Chapter 1

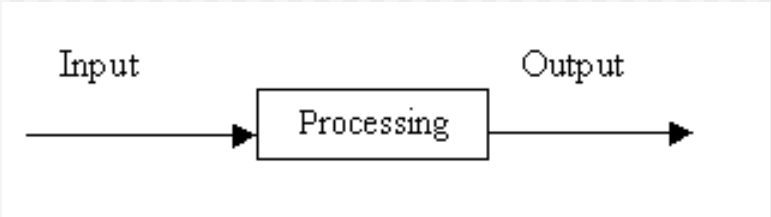
Fundamentals of Software Engineering

What is Systems Analysis and Design?

- Systems are created to solve problems.
- Think of the systems approach as an organized way of dealing with a problem.
- System Analysis and Design, mainly deals with the software development activities.

Defining A System

- A collection of components that work together to realize some objective forms a system.
- Basically there are three major components in every system, namely input, processing and output.



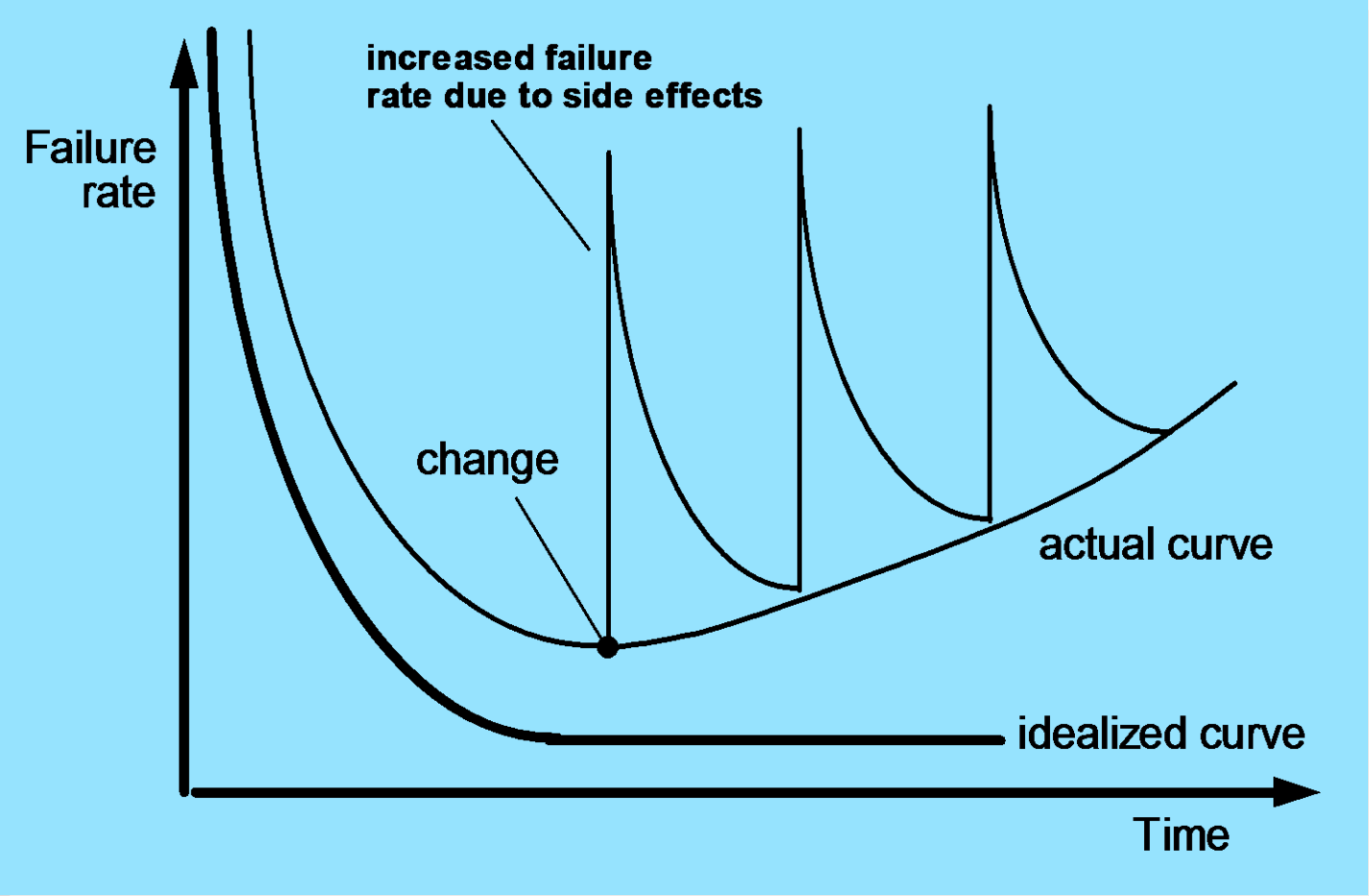
What is Software?

