EAST WEST UNIVERSITY

CSE412
Software Engineering

Nishat Tasnim Niloy

Lecturer

Department of Computer Science and Engineering

Faculty of Science and Engineering

Topic 1

Software Development Process

Adapted from *Software Engineering: A Practitioner's Approach* by Roger Pressman, 7th Edition, Chapter two

Software Engineering

- Software Engineering is the application of engineering to software. This unit looks at its goals and principles.
- It is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software, and the study of these approaches. In other words, it is the application of engineering to software.

Sub-disciplines of Software engineering

Software engineering can be divided into several sub-disciplines. Such as-

- **Software requirements:** The elicitation, analysis, specification, and validation of requirements for software.
- **Software design:** Software Design consists of the steps a programmer should do before they start coding the program in a specific language. It is usually done with Computer-Aided Software Engineering (CASE) tools and use standards for the format, such as the Unified Modeling Language (UML).
- **Software development:** It is construction of software using programming languages.
- **Software testing:** Software Testing is an empirical investigation conducted to provide stakeholders with information about the quality of the product or service under test.
- **Software maintenance:** This deals with enhancements of Software systems to solve the problems they may have after being used for a long time after they are first completed..
- **Software configuration management:** It is the task of tracking and controlling changes in the software. Configuration management practices include revision control and the establishment of baselines.
- **Software Quality Assurance:** The totality of functionality and features of a software product that bear on its ability to satisfy stated or implied needs.

Software Engineering Goals

- Maintainability
- Reliability
- Efficiency
- Comprehensibility

Software Development Life Cycle

- Software life cycle models describe phases of the software cycle and the order in which those phases are executed. There are a lot of models, and many companies adopt their own, but all have very similar patterns.

- The general model:

Plan Driven Model

- It is a process where all the activities are planned in advance, and we measure the progress in advance against that plan.
- It requires knowledgeable personnel at the beginning.
- It is appropriate for big software development or extensive systems and sides.
- It is suitable for a stable growth environment.
- This model can not lodge changes any time
- There is a lack of user participation through the life cycle of the creation.
- It is expensive for the dynamic expansion environment.

