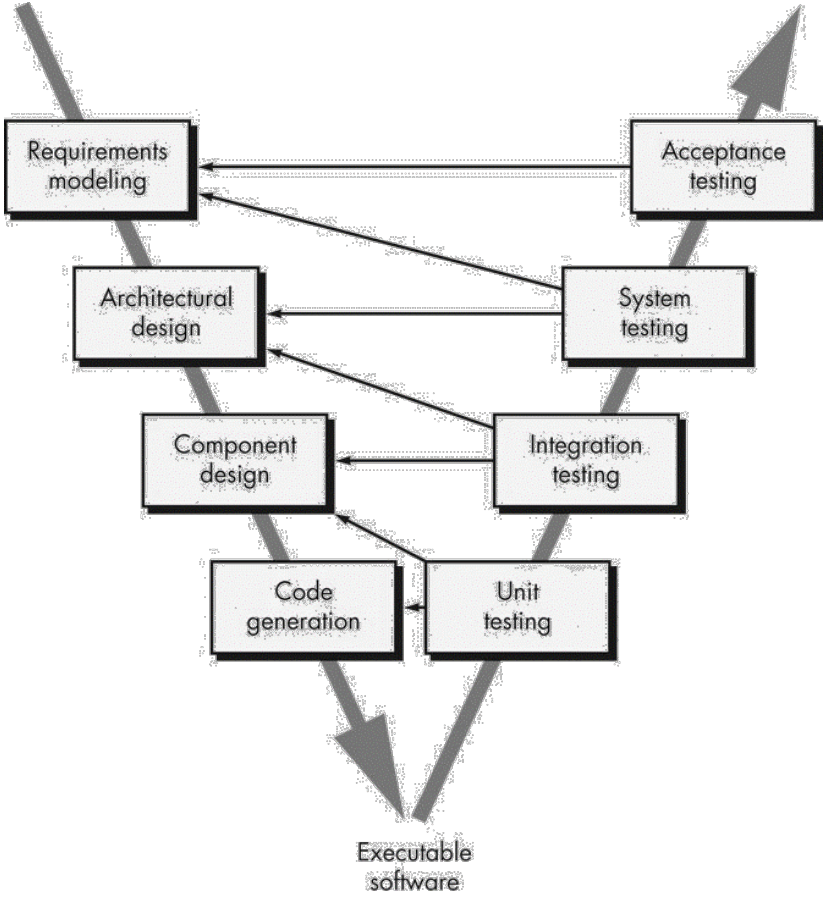
Pros

- It is easy to understand and plan.
- It works for well-understood small projects.
- Analysis and testing are straightforward.

Cons

- It does not accommodate change well.
- Testing occurs late in the process.
- Customer approval is at the end.

