COSC 412
Software Process and Process Models

Spring 2025

Dr. Akshita Maradapu Vera Venkata Sai

Department of Computer and Information Sciences

Towson University

Outline

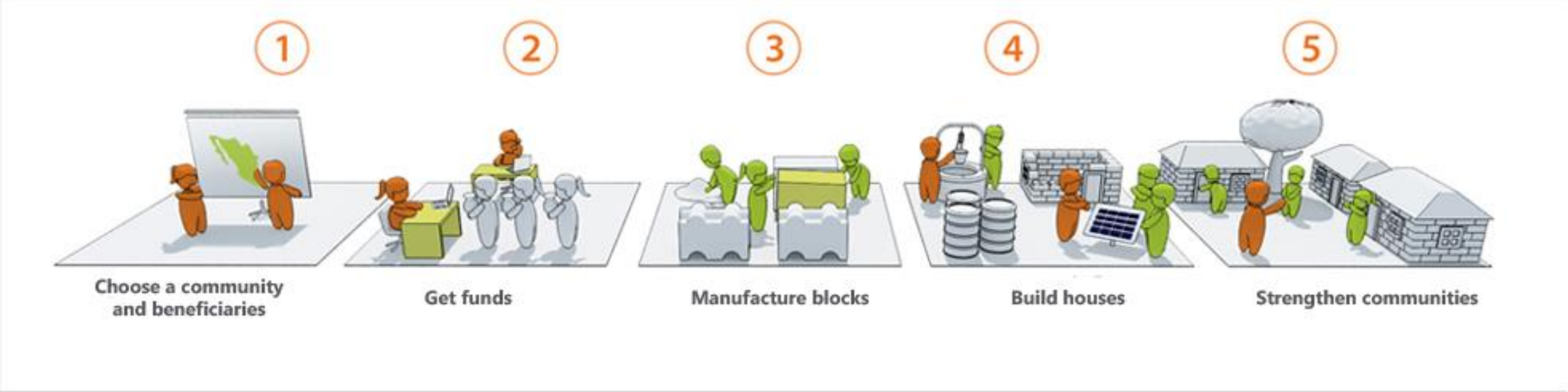
- Software Process
  - Framework
  - Umbrella Activities
- Process Model
  - Types of Flow
  - Perspective Process Models
- Coping with change
- Process Improvement

Problem

- You have a whole team of software engineers
- Someone asks you to build a software system, what would you do? How to start?
  - Joe, write the code.
  - Michele, design the UI.
  - Mike, I need you to set up the server.
  - Tim, go build the db.

We need to follow a process

- If you were building new houses, what is the process?
  - Discuss with your neighbors.
  - Customer requirements?
  - Plan?
  - Quality control?



Software Process

Definition:

Software process is a collection of ***activities, actions and tasks*** performed to create a work product.

- A Software Process has the following:
  - Framework Activities
  - Umbrella Activities

Software Process

Activity

- It is a broad objective that we want to achieve with respect to the product.
  - ✗Application domain
  - ✗Project size
  - ✗Project complexity

Example : Developing a video game :

Meeting with stakeholders

Defining the budget and timeline

Software Process

Action

It is a set of tasks focused on producing a major work product

For example:

An architectural model, or a general framework/ prototype of a game

Software Process

Task

- Focuses on a small, but well-defined objective that produces a tangible outcome.

- For example:
  - Creating a avatar in a video game.
  - Testing all the components of an avatar, etc.

Process Activities

Process Framework

This establishes the foundation to complete the software engineering process

Framework Activities include:

- Communication
- Planning
- Modeling
- Construction
- Deployment

Framework Activities

- Communication
  - Before start anything, it is important to communicate and collaborate with the stakeholders.
  - Intent:
    - to understand stakeholders’ objectives for the project
      - *What is my general problem or what does my client need?*
    - to gather requirements that help define software features and functions
    - identify: risks, required resources, work products, schedule, budget
  - For building a house, we need to ask
    - How many bedrooms?
    - Walk-out basement?
    - Laminate or hardwood?

Framework Activities

- Planning
  - Any complicated journey can be simplified if a map exists
  - Create a map to guide the team
  - software project plan includes
    - the *technical tasks* to be conducted, the *risks* that are likely, the *resources* that will be required, the *work products* to be produced, and a work *schedule*.
  - Identify parts of problem you have seen before
    - Patterns?
    - Existing code to implement part or all of task?
      - i.e. An existing library
      - For example – if your problem requires parsing a csv file then there is tons of code available online for free to handle this need already.

Framework Activities

- Modeling
  - If you build a bridge, create a sketch first, then go into details
  - creating models to better understand software requirements and the design that will achieve those requirements
  - Class diagrams, flow charts, etc.
- Construction
  - Write code and build the software
    - Can trace back to design model?
  - Test it
    - Does it meet the requirements?
- Deployment
  - Deliver to customers, evaluate, and get feedback

Framework Activities

- These five framework activities can be used during
  - the development of small, simple programs,
  - the creation of Web applications, and
  - the engineering of large, complex computer-based systems.

- The details of the software process will be quite different in each case, but the framework activities remain the same.

Umbrella Activities

- Applied *throughout a software project* and help a software team manage and control progress, quality, change, and risk
- Include:
  - Software project tracking and control
  - Risk management
  - Software quality assurance
  - Technical reviews
  - Measurement
  - Software configuration management
  - Reusability management
  - Work product preparation and production

Process Adaptation

- Software engineering process is *not* a rigid prescription
- A process for project A might be significantly different from a process of project B
- Need to consider:
    - Overall flow of activities, actions or tasks and how they depend on each other
    - Level of customer involvement in project
        - Can we keep bothering them to check our prototype?
    - How well have work products been identified
        - Clear or vague?
    - Level of organization in team
    - Degree to which team is “self-governed”
        - Is management heavily involved in day-to-day activities or is team empowered to get the job done without constant oversight?

Process Models

- A process model provides a specific roadmap for software engineering work.

- Defines the ***flow*** of all activities, actions, tasks, the degree of iteration, the work products, the organization of the work.

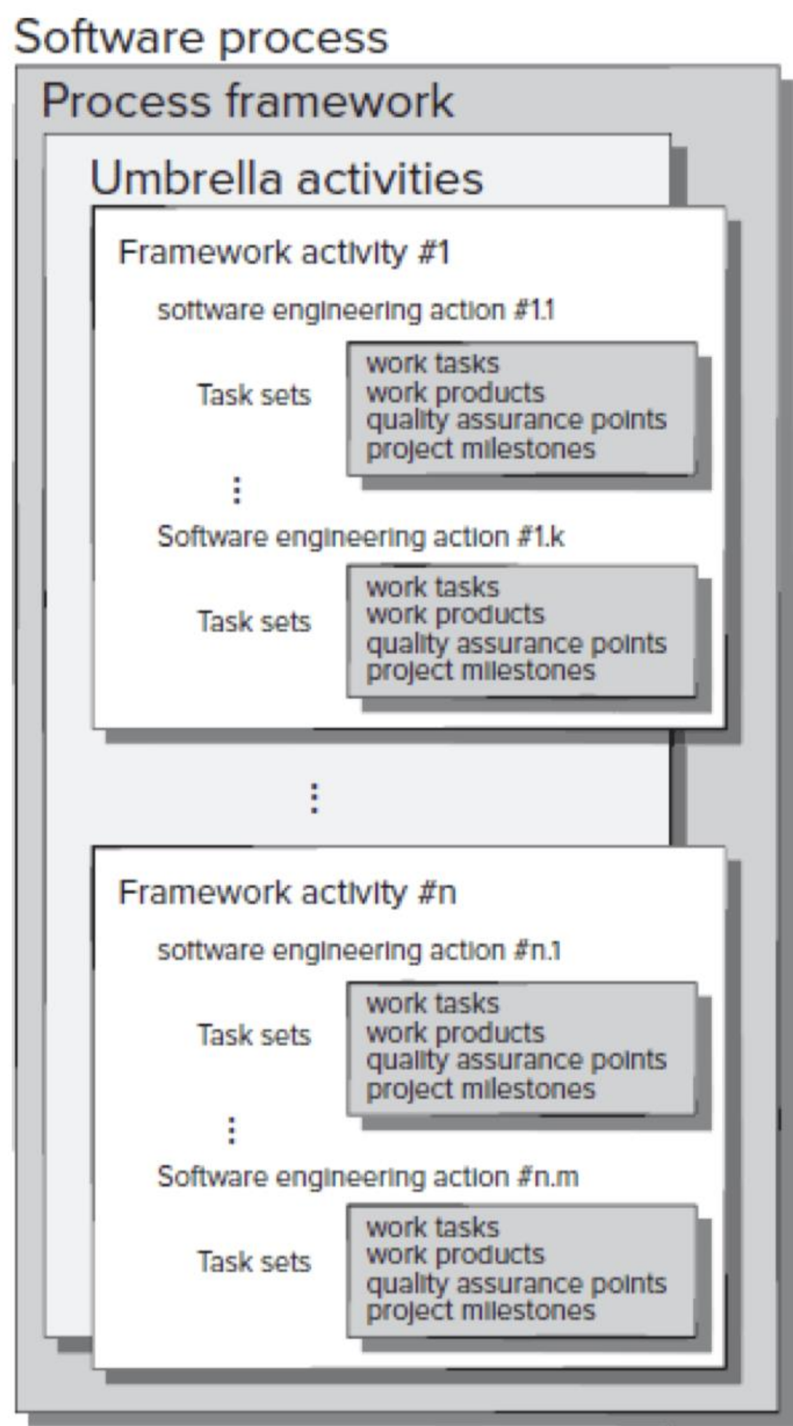
- Who does it? Software engineers and managers
  - So, you need to know it…

Process

What is a software process?

Each of these activities, actions, and tasks resides within a framework or model that defines their relationship with the process and with one another.

- Each *framework activity* is populated by a set of software engineering actions.
- Each software engineering *action* is defined by a task set that identifies the *work tasks* that are to be completed, the *work products* that will be produced, the *quality assurance points* that will be required, and the *milestones* that will be used to indicate progress.