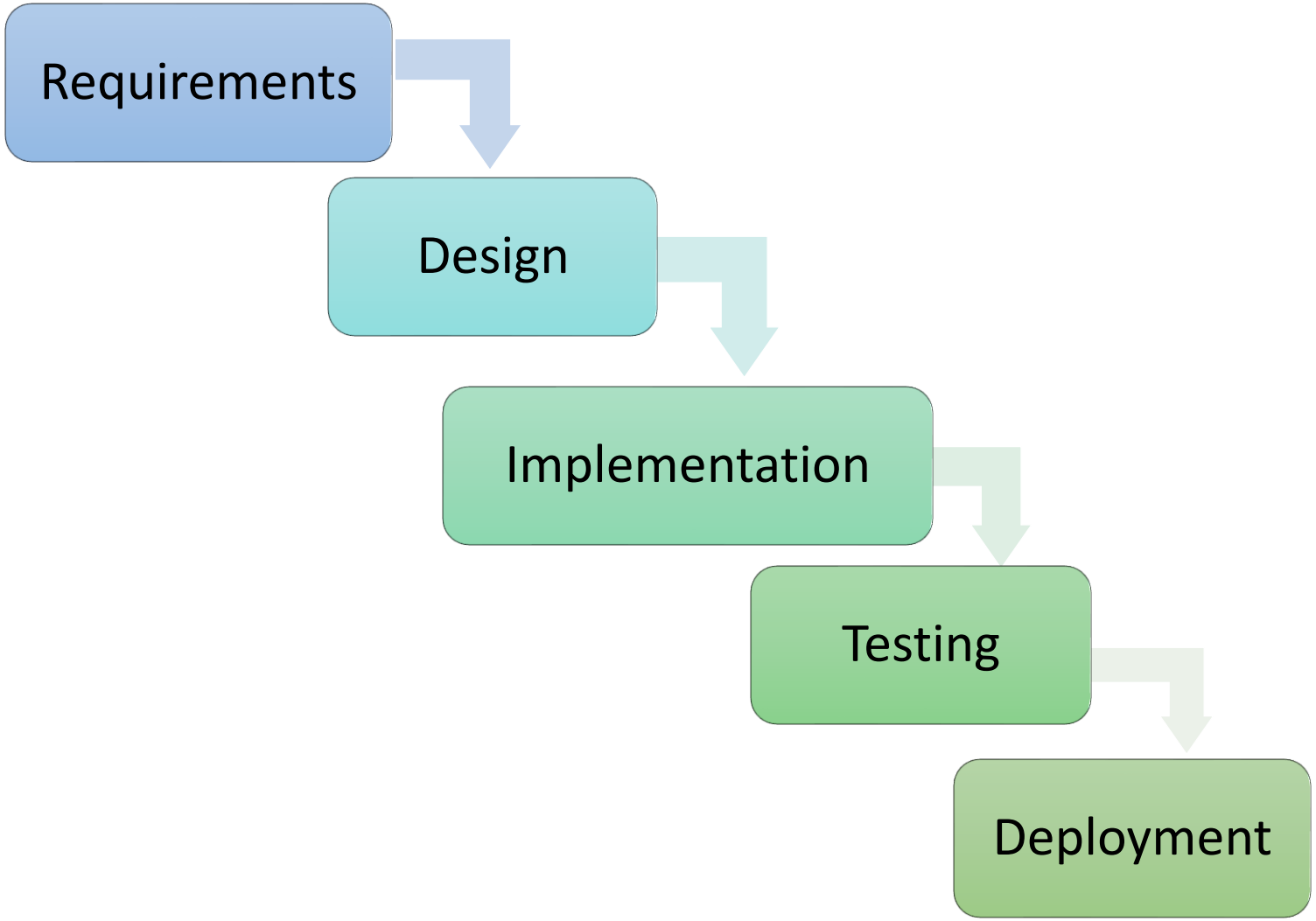
Popular Software Development Models

- Waterfall model
- Incremental Development
- Prototype Model
- Iterative development
- Rapid Application Development
- Spiral Development

Waterfall Model

- This is the most common life cycle models, also referred to as a linear-sequential life cycle model.
- It is very simple to understand and use.
- In a waterfall model, each phase must be completed before the next phase can begin.
- At the end of each phase, there is always a review to ascertain if the project is in the right direction and whether to carry on or abandon the project.
- Unlike the general model, phases do not overlap in waterfall model.