Software is: (1) instructions (computer programs) that when executed provide desired features, function, and performance; (2) data structures that enable the programs to adequately manipulate information and (3) documentation that describes the operation and use of the programs.

What is Software?

- ***Software is developed or engineered, it is not manufactured in the classical sense.***
  - ***High quality through good design***
  - ***Dependent on people***
  - ***Construction of product, with different approach.***
- ***Software doesn't "wear out."***
  - **h/w susceptible with environmental causes**
- ***Although the industry is moving toward component-based construction, most software continues to be custom-built.***

Wear vs. Deterioration



Software Myths

- Affect managers, customers (and other non-technical stakeholders) and practitioners
- Are believable because they often have elements of truth,

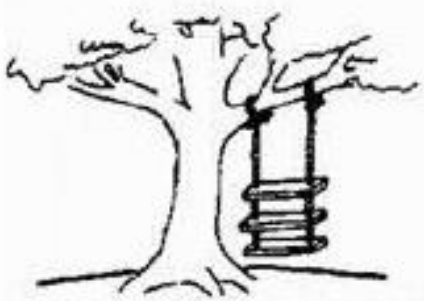
but …

- Invariably lead to bad decisions,

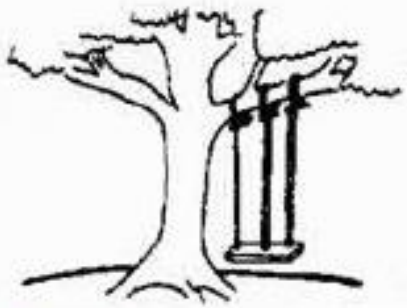
therefore …

- Insist on reality as you navigate your way through software engineering

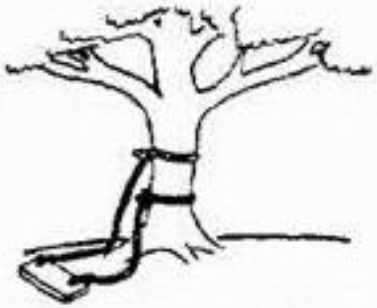
How is Software usually Constructed …



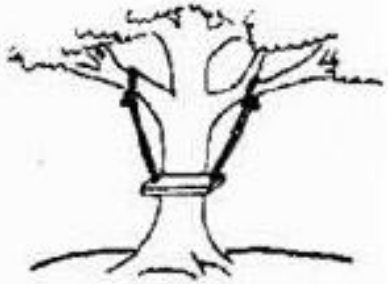
The requirements specification was defined like this



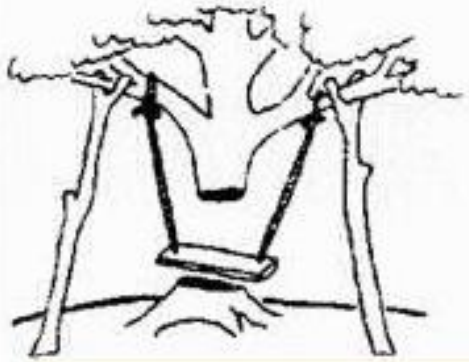
The developers understood it in that way



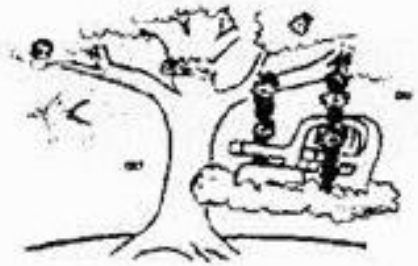
This is how the problem was solved before.



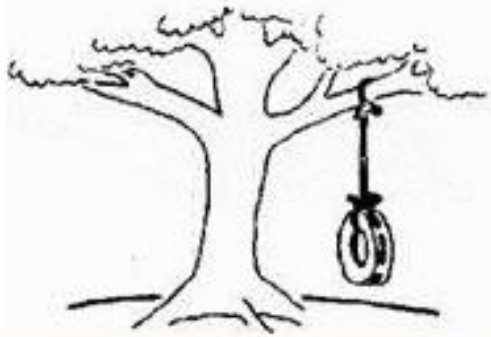
This is how the problem is solved now



That is the program after debugging



This is how the program is described by marketing dept.