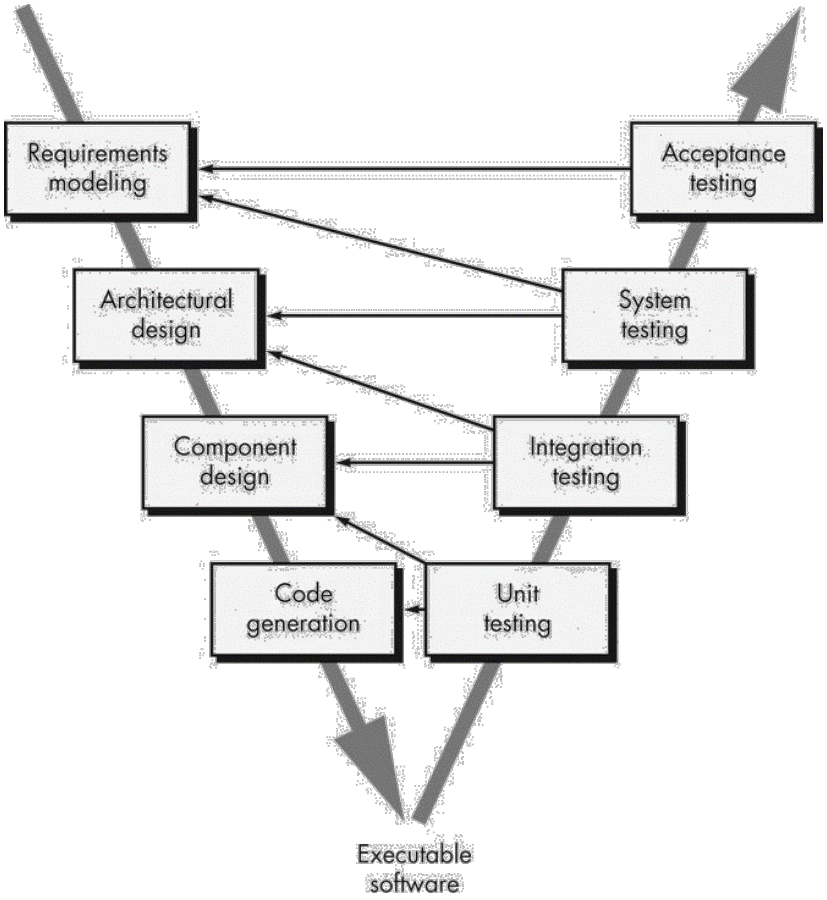
V-Model Process Model



V-Model

- A variation of the waterfall model is the V-model.
- The V-model attempts to capture the relationship between quality assurance actions and the actions associated with communication, modelling, and early construction activities.
- Once code has been generated, the team moves up the right side of the V carrying out a series of tests that validate the work done as the team moved down the left side during development.

V-Model Process Model



Pros

- Remains easy to understand and plan.
- It works for well-understood small projects.
- Analysis and testing are straightforward.
- Good way of visualizing how verification and validation actions are applied to earlier engineering work