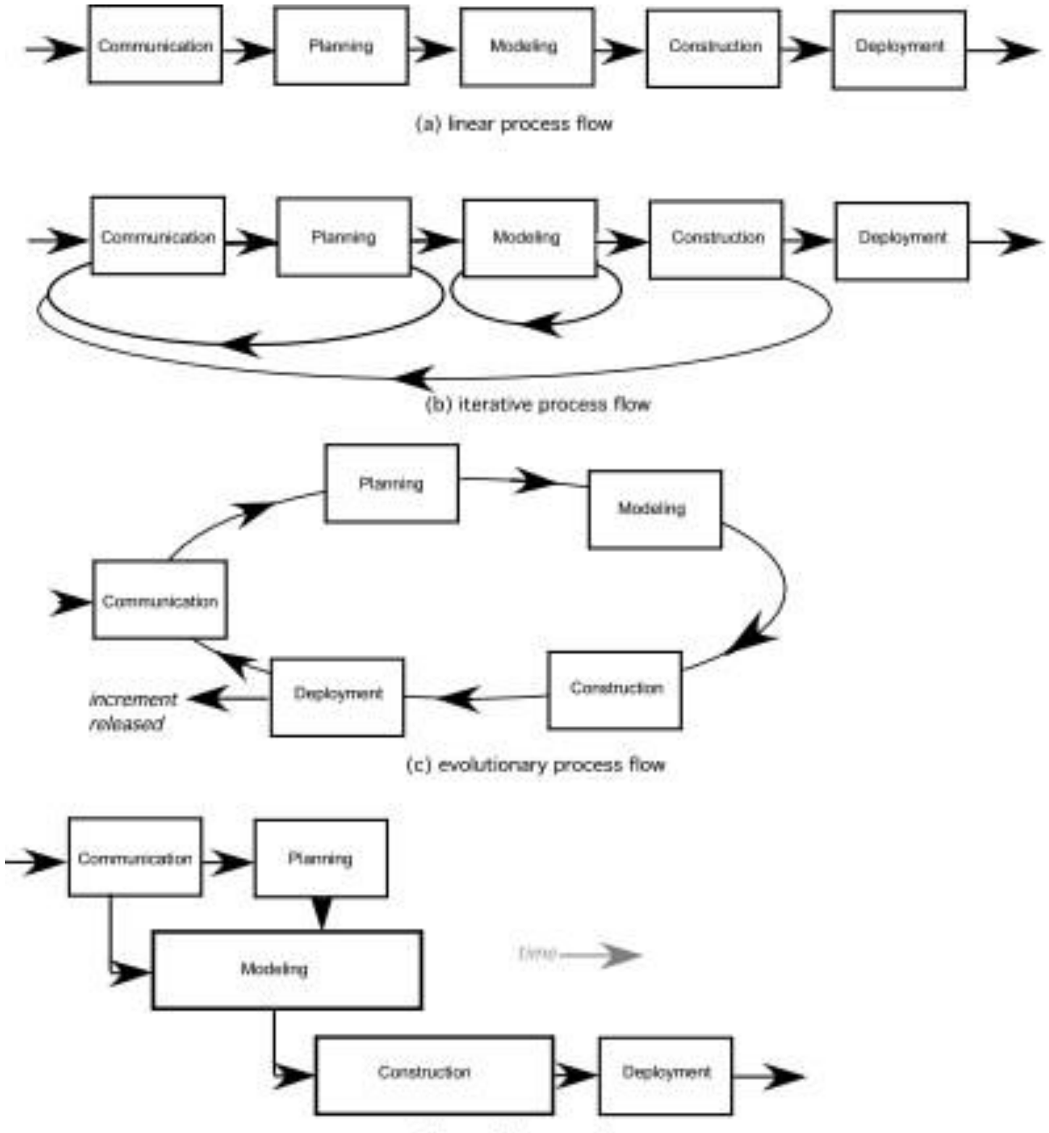


Process Flow

- We have some framework activities (do you still remember?), but how do we organize them?

- Based on **sequence** and **time**

Process flow: describes how the framework activities and the actions and tasks that occur within each framework activity are organized



(a) linear process flow

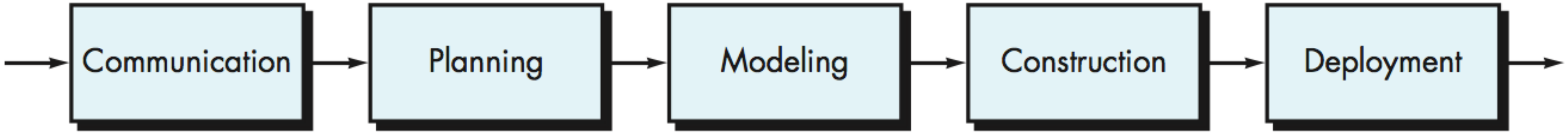
(b) iterative process flow

(c) evolutionary process flow

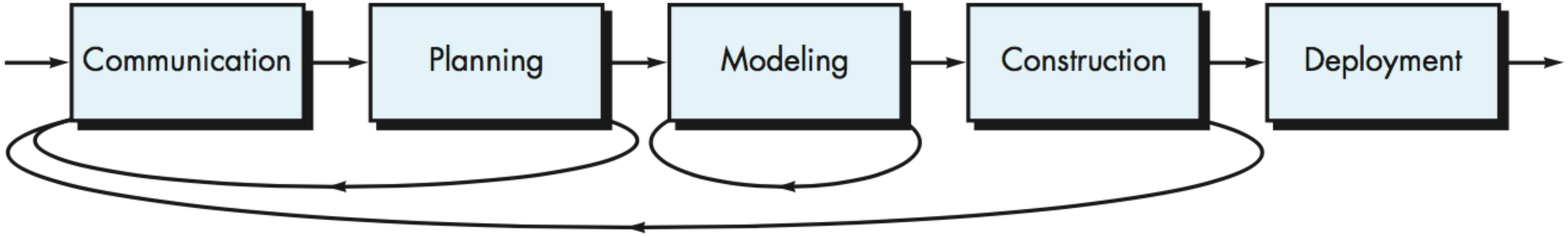
(d) parallel process flow

Process Flow

- A linear process flow executes each of the five framework activities *<u>in sequence</u>*, beginning with communication and ending with deployment.

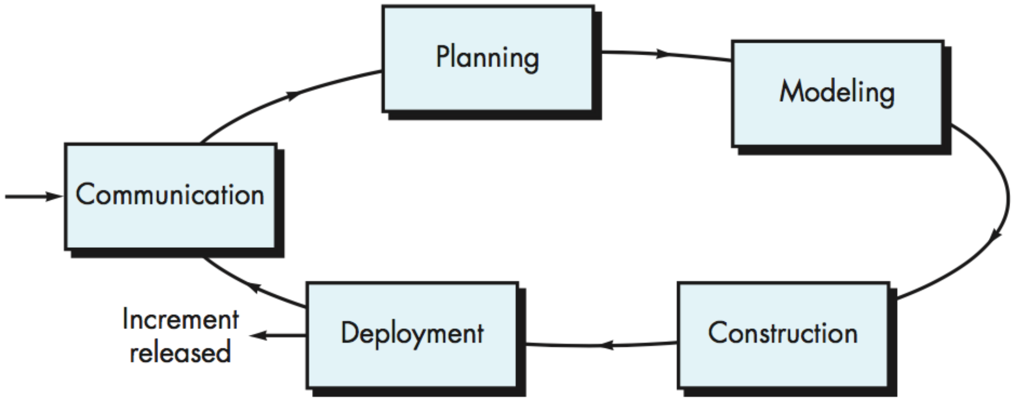


- An iterative process flow *<u>repeats one or more of the activities</u>* before proceeding to the next.

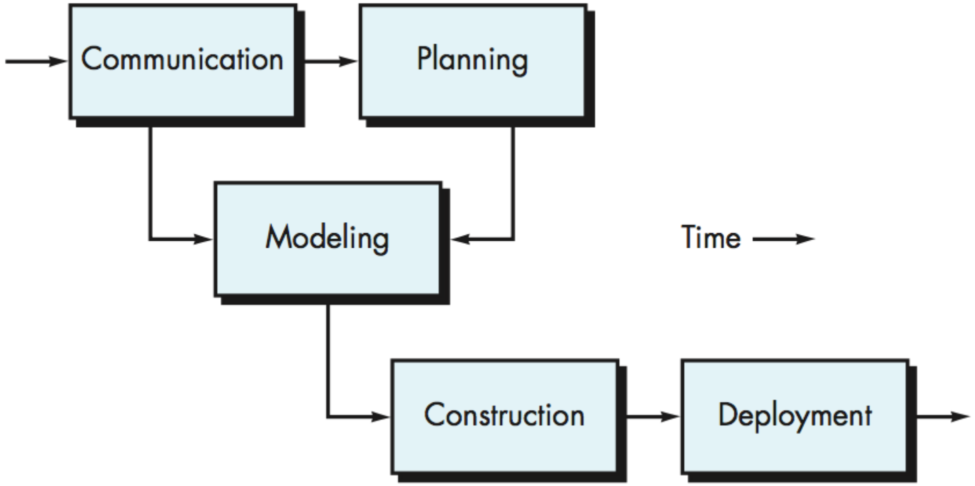


Process Flow

- An evolutionary process flow executes the activities *in a "circular" manner*. Each circuit through the five activities leads to a more complete version of the software.



- A parallel process flow *executes one or more activities in parallel with other activities*.(e.g., modeling for one aspect of the software might be executed in parallel with construction of another aspect of the software).