Waterfall Model



Waterfall Model

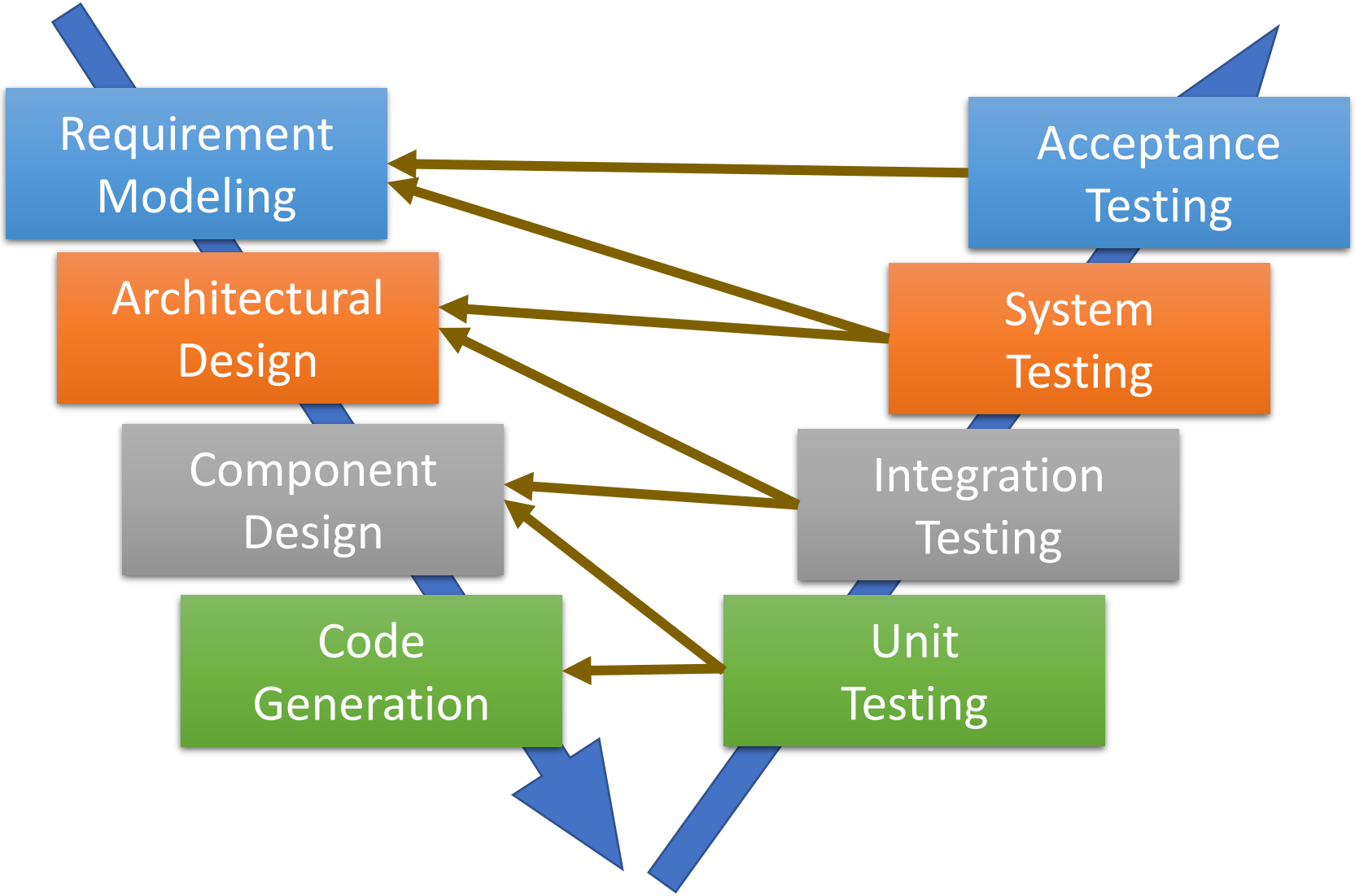
Advantages

- Simple and easy to use.
- Easy to manage due to the rigidity of the model – each phase has specific deliverables and a review process.
- Phases are processed and completed one at a time.
- Works well for smaller projects where requirements are very well understood.

Disadvantages

- Adjusting scope during the life cycle can kill a project
- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Poor model for complex, on-going, object-oriented projects, where requirements are at a moderate to high risk of changing

V-Shaped Model



V-Shaped Model

Advantages

- Simple and easy to use.
- Each phase has specific deliverables.
- Higher chance of success over the waterfall model due to the development of test plans early on during the life cycle.
- Works well for small projects where requirements are easily understood.