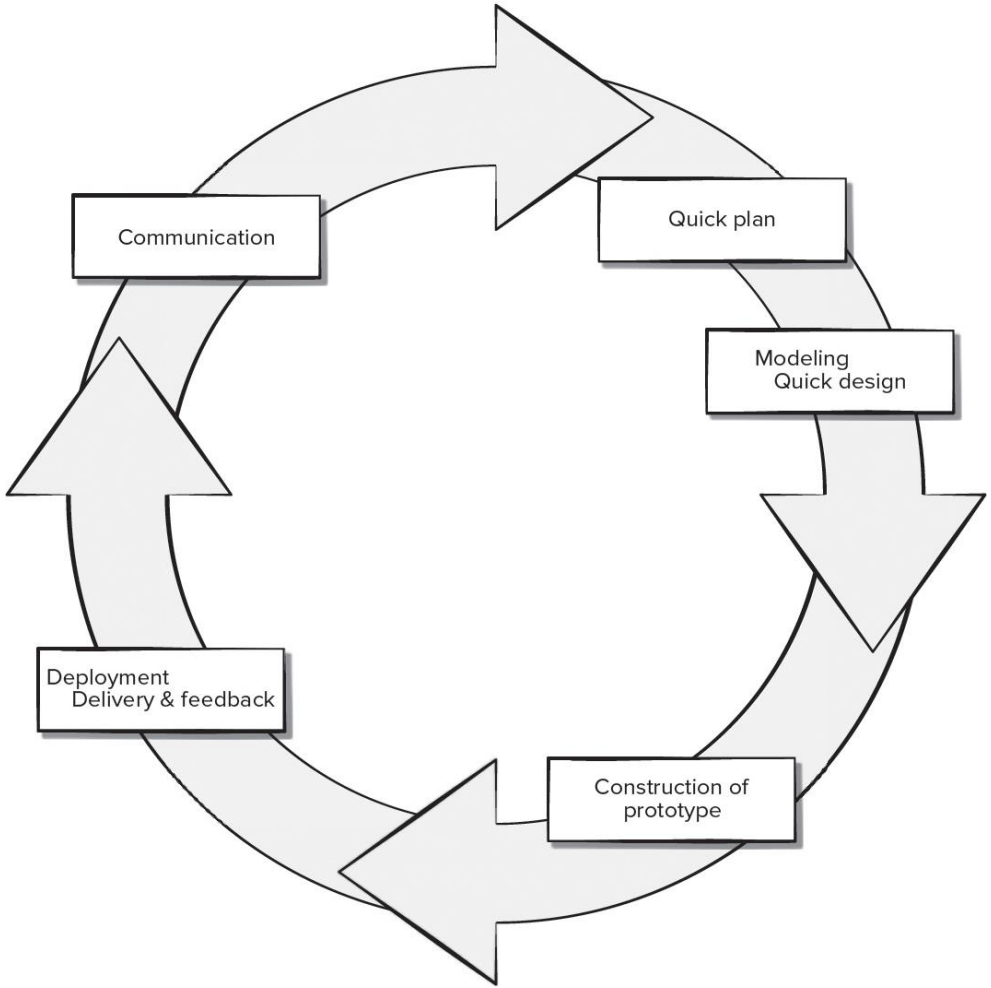
Cons

- No fundamental difference between the V-model and the waterfall model.
- Waterfall model is being split and folded up to create the shape of the V-model.
- It does not accommodate change well.
- Testing occurs late in the process.
- Customer approval is at the end.

Prototyping Process Model

Prototyping Model

- Prototyping begins with communication, collecting initial known requirements and outlining areas where further definition is necessary.
- A prototyping iteration is planned, with quick models and designs created, focusing on aspects of the software visible to end users.
- This design work leads to the construction of a prototype.
- This prototype is deployed and evaluated by stakeholders, who provide feedback to further refine requirements and design.