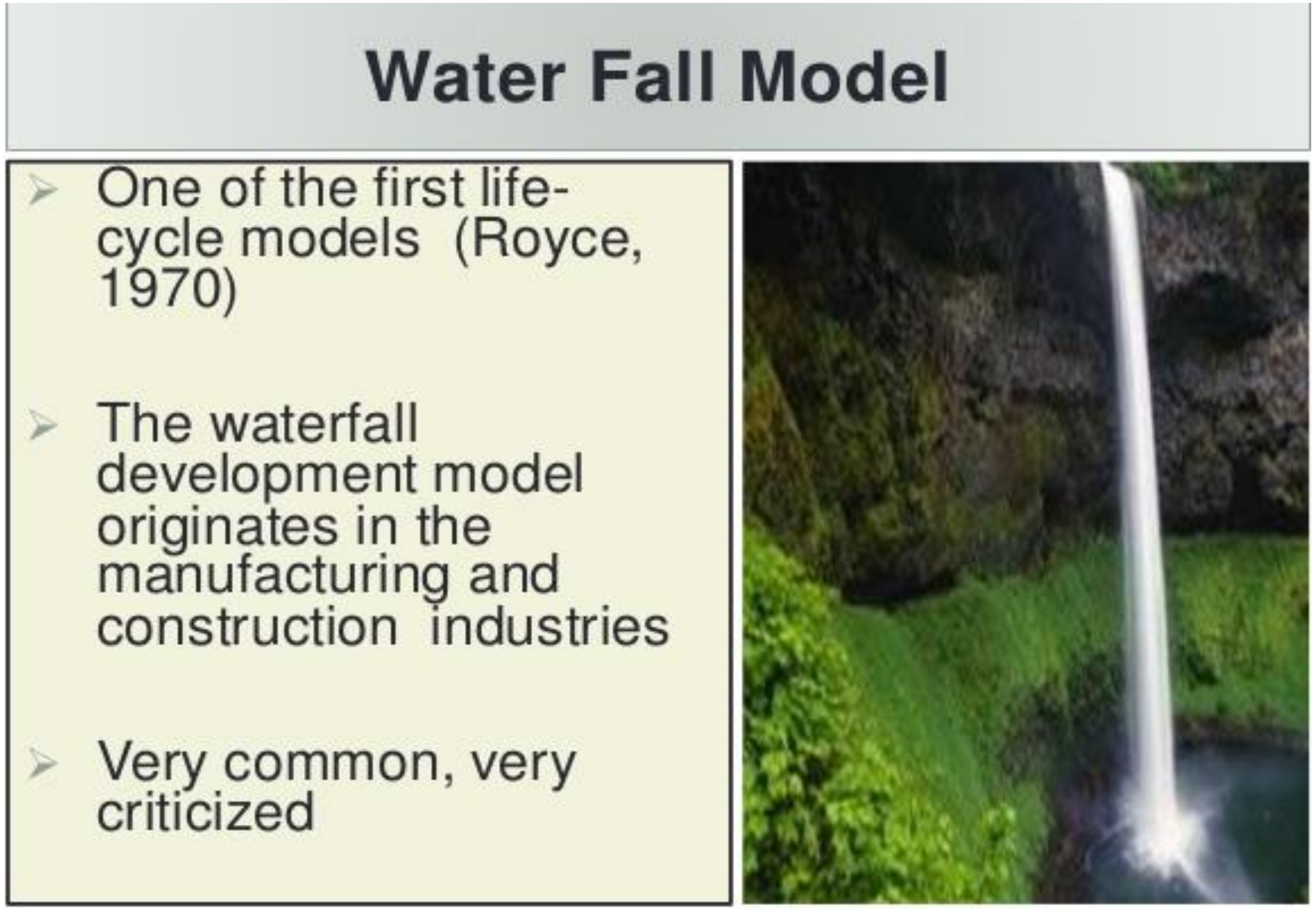


Prescriptive Models

- These models advocate an *orderly approach* to software engineering.

- Examples of such models are:
    - Waterfall model
    - Incremental/ Prototyping model
    - Evolutionary Process model

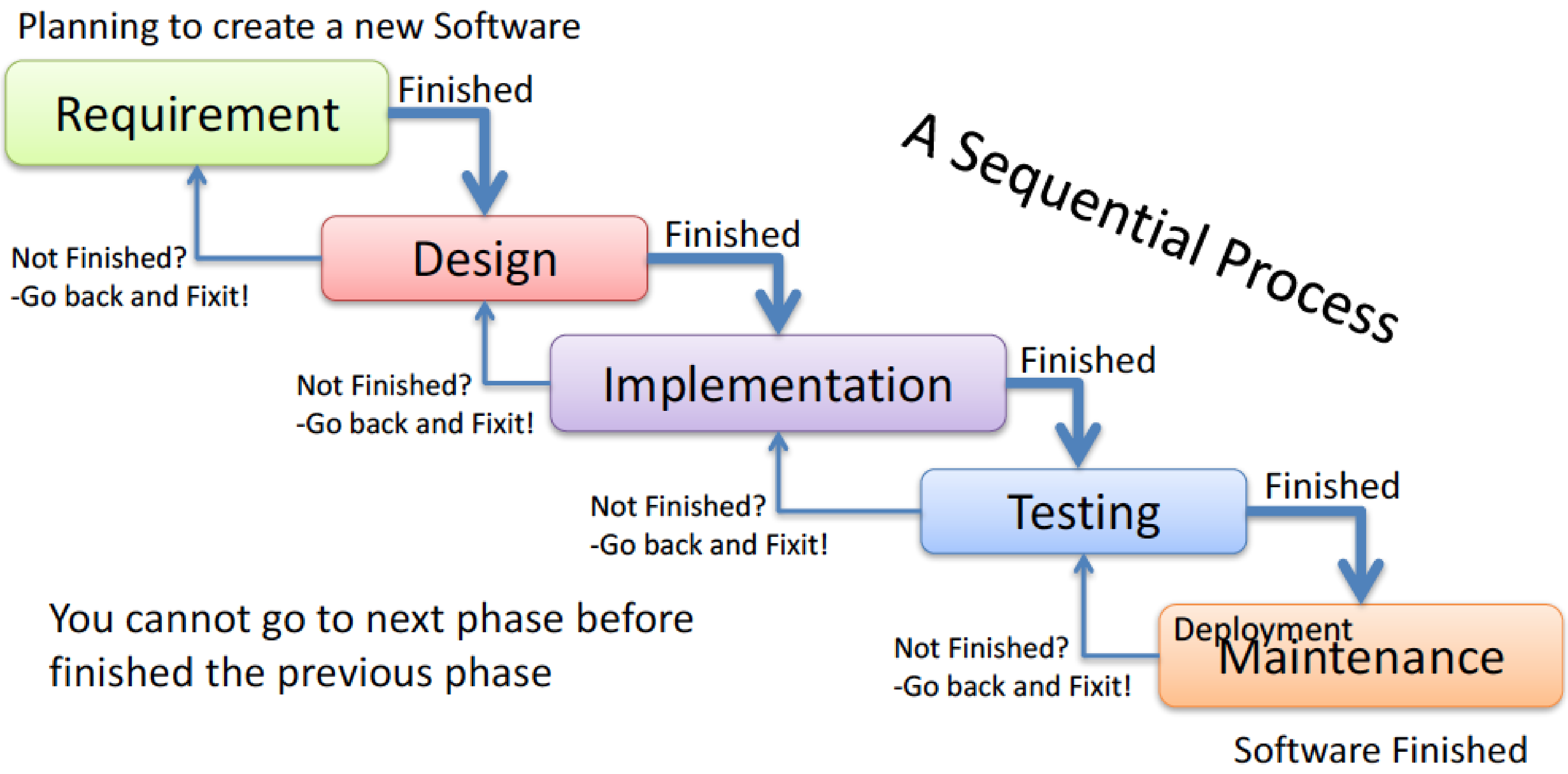
Waterfall Model



Water Fall Model

- One of the first life-cycle models (Royce, 1970)
- The waterfall development model originates in the manufacturing and construction industries
- Very common, very criticized

Waterfall Model



Planning to create a new Software

Requirement

Finished

Design

Not Finished?
-Go back and Fixit!

Finished

Implementation

Not Finished?
-Go back and Fixit!

Finished

Testing

Not Finished?
-Go back and Fixit!

Finished

Deployment

Maintenance

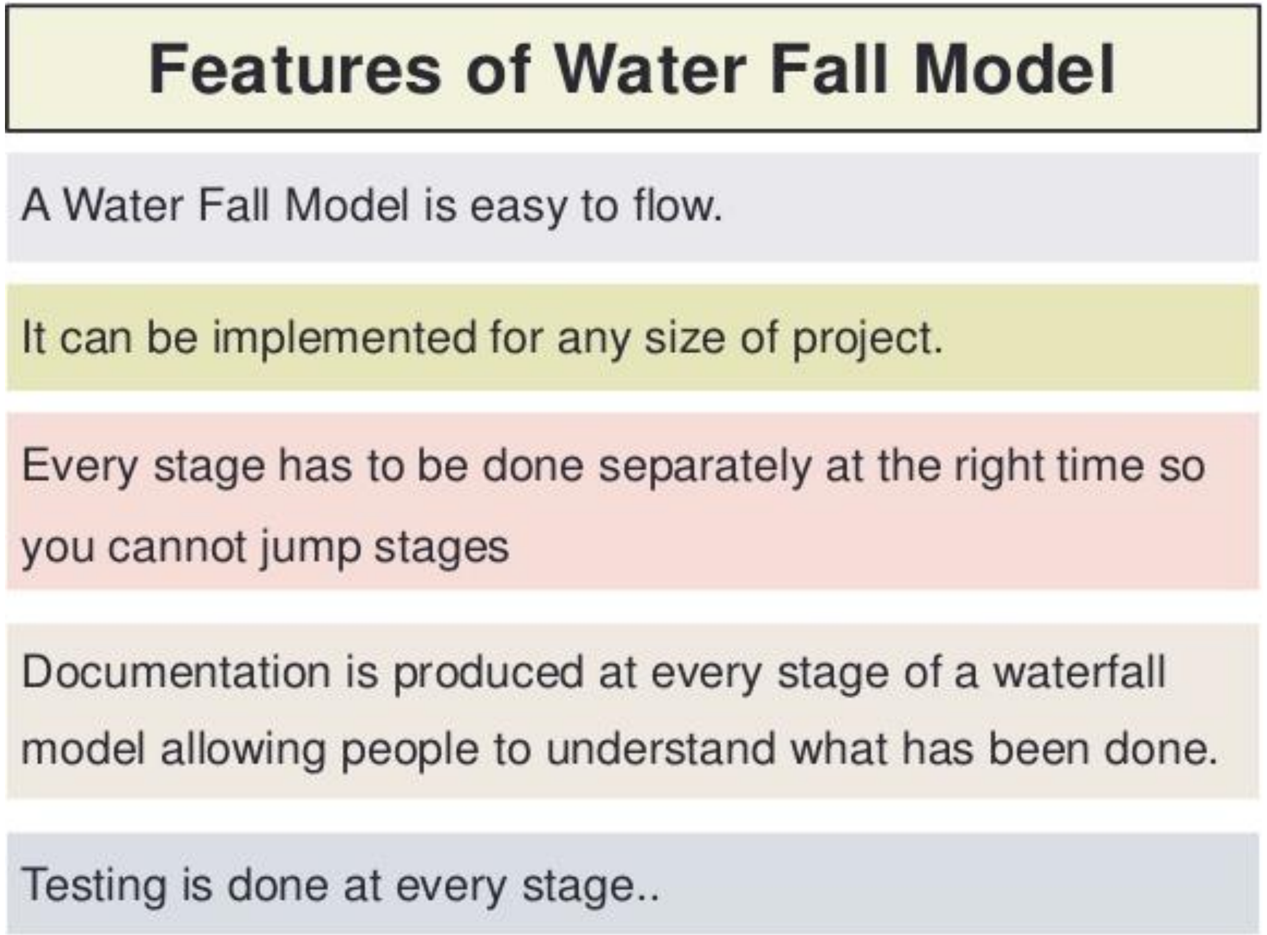
Not Finished?
-Go back and Fixit!