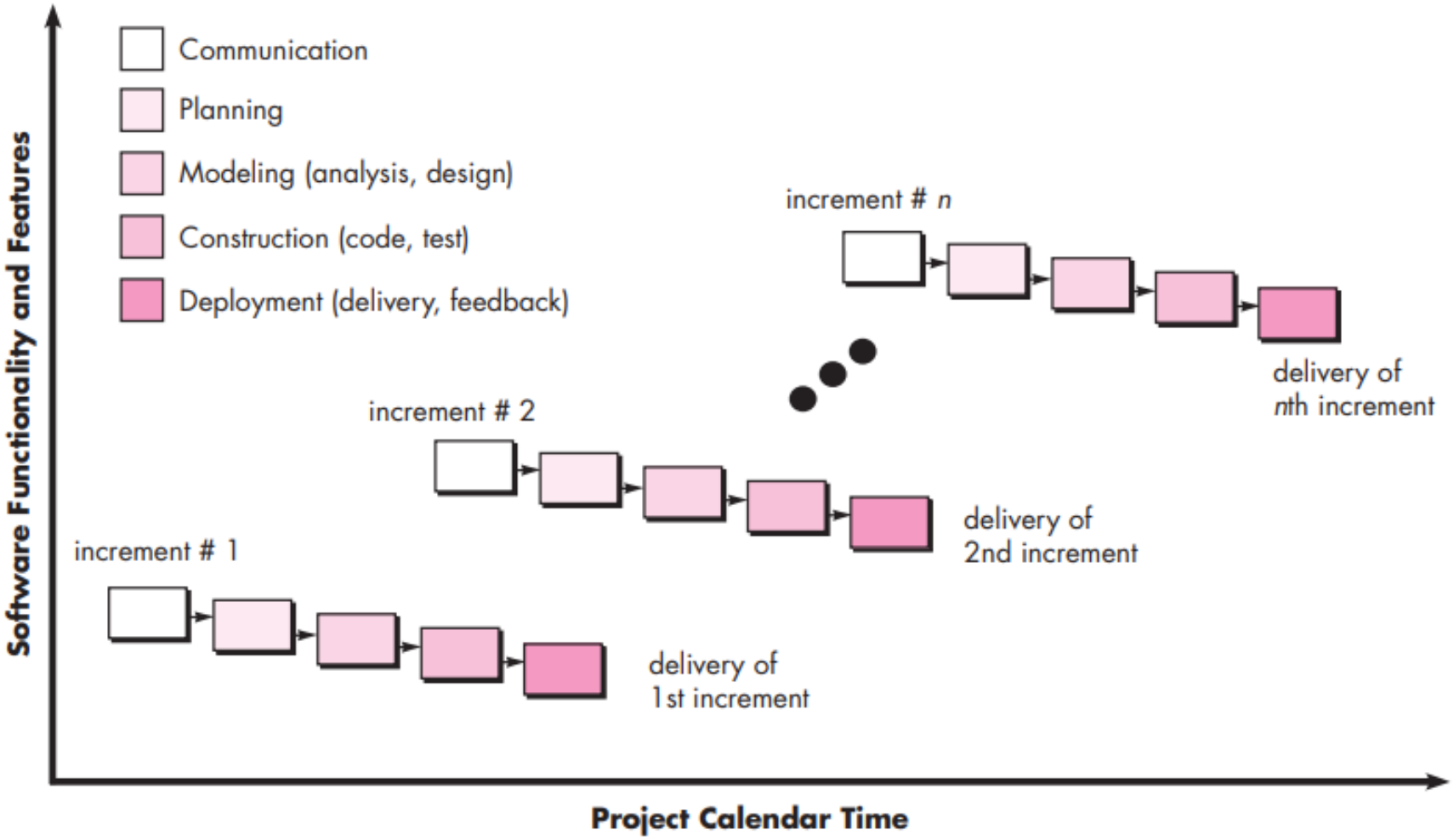
Disadvantages

- Very rigid, like the waterfall model.
- Little flexibility and adjusting scope is difficult and expensive.
- Software is developed during the implementation phase, so no early prototypes of the software are produced.
- Model doesn‘t provide a clear path for problems discovered during testing phases.

Incremental Model

- The incremental model is an intuitive approach to the waterfall model. It is a kind of a multi-waterfall cycle.
- In that multiple development cycles take at this point. Cycles are broken into smaller, more easily managed iterations. Each of the iterations goes through the requirements, design, implementation and testing phases.
- The first iteration produces a working version of software, and this makes possible to have working software early on during the software life cycle. Subsequent iterations build on the initial software produced during the first iteration.