This, in fact, is what the customer wanted …

Software Management Myths

- We have standards and procedures for building software, so developers have everything they need to know.
- We have state-of-the-art software development tools; after all, we buy the latest computers.
- If we're behind schedule, we can add more programmers to catch up.
- A good manger can manage any project.

Customer Myths

- A general statement of objectives is sufficient to begin writing programs -we can fill in the details later.
- Requirement changes are easy to accommodate because software is flexible.
- I know what my problem is; therefore I know how to solve it. This primarily is seen evidently because the clients do not have a first hand

Developer Myths

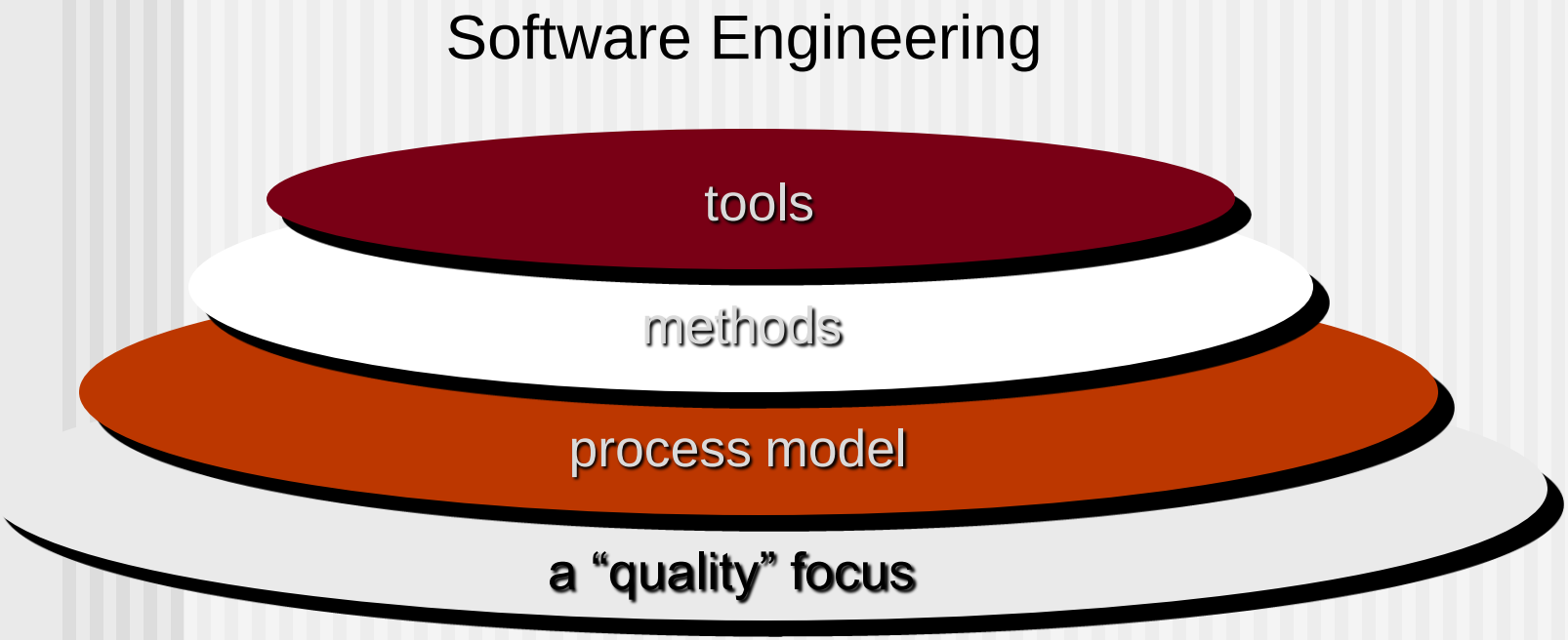
- If I miss something now, I can fix it later.
- Once the program is written and running, my job is done.
- Until a program is running, there's no way of assessing its quality.
- The only deliverable for a software project is a working program

Software Engineering- A layered Technology

- SE: Establishment and use of sound engineering principles to obtain economically s/w that is reliable and works efficiently on real machines

- IEEE: SE (1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of s/w that is, the application of engineering to s/w (2) the study of approaches as in (1).