Software Finished

A Sequential Process

You cannot go to next phase before finished the previous phase

Waterfall Model



Features of Water Fall Model

A Water Fall Model is easy to flow.

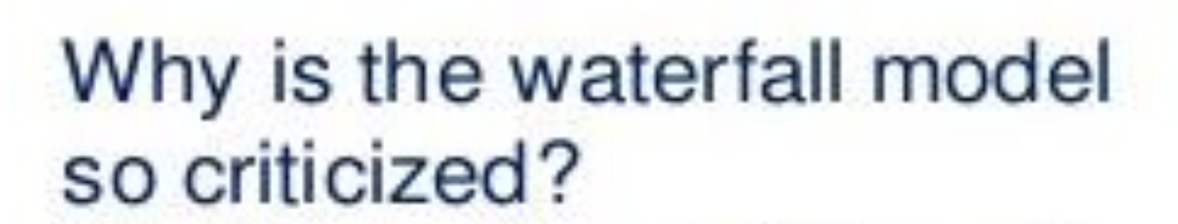
✓ It can be implemented for any size of project.

✓ Every stage has to be done separately at the right time so you cannot jump stages

Documentation is produced at every stage of a waterfall model allowing people to understand what has been done.

✓ Testing is done at every stage..

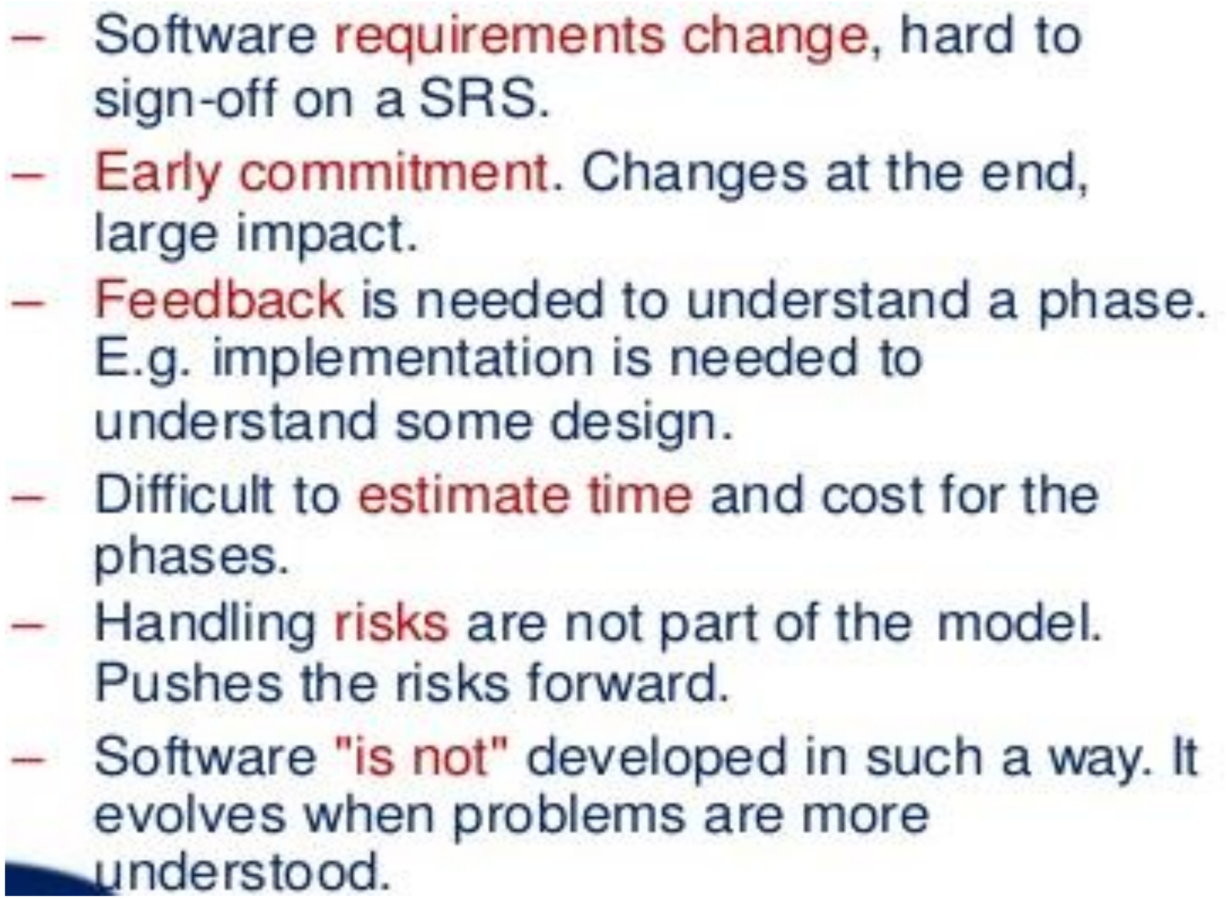
Waterfall Model



Why is the waterfall model so criticized?

Waterfall Model

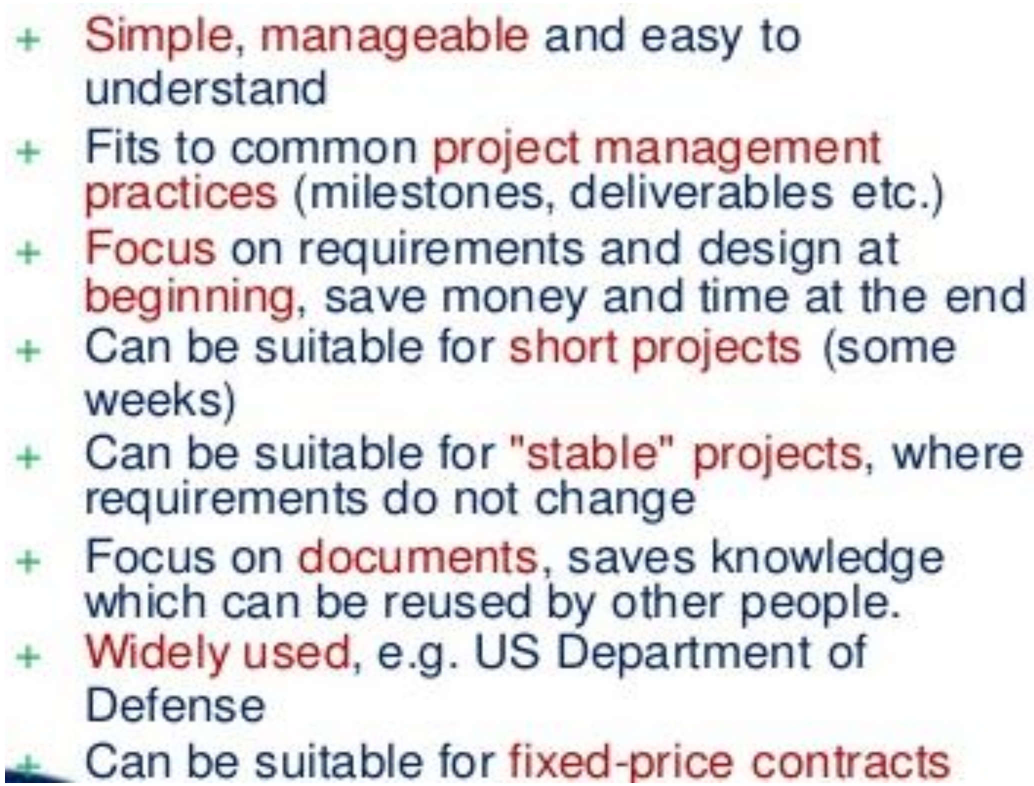
- The main drawback



- ✓ Software requirements change, hard to sign-off on a SRS.
- ✓ Early commitment. Changes at the end, large impact.
- ✓ Feedback is needed to understand a phase. E.g. implementation is needed to understand some design.
- Difficult to estimate time and cost for the phases.
- Handling risks are not part of the model. Pushes the risks forward.
- Software "is not" developed in such a way. It evolves when problems are more understood.

Waterfall Model

- The main Pros



✓ + Simple, manageable and easy to understand

+ Fits to common project management practices (milestones, deliverables etc.)

✓ + Focus on requirements and design at beginning, save money and time at the end

+ Can be suitable for short projects (some weeks)

✓ + Can be suitable for "stable" projects, where requirements do not change

+ Focus on documents, saves knowledge which can be reused by other people.