Incremental Model



Incremental Model

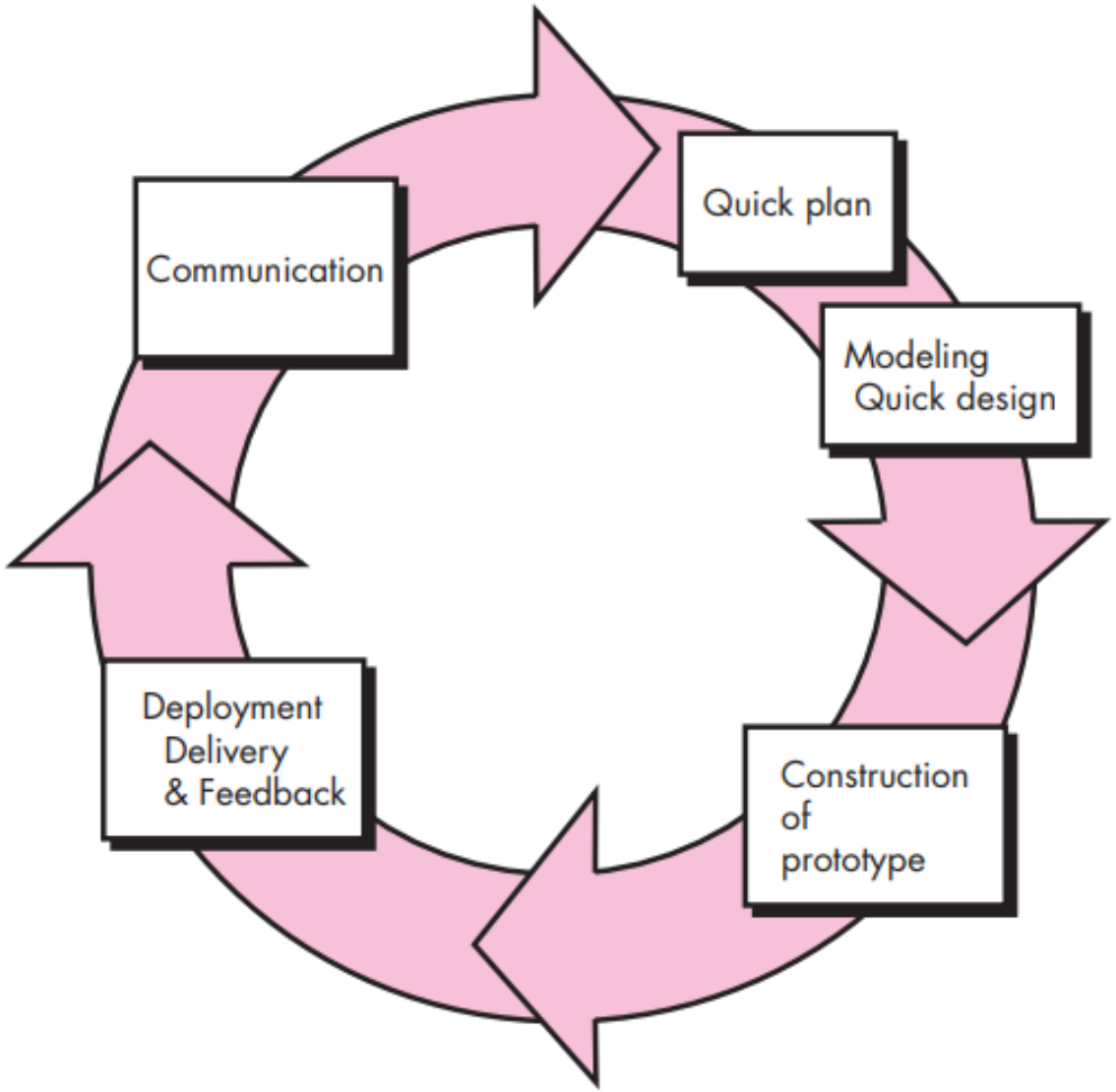
Advantages

- Generates working software quickly and early during the software life cycle.
- More flexible and inexpensive to change scope and requirements.
- Easier to test and debug during a smaller iteration.
- Easier to manage risk because risky pieces are identified and handled during its iteration.
- Each of the iterations is an easily managed landmark

Disadvantages

- Each phase of an iteration is rigid and do not overlap each other.
- Problems as regard to system architecture may arise as a result of inability to gathered requirements up front for the entire software life cycle

Evolutionary or Iterative or Prototype Model



Evolutionary Model

Advantages

- Quick design and planning helps in faster development
- Prototypes helps to get early feedback
- Several iteration helps to satisfy needs of various stakeholders