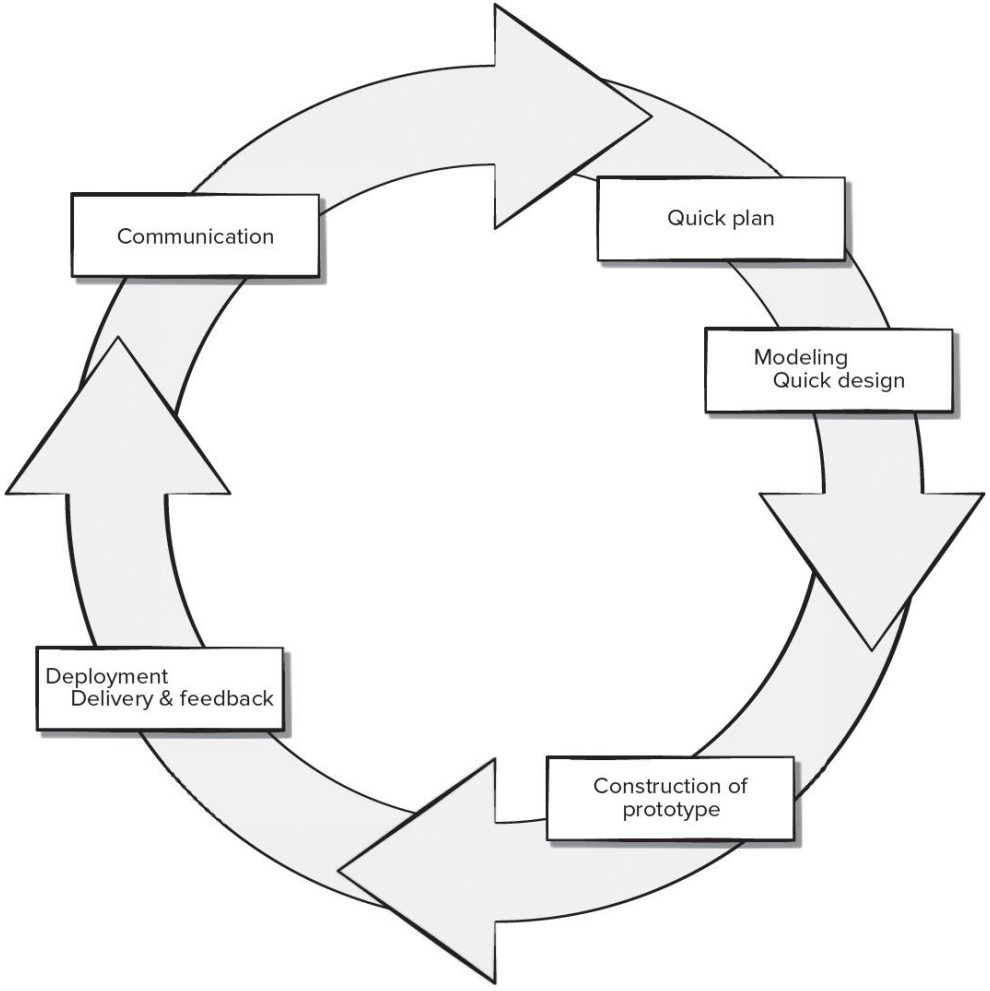
Prototyping Process Model

Pros

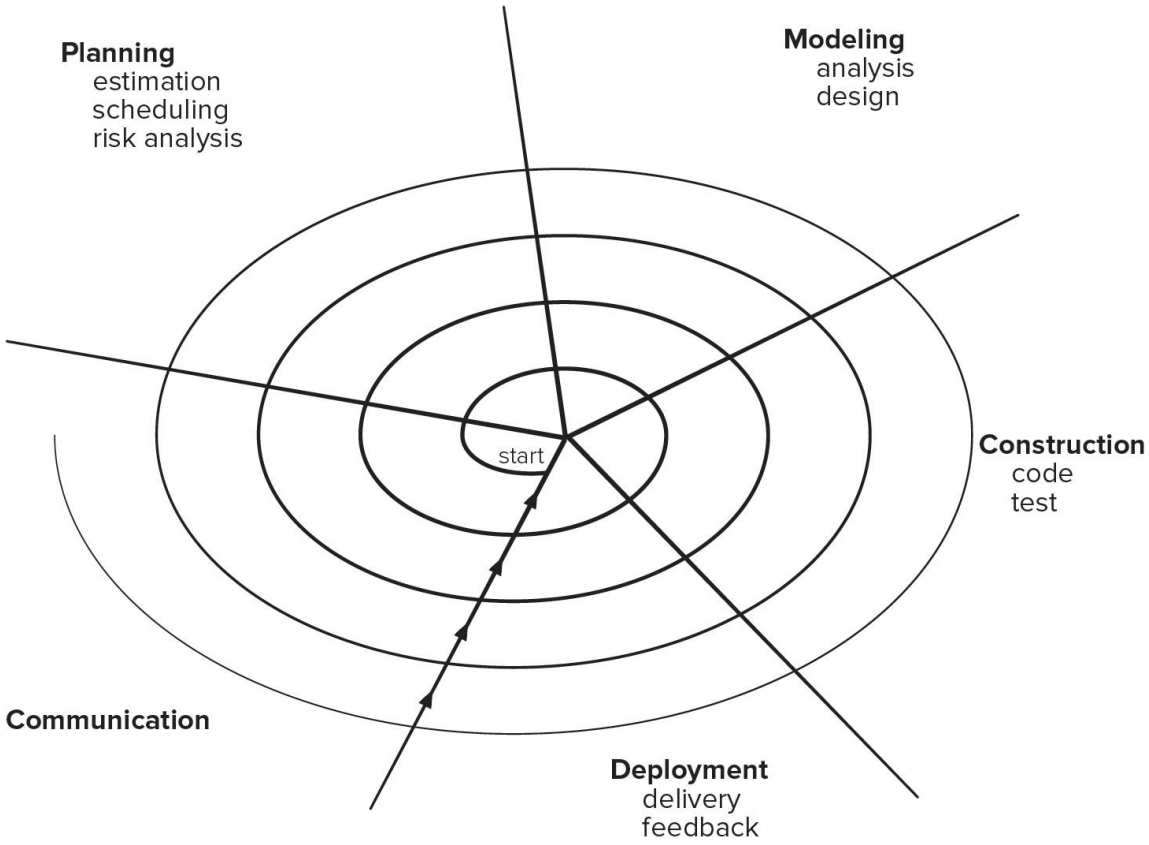
- Reduced impact of requirements changes.
- Customer is involved early and often.
- Works well for small projects.
- Reduced likelihood of product rejection.