+ Widely used, e.g. US Department of Defense

✓ + Can be suitable for fixed-price contracts

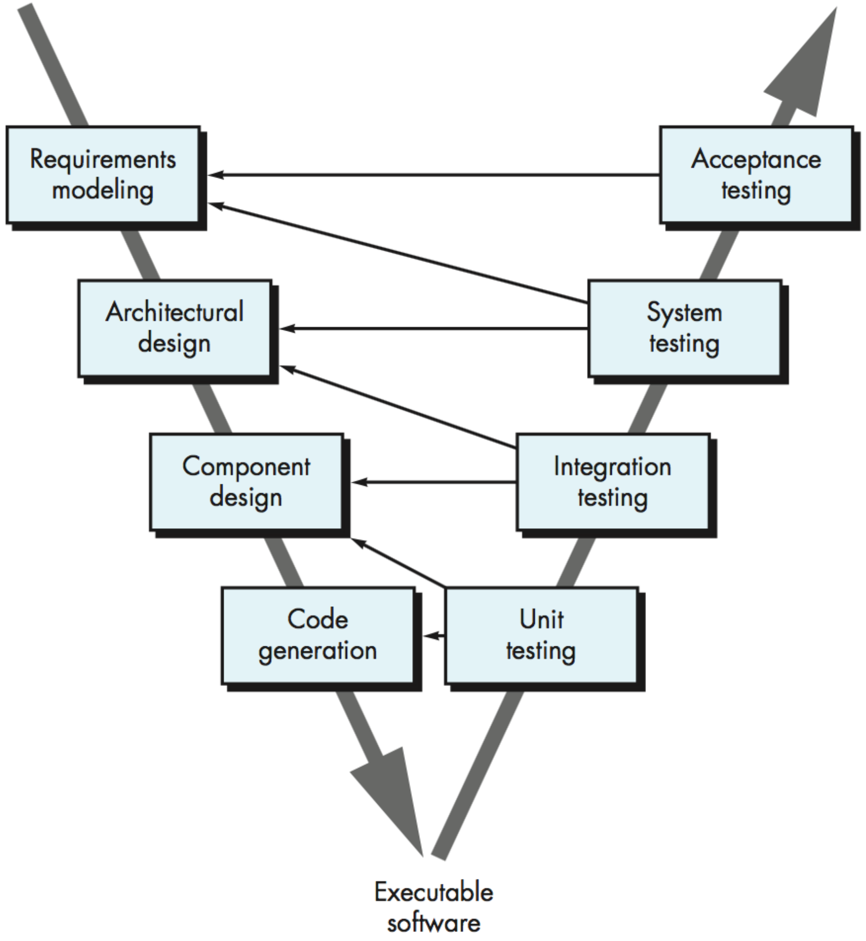
Waterfall Model

- When to use?
  - When the requirements are well-understood
  - When changes will be fairly limited during the design process.
    - Few business systems have stable requirements.
  - When large systems engineering projects are developed at several sites.
    - the waterfall model helps coordinate the work.

Mostly used in the development of software systems that have stringent **safety**, **reliability**, or **security** requirements

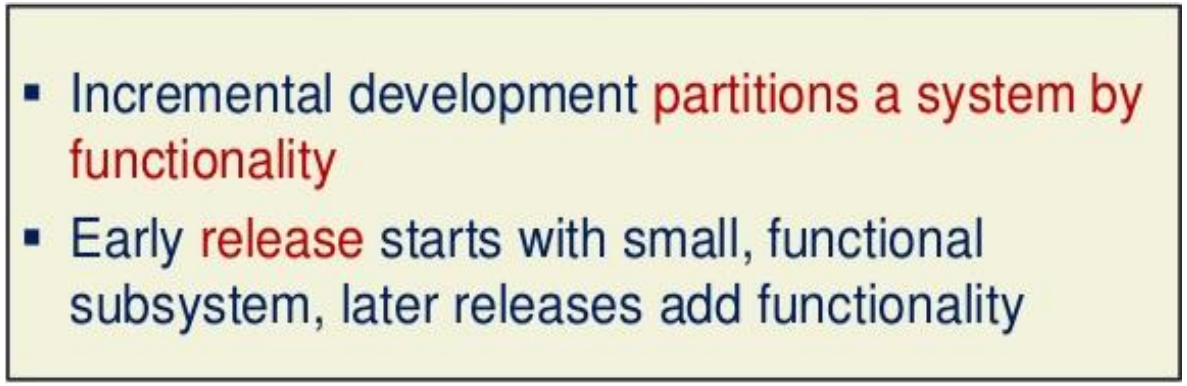
Waterfall Model – the V model

- A variation in the representation of the waterfall model
- Visualizes how verification and validation actions are applied to earlier engineering work.
- Depicts the relationship of quality assurance actions to the actions associated with communication, modeling, and early construction activities

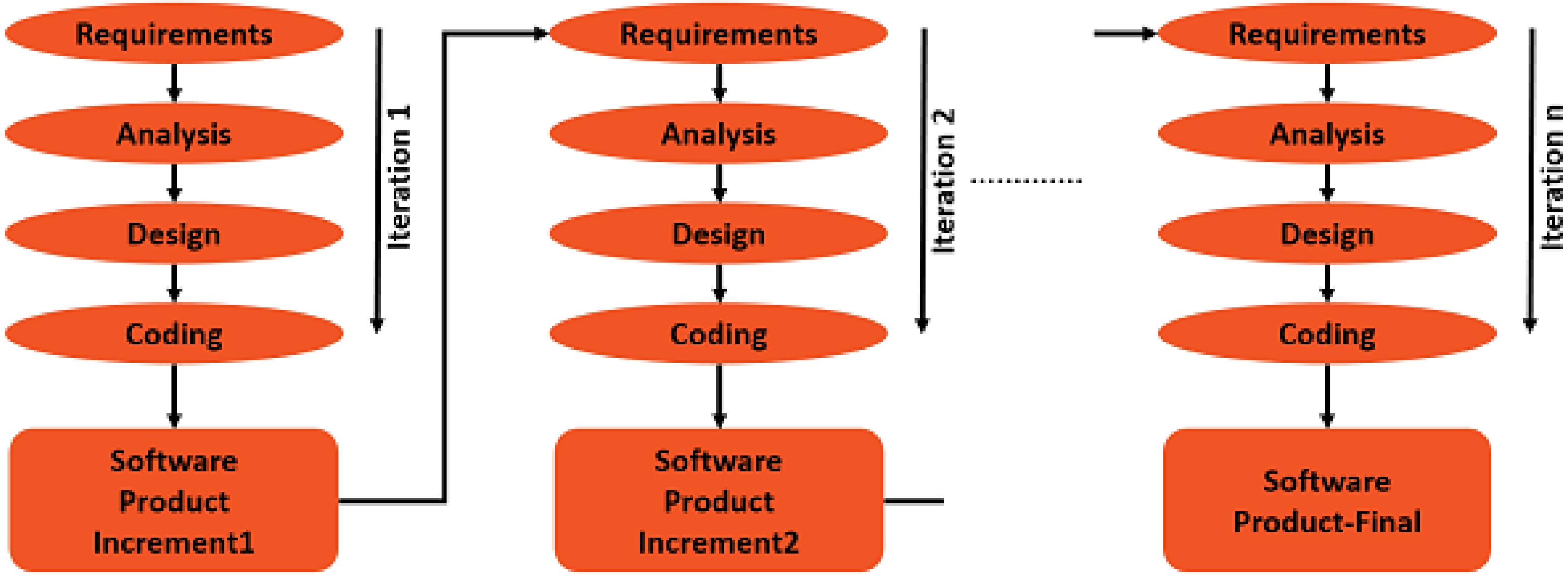


Incremental Development

-

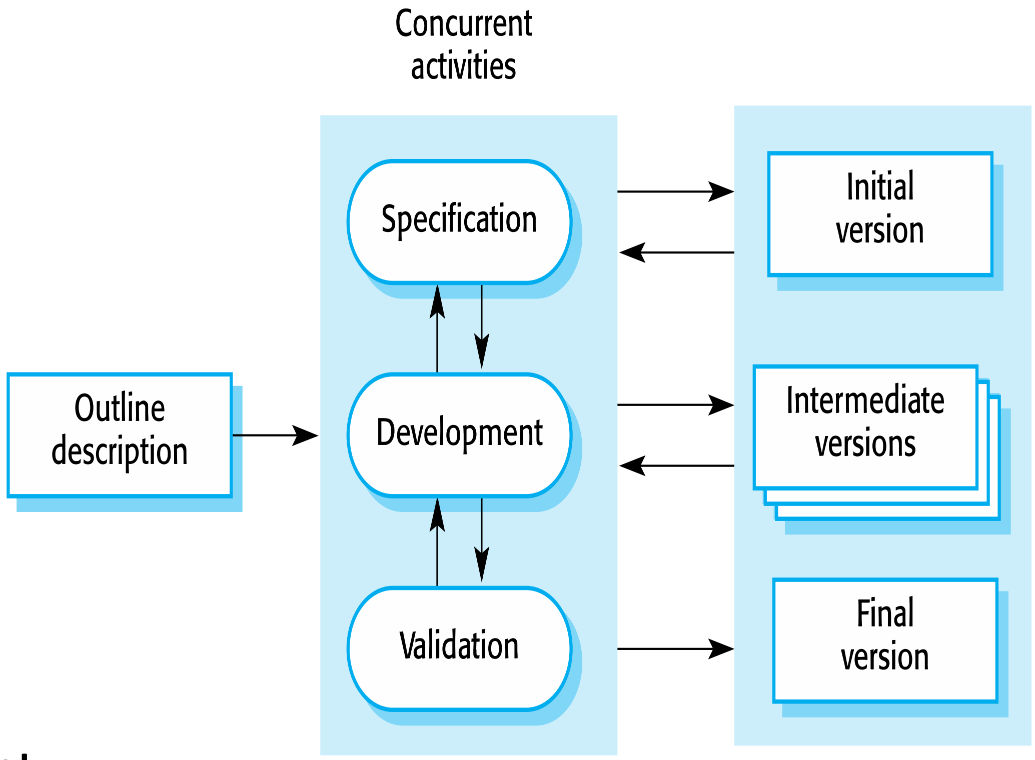


- Incremental development partitions a system by functionality
- Early release starts with small, functional subsystem, later releases add functionality