A Layered Technology



Process, Methods, and Tools

- **Process**
  - The foundation for software engineering is the **process layer**. Software engineering process is the glue that holds the technology layers together and enables rational and timely development of computer software
  - Process defines the framework for a set of key process areas (KPAs) that must be established for effective delivery of software engineering technology.

Process, Methods, and Tools

- **Methods**
  - Software engineering **methods** provide the technical "how to's" for building software.
  - Methods includes array of tasks that are, requirements analysis, design, program construction, testing, and maintenance.
  - SE methods relay on a set of basic principles that govern each area of the technology and include modeling activities and other descriptive techniques.

Process, Methods, and Tools

- **Tools**
  - Software engineering **tools** provide automated or semi-automated support for the process and the methods.

  - When tools are integrated so that information created by one tool can be used by another, a system for the support of software development, called **C**omputer **A**ided **S**oftware **E**ngineering (CASE) is established.

A Process Framework

Process framework

Framework activities

work tasks

work products

milestones & deliverables

QA checkpoints

Umbrella Activities

Framework Activities

- Communication

- Planning

- Modeling
  - Analysis of requirements
  - Design

- Construction
  - Code generation
  - Testing

- Deployment

- **Communication**: involves heavy communication with the customer (and other stakeholders) and encompasses requirements gathering.
- **Planning**: Describes the technical tasks to be conducted, the risks that are likely, resources that will be required, the work products to be produced and a work schedule.
- **Modeling**: encompasses the creation of models that allow the developer and customer to better understand S/W req. and the design that will achieve those req.

- **Construction**: combines code generation and the testing required uncovering errors in the code.
- **Deployment**: deliver the product to the customer who evaluates the delivered product and provides feedback.
- Each S/W eng action is represented by a number of different task sets – each a collection of S/W eng work tasks, related work products, quality assurance points, and project milestones.
- The framework described in the generic view of S/W eng is complemented by a number of *umbrella activities*.

Umbrella Activities

- Software project management
- Formal technical reviews
- Software quality assurance
- Software configuration management
- Work product preparation and production
- Reusability management
- Measurement
- Risk management

Software Engineering Products and Processes

The software customer wants

quality **software products** at reasonable prices

The software producer wants

a well managed **software production process**

that consistently produces quality software

in a cost-effective manner

Some organizations have a defined and managed **software engineering process**

The Software Process

- The software product is created as part of the **Software Engineering Process**

Definition:

- **the Software Process is a description of the process which guides the software engineers as they work**

The Software Product

- Composed of **programs, data and documents**
- **Delivers hardware computing potential**
- **Delivers information**

Software Product Characteristics

- »developed or engineered, not manufactured
- »doesn’t wear out
- »most is custom built

The Software Process

The set of activities which produce a software product

The sequence of steps to develop and maintain software

Sets out the technical and management framework for applying methods, tools and people to the software task

Generic phases of software engineering process

The work associated with software engineering is associated with three generic phases.

- Definition phase
- Development phase
- Maintenance phase

Generic phases of software engineering process

- **Definition phase**
  - The software developer attempts to identify what information is to be processed, what function and performance are desired, what interfaces are to be established and what validation criteria are required to define a successful system.
  - Three major tasks will occur : system or information engineering, software project planning and requirements analysis

Generic phases of software engineering process

- **Development phase**
  - •During development a software engineer attempts to define how data are to be structured, how function is to be implemented as a software architecture, how interfaces are to be characterized, how the design will be translated into a programming language and how testing is performed.
  - •Three specific technical tasks will always occur in this phase: software design, code generation and software testing

Generic phases of software engineering process

- **Maintenance phase**
  - Maintenance focuses on changes. Four types of changes are,
    - Correction
    - Adaptation
    - Enhancement
    - Prevention

Roles & Responsibilities

Senior Managers

Practitioners

End-Users

Project Managers

Customers

Roles & Responsibilities

- Senior Managers define the business issue.
- Project Managers plan, motivate, Organize and control the practitioners who do the Software work.
- Practitioners deliver the technical skills that are necessary to engineer a product or application.
- Customer specifies the requirements for the software to be developed.
- End Users interact with the software once it is released.

S/W Development Process

- A set of phases and each phase being a sequence of steps
- For each phase there are
  - A variety of methodologies
  - Corresponding to different sequence of steps for a phase
- Why have phases?
  - To employ divide and conquer
  - Each phase handles a different part of the problem
  - Helps in continuous validation

Software Development Life Cycle (SDLC)

A framework that describes the activities performed at each stage of a software development project.