Cons

- Customer involvement may cause delays.
- Temptation to “ship” a prototype.
- Work lost in a throwaway prototype.
- Hard to plan and manage.



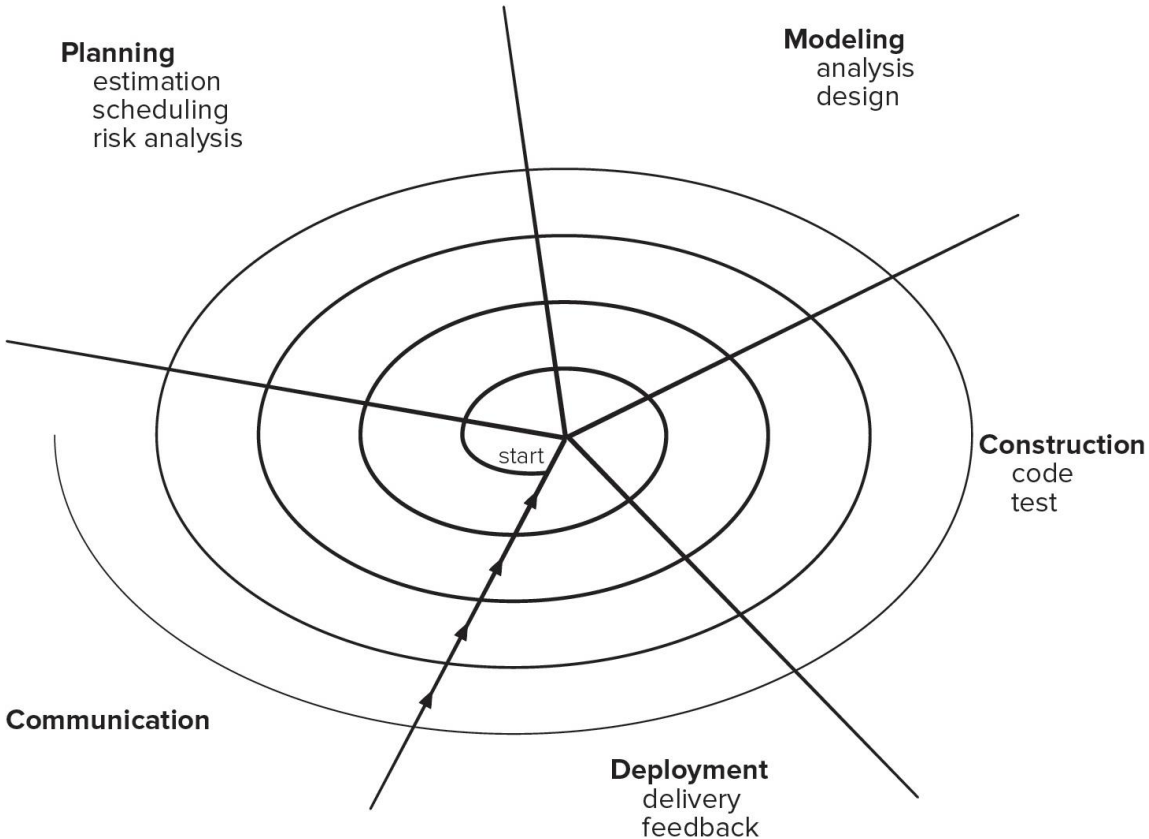
Spiral Process Model



Spiral Model

- Evolutionary approach that couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model.
- Risk-driven approach in which each cycle through the model incrementally grows the software's degree of definition and implementation while also decreasing its degree of risk.
- Prototyping becomes a risk reduction mechanism that moves the software closer and closer to an ideal form each cycle.