Incremental Development

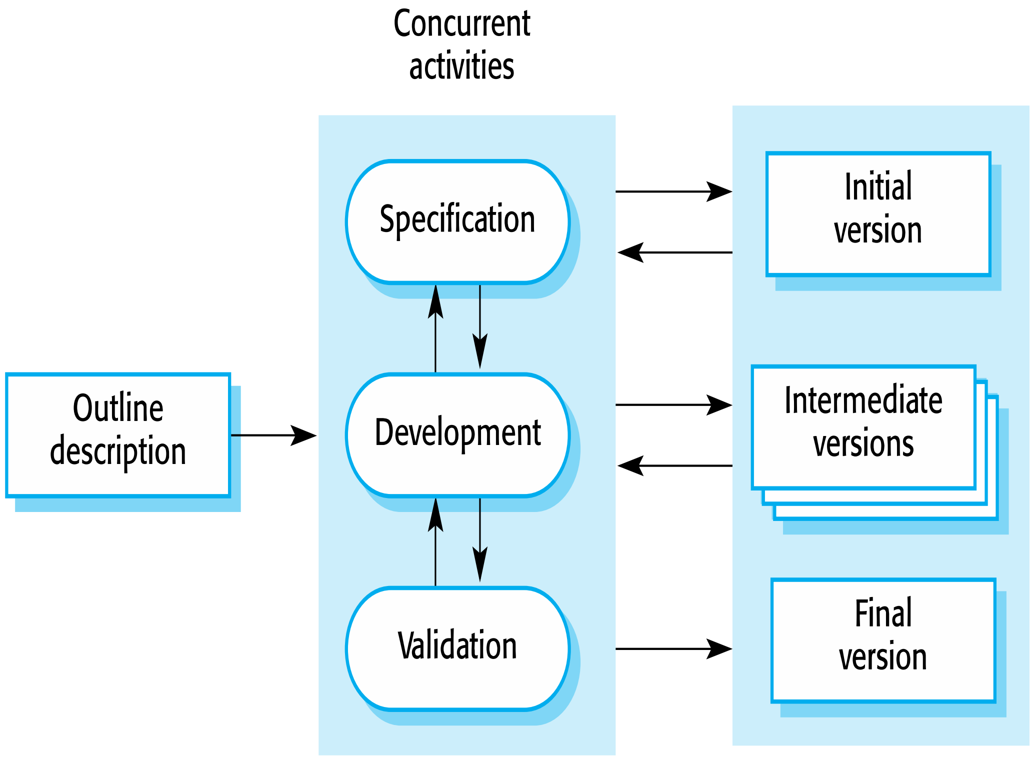
- Incremental development is based on the idea of
  - developing an **initial implementation**,
  - exposing this to **user feedback**,
  - **evolving it through several versions** until an acceptable system has been developed.
- The **activities** of a process
  - **not separated**
  - **but interleaved** with feedback involved across those activities.



Incremental Development

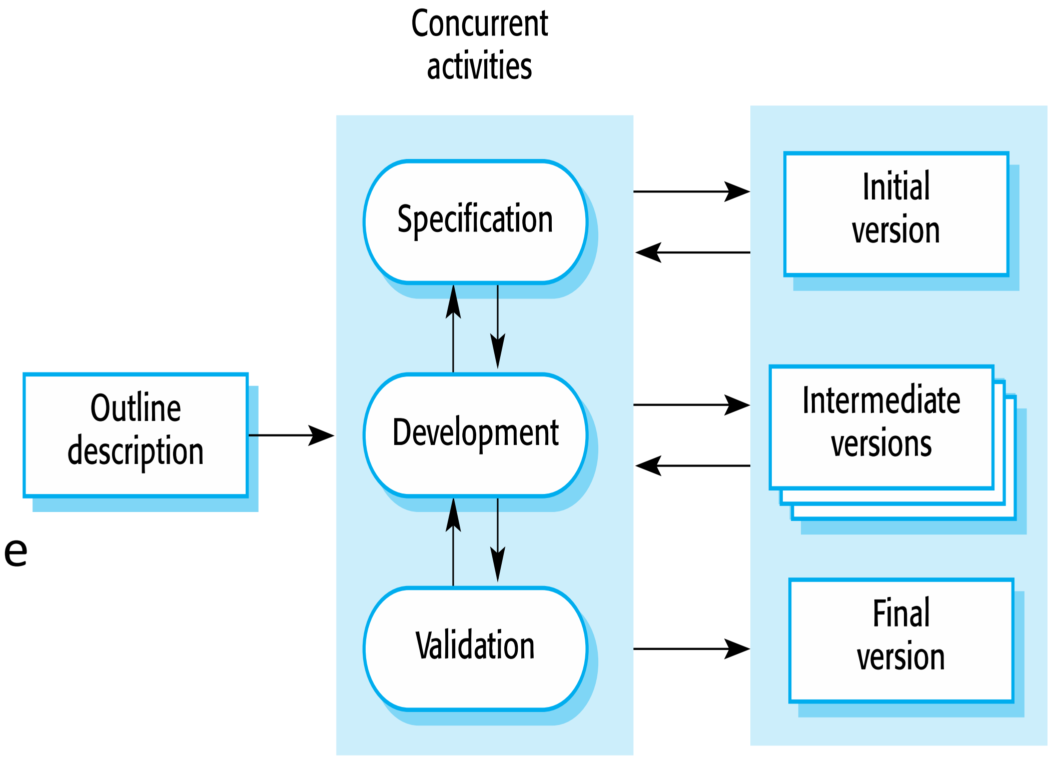
- **Each system increment** reflects
  - a **needed piece of the functionality**
  - the early increments of the system
    - should include the **most important or most urgently** required **functionality**.

- **Customer can evaluate the system** at early stage in the development
  - see if it delivers what’s required
  - If not,
    - only the current increment has to be changed,
    - and possibly, new functionality defined for later increments.



Incremental Development

- **It can be a plan-driven or agile, or both**
    - In **a plan-driven** approach,
        - the **system increments are identified in advance**
    - in the **agile** approach,
        - **only the early increments are identified**
        - the development of later increments depends on the progress and customer priorities.



Incremental Development: Benefits

1. The cost of **implementing requirements changes** is reduced.

2. Amount of documentation and analysis is reduced to a great extent.

3. It is easier to **get customer feedback** on the development work that has been done.

4. **Early delivery and deployment** of useful software to the customer is possible, even if all of the functionality has not been included.

Incremental Development: Problems

1. The process is not visible. Managers need regular deliverables to **measure progress**.

   - If systems are developed quickly, it is not cost effective to produce documents that reflect every version of the system.

Incremental Development: Problems

2. System **structure tends to degrade** as new increments are added.

    - **Regular change** leads to messy code as new functionality is added.
        - It becomes increasingly **difficult and costly to add new features** to a system.
    - To **reduce structural degradation**,
        - Agile methods suggest that you should regularly **refactor** (improve and restructure) the software.

Incremental Development

- When to use?
  - When requirements are likely to change during the development process
    - This is the case for most business systems and software products.
  - When systems is not large, complex, long-lifetime systems, where different teams develop different parts of the system.

Evolutionary Process Models

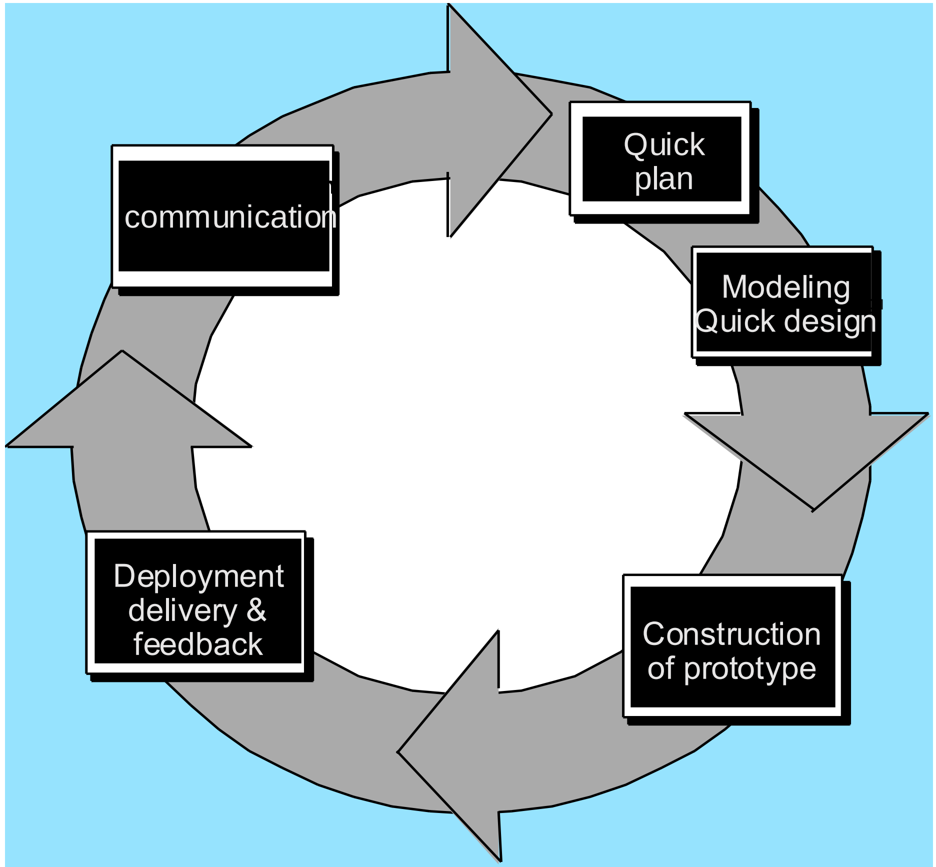
- A straight path to an end product is unrealistic.

- These models accommodates for changes in software/ product requirements over time.

- A limited version that meets the competitive or business pressure is introduced.

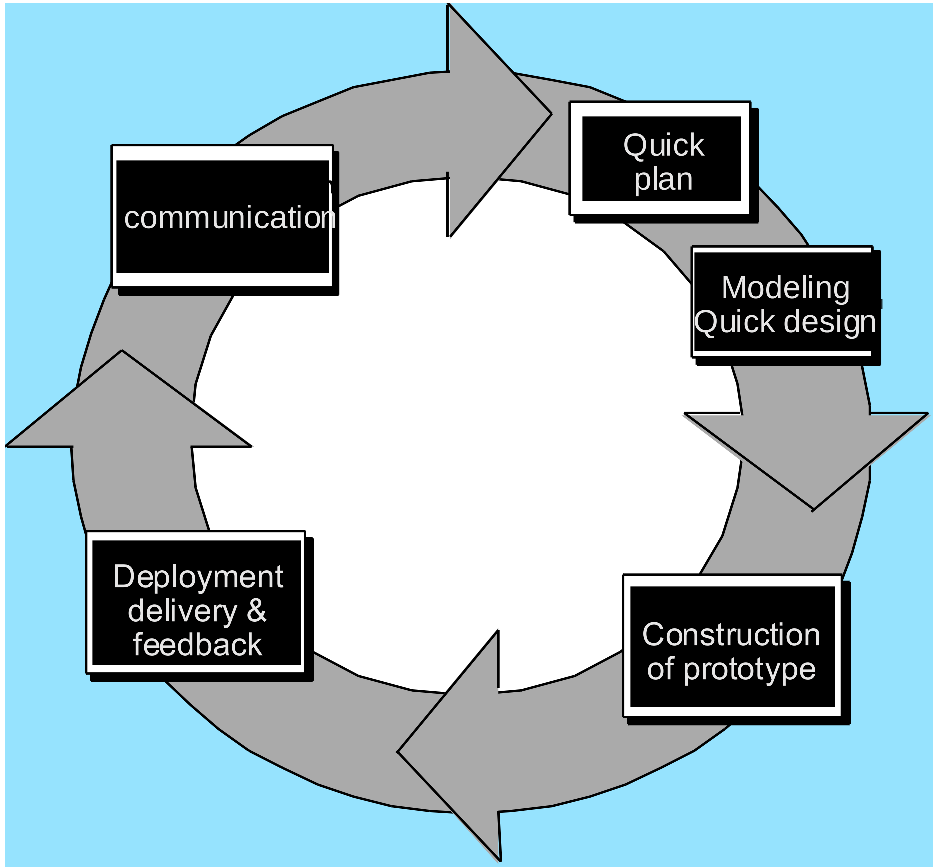
Evolutionary Models: Prototyping

- What if:
- A customer defines a set of general objectives for software, but does **not** identify detailed requirements
- The developer is **unsure** of the efficiency of an algorithm, the adaptability of an operating system, or the form that human-machine interaction should take