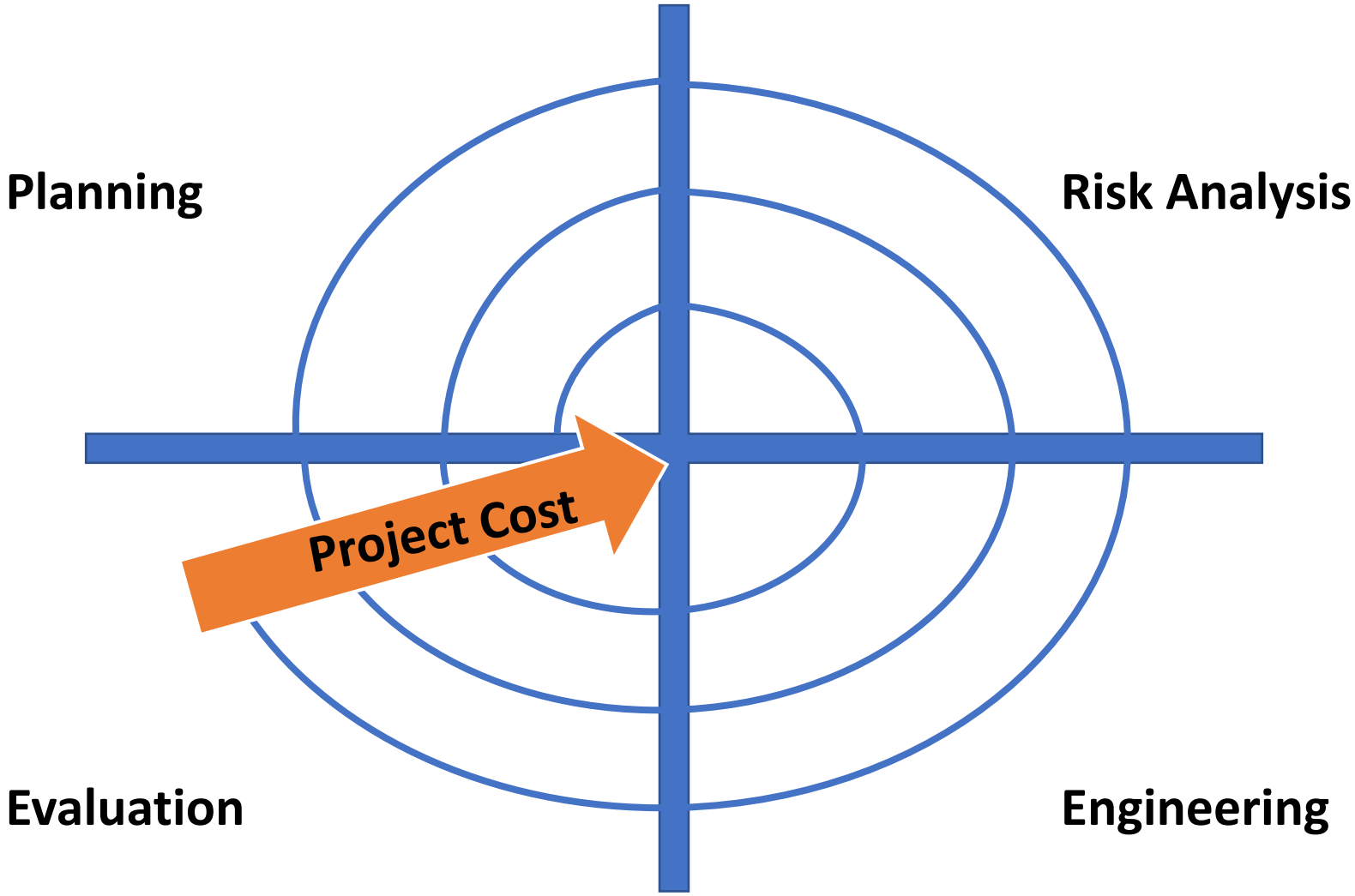
Disadvantages

- Time consuming and expensive if too many changes occurs in every iterations
- Stakeholders may play foul to get unnecessary changes and updates in the name of bug fixing.

Spiral Model

- The spiral model is similar to the incremental model, with more emphases placed on risk analysis.
- A software project continually goes through these phases in iterations which are called spirals. In the baseline spiral requirements are gathered and risk is assessed. Each subsequent spiral builds on the baseline spiral.
- The spiral model has four phases namely Planning, Risk Analysis, Engineering and Evaluation.
  - Requirements are gathered during the planning phase.
  - In the risk analysis phase, a process is carried out to discover risk and alternate solutions. A prototype is produced at the end of the risk analysis phase.
  - Software is produced in the engineering phase, alongside with testing at the end of the phase.
  - The evaluation phase provides the customer with opportunity to evaluate the output of the project to date before the project continues to the next spiral.