Spiral Process Model



Pros

- Continuous customer involvement.
- Development risks are managed.
- Suitable for large, complex projects.
- It works well for extensible products.

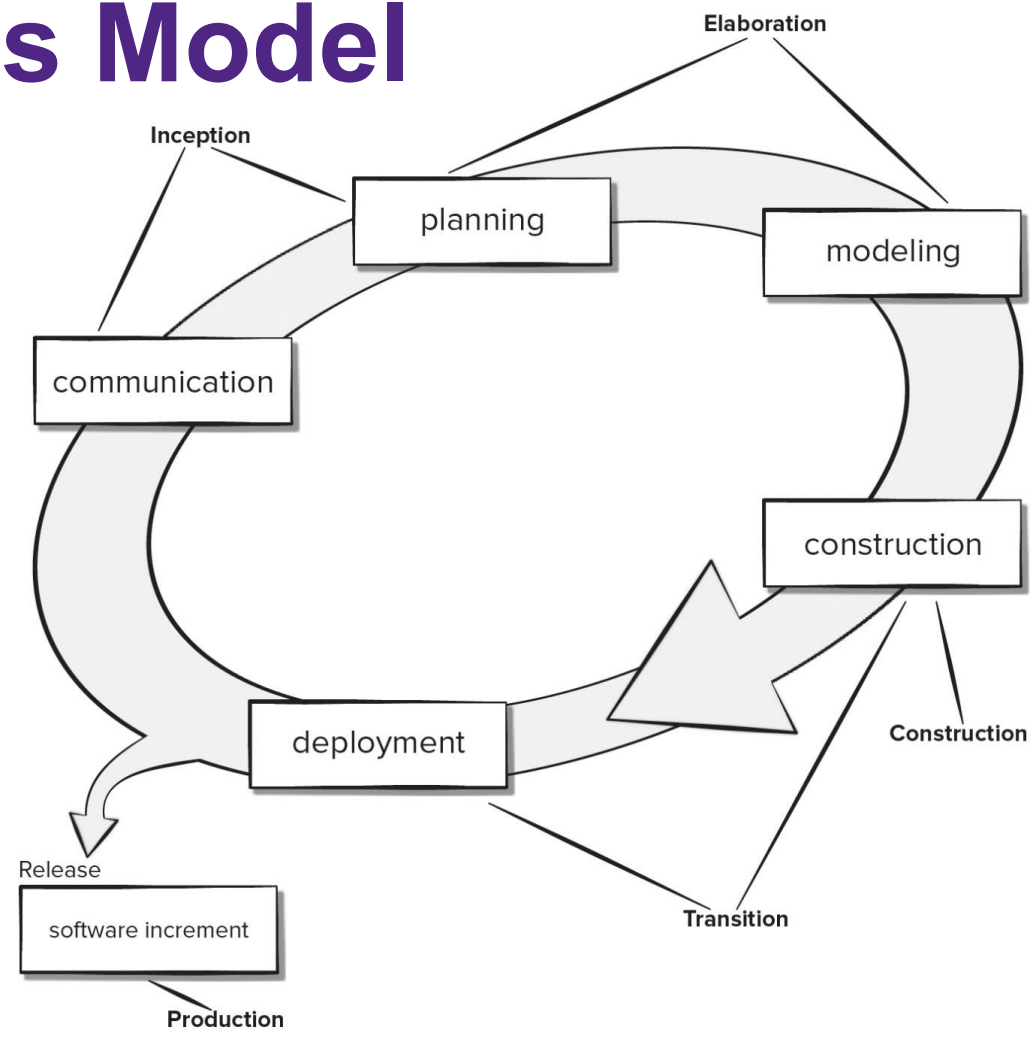
Cons

- Risk analysis failures can doom the project.
- Project may be hard to manage.
- Requires an expert development team.

The Unified Process Model

Unified Model

- The Unified Process (UP) is a "use-case driven, architecture-centric, iterative and incremental" software process closely aligned with the Unified Modeling Language (UML).
- It consists of a number of phases that map to the generic framework activities we have be using thus far.