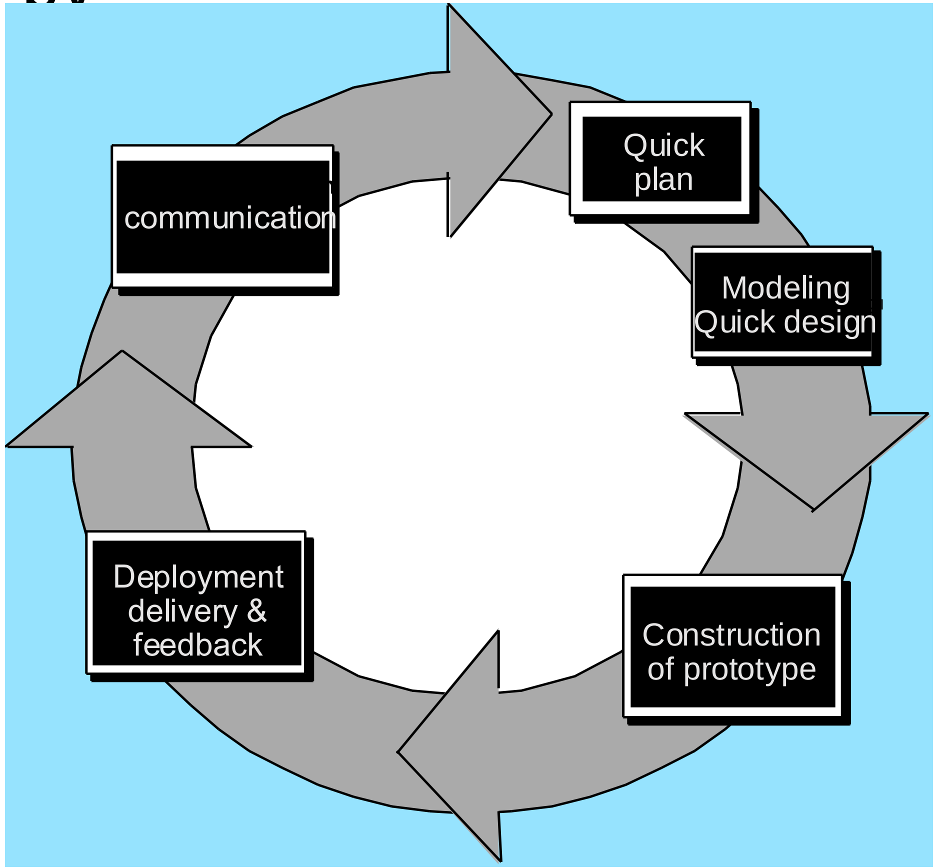
Evolutionary Models: Prototyping

- Here, you first define the general objectives for the software after meeting the stakeholders
- You then identify the known requirements and start planning the prototype.
- A quick design focuses on a representation of those aspects of the software that will be **visible** to end users (e.g., human interface layout or output display formats)
- The quick design leads to the construction of a **prototype**



Evolutionary Models: Prototyping

- You then release the prototype to be evaluated by stakeholders.
- Based on the feedback from stakeholder, the prototype is tuned.
- This continues until the complete product is developed.
- Some of these prototypes can be completed discarded if they don’t meet the requirements. These are called “throwaways”.



Benefits OF Prototyping

- Improved system usability.
- A closer match to users’ real needs.
- Improved design quality.
- Improved maintainability.
- Reduced development effort.

Problems of Prototyping

- Stakeholders see a working version, unaware that it is still a prototype with lower quality, insufficient testing, low maintainability.

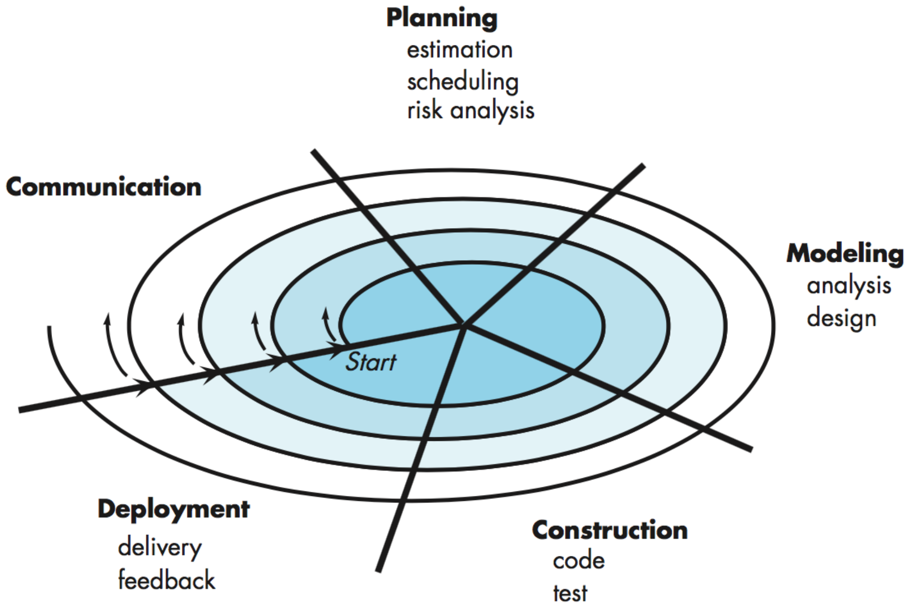
- Stakeholders ask for fixes on prototypes instead of rebuilding a high quality system.

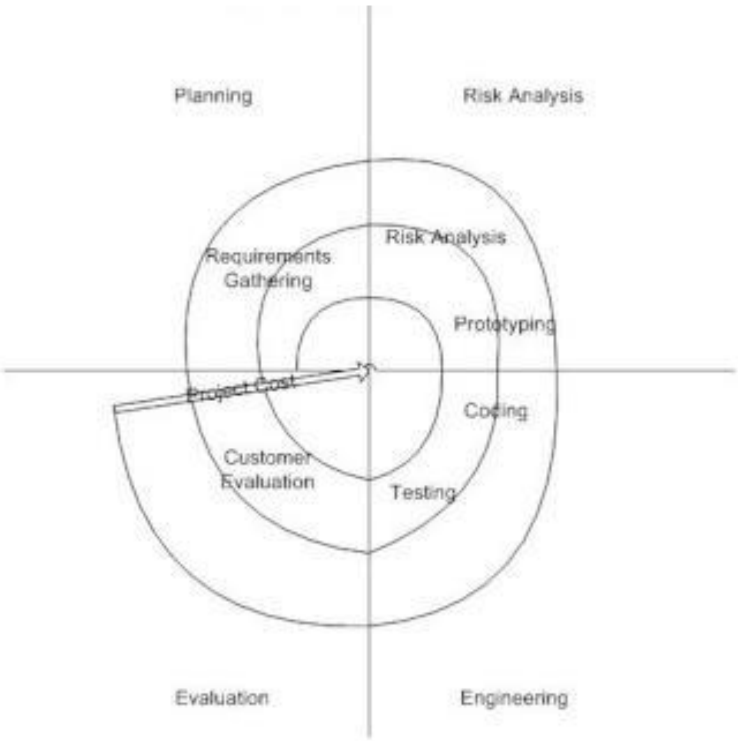
- Software engineers often make implementation compromises just for getting a prototype working quickly
    - Inappropriate programming language
    - Inefficient algorithm
    - Make many less-than-ideal choice

Key of Prototyping

- Define the rules of the game at the beginning
- All stakeholders should agree that *the* ***prototype is built to serve as a mechanism for defining requirements***.
- It is then discarded (at least in part), and the actual software is engineered with an eye toward quality.

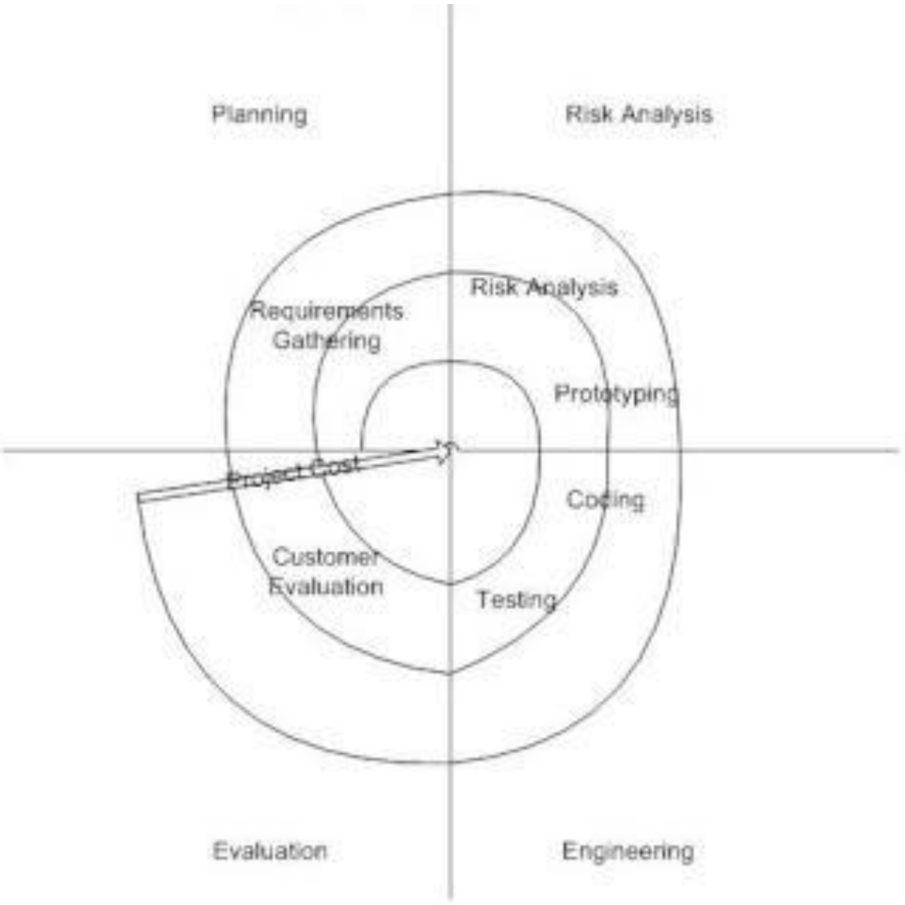
Evolutionary Models: The Spiral





The *spiral model* is an evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model.