- Two types:
  1) Sequential
  2) Incremental

Software Development Life Cycle (SDLC)

- Commonly has these activities:
  1. Requirements analysis,
  2. Design
  3. Coding,
  4. Testing,
  5. Delivery

- Different models perform them in different manner!

Waterfall Model

- It is also called as linear sequential model.
- In this model whole application is developed in a sequential approach.
- In this model each phase must be completed fully before the next phase begin.
- Provides structure to inexperienced staff.

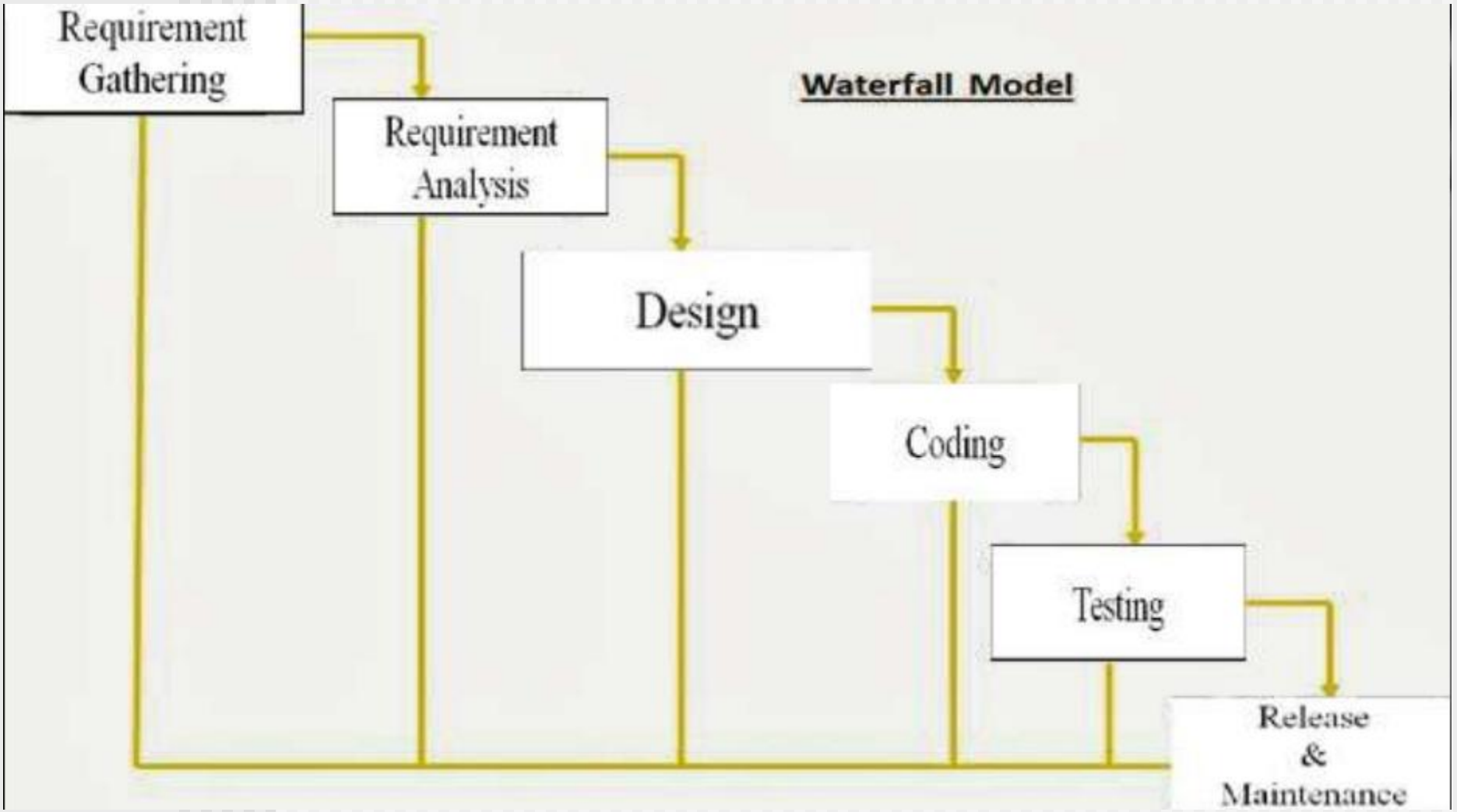
Waterfall Model

- The first formal description of the waterfall model is preseted in 1970 by Winston W.Royce.
- Royce presented this model as as an example of a flawed, on-working model.
- It has been widely used for software projects ever since.