Spiral Model



Spiral Model

Advantages

- High amount of risk analysis
- Good for large and mission-critical projects.
- Software is produced early in the software life cycle

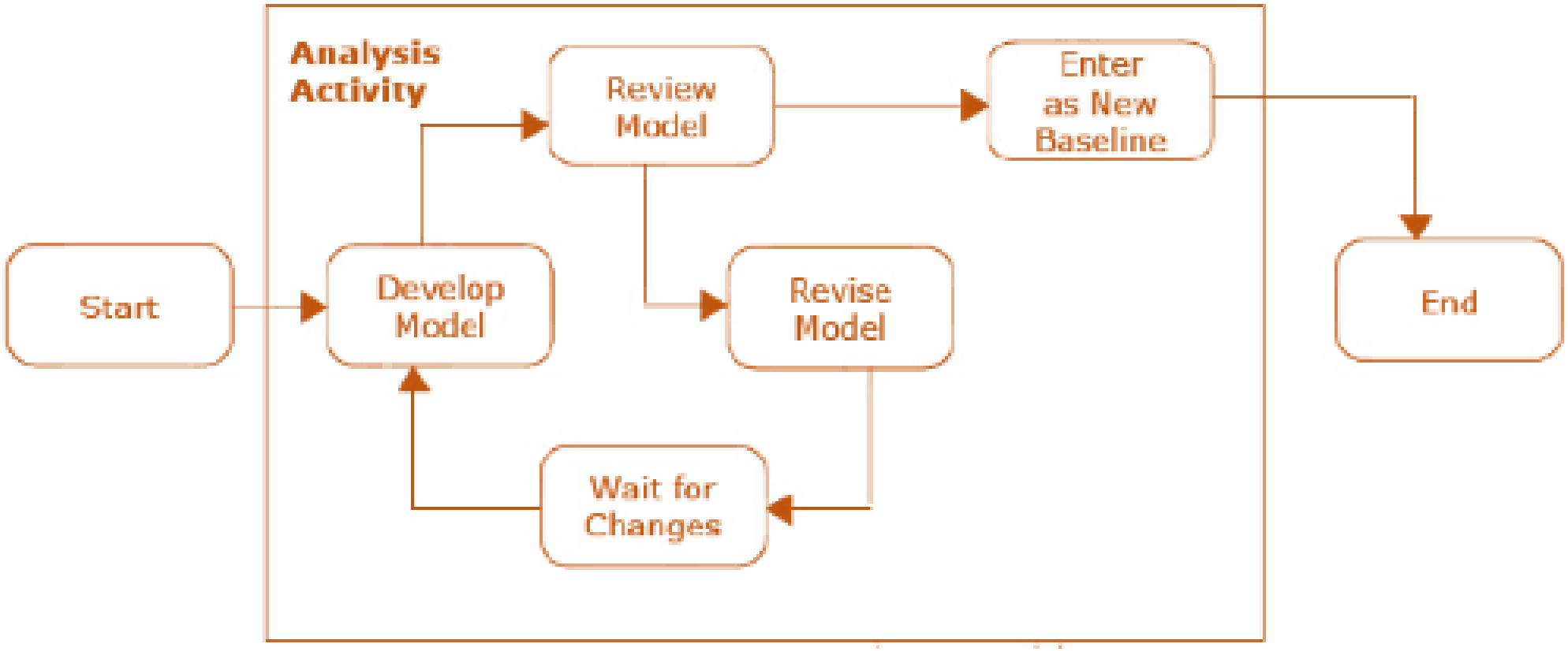
Disadvantages

- 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 smaller projects.

Concurrent Development Model

- In this model all the software engineering activities analysis design etc. Are built / performed parallel with each other.
- For each of the activity, a state transition diagram is made.
- All the events of an activity are shown as states and there should be transitions among them.

Concurrent Development Model



Specialized Process Model

- Component-Based Development
  - Commercial off-the-shelf (COTS) software components, developed by vendors who offer them as products, provide targeted functionality with well-defined interfaces that enable the component to be integrated into the software that is to be built.
  - 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.

Specialized Process Model

- Formal Methods Model
  - The formal methods model encompasses a set of activities that leads to formal mathematical specification of computer software. Formal methods enable to specify, develop, and verify a computer-based system by applying a rigorous, mathematical notation.
  - The development of formal models is currently quite time consuming and expensive.
  - Because few software developers have the necessary background to apply formal methods, extensive training is required.
  - It is difficult to use the models as a communication mechanism for technically unsophisticated customers.

Summary

- Brief introduction regarding software engineering domain
- Several process models
  - Model descriptions
  - Model illustrations
  - Benefits and limitations