Evolutionary Models: The Spiral



Planning Phase: Requirements are gathered

Risk Analysis: Identify risk and alternate solutions. A prototype is produced at the end of the risk analysis phase. If any risk is found during the risk analysis then alternate solutions are suggested and implemented.

Engineering Phase: software is developed, along with testing at the end of the phase. Hence in this phase the development and testing is done.

Evaluation phase: This phase allows the customer to evaluate the output of the project to date before the project continues to the next spiral.

Evolutionary Models: The Spiral

Advantages of Spiral model:

- High amount of risk analysis hence, avoidance of risk is enhanced.
- Good for large and mission-critical projects.
- Strong approval and documentation control.
- Additional Functionality can be added at a later date.
- Software is produced early in the software life cycle.
- Changing requirements can be accommodated.

Evolutionary Models: The Spiral

Disadvantages of Spiral model:

- Can be a costly model to use.
- Risk analysis requires highly specific expertise.
- Project’s success is highly dependent on the risk analysis phase.
- Doesn’t work well for small projects or low risk projects
- Large number of intermediate stages requires excessive documentation

Evolutionary Models: The Spiral

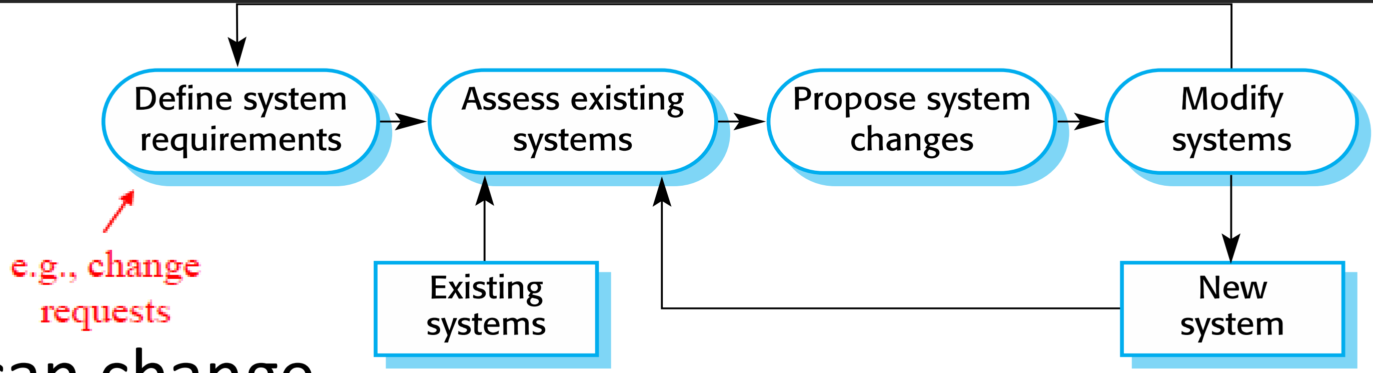
When to use Spiral model

- When costs and risk evaluation is important
- For medium to high-risk projects
- For large projects
- Long-term project commitment unwise because of potential changes to economic priorities
- Users are unsure of their needs
- Requirements are complex
- New product line
- Significant changes are expected
- When releases are required to be frequent

Component-Based Development

- Based on systematic reuse where systems are integrated from existing components or COTS (Commercial-off-the-shelf) systems.
  - Available component-based products are researched and evaluated for the application domain in question.
  - Component integration issues are considered.
  - A software architecture is designed to accommodate the components.
  - Components are integrated into the architecture.
  - Comprehensive testing is conducted to ensure proper functionality.
  - leads to software reuse
  - Save development cycle time
  - Save project cost

Software Evolution (maintenance)



- Software is inherently flexible and can change.

- As requirements change through changing business circumstances,
    - the software that supports the business must also evolve and change.

- Although there has been a demarcation between development and evolution
    - this is increasingly irrelevant as fewer and fewer systems are completely new.

Coping with Change

- Change is inevitable in all large software projects?
  - Business changes
    - lead to new and changed system requirements
  - New technologies
    - open up new possibilities for improving implementations
  - Changing platforms
    - require application changes
- Change leads to rework so the costs of change include both
  - Re-analyzing requirements
  - The costs of implementing new functionality

Reducing the Costs of Rework

Change anticipation

- includes process activities that can anticipate possible
  - user/stakeholder changes before development begins.
  - E.G., a prototype
    - may allow users/stakeholders to better envision how the system would actually be used.

SOLUTION

- System prototyping:
  - where a version of the system or part of the system is developed quickly to check
    - the customer’s requirements and
    - the feasibility of design decisions.

Reducing the Costs of Rework

Change tolerance

- The process is designed so that changes can be accommodated at relatively low cost.

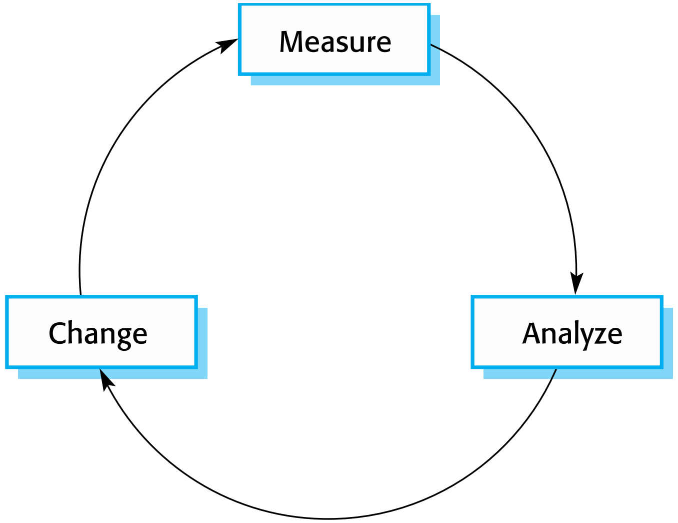
- This normally involves some form of incremental development.
  - Proposed changes may be implemented in increments that have not yet been developed.

  - If this is impossible, then
    - only a single increment (a small part of the system) may be altered to incorporate the change.

Process Improvement

- Many software companies have turned to software process improvement as a way of
  - enhancing the quality of their software,
  - reducing costs or accelerating their development processes.
- Process improvement means
  - understanding existing processes and
  - changing these processes
  - to increase product quality and/or reduce costs and development time.

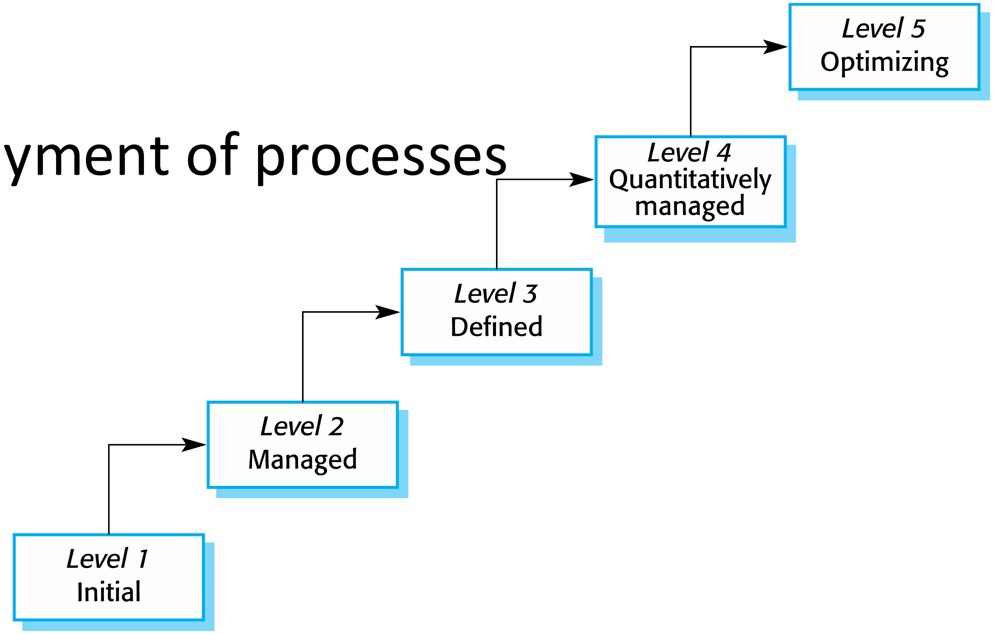
Approaches to Improvement



- The process maturity approach, focuses on
  - improving process and project management
  - introducing good software engineering practice.
    - The level of process maturity reflects the extent to which
      - good technical and management practice has been adopted in organizational software development processes.

The SEI capability maturity model

- Initial
  - For each process a scope of work is defined
- Managed
  - Product management procedures defined and used
- Defined
  - focus on organizational standardization and deployment of processes
- Quantitatively Managed
  - Quality management strategies defined and used
- Optimising
  - Process improvement strategies defined and used



Approaches to Improvement

- The agile approach, focuses on
  - iterative development and the reduction of overheads in the software process.
  - The primary characteristics of agile methods are
    - rapid delivery of functionality
    - responsiveness to changing customer requirements.