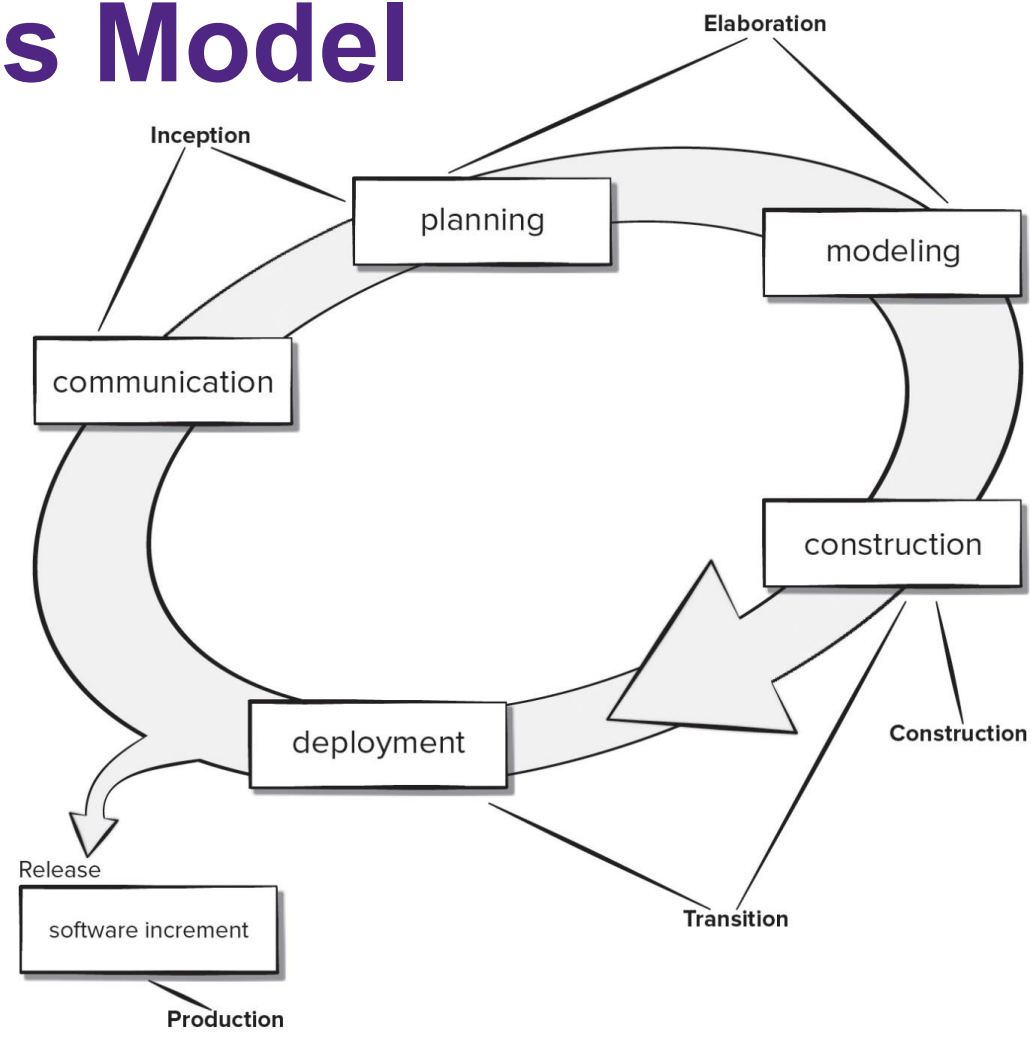
The Unified Process Model

Pros

- Quality documentation emphasized.
- Continuous customer involvement.
- Accommodates requirements changes.
- Works well for maintenance projects.

Cons

- Use cases are not always precise.
- Tricky software increment integration.
- Overlapping phases can cause problems.
- Requires expert development team.



And Many More

- There are many more Prescriptive Process Models.
- All share the same issues with change due to their highly structured nature.
- Still appropriate for well understood projects or large teams.