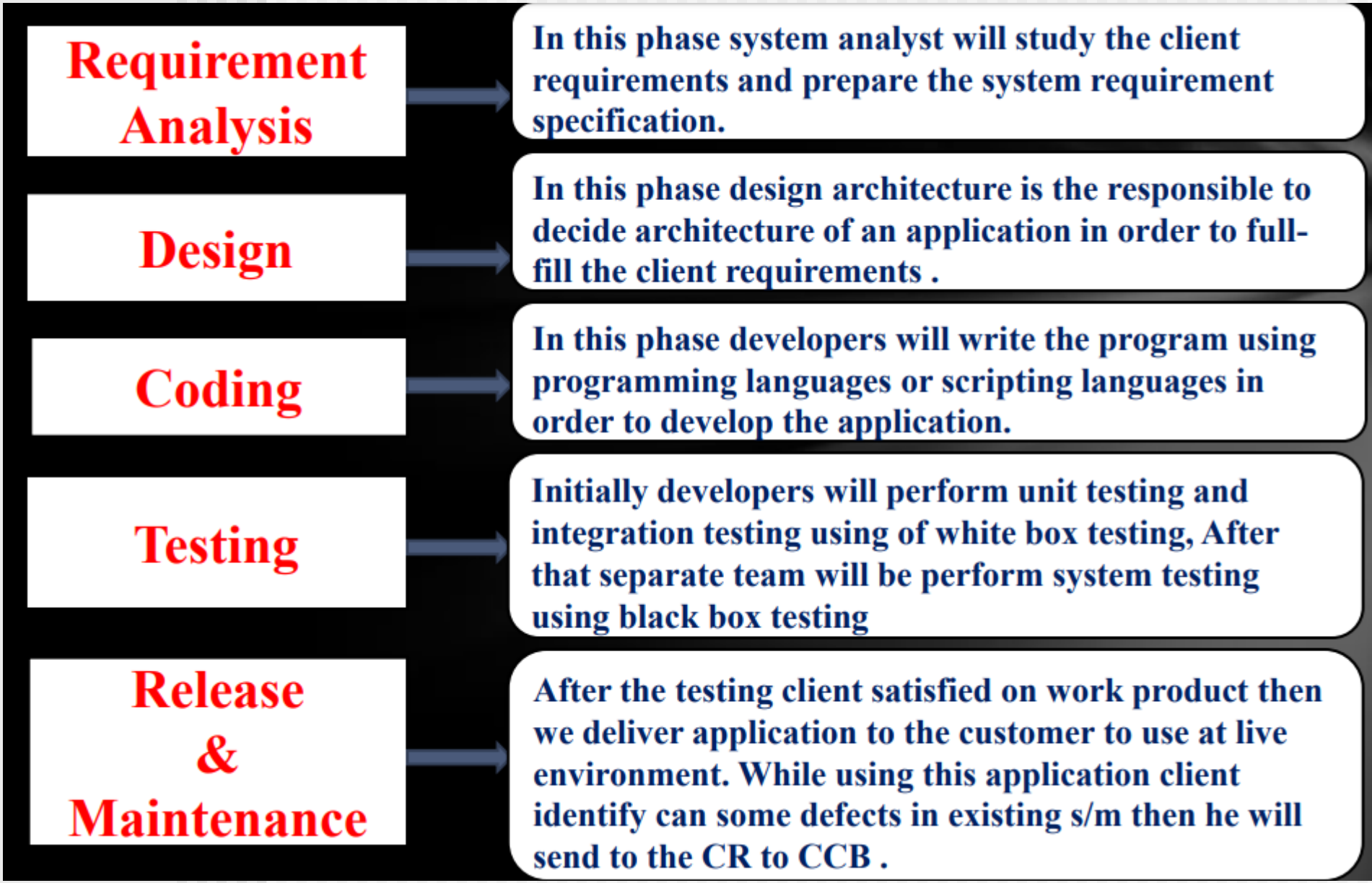
Waterfall Model

- Requirements are very well known.
- Product definition is stable.
- Technology is understood.
- New version of an existing product.

Waterfall Model



Waterfall Model



Waterfall Model Advantages

- A waterfall model is easy to implementation.
- It helps to find errors earlier
- Easy to understand, easy to use.
- Works well when quality is more important than cost or schedule
- 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 Dis-advantages

- It is only suitable for the small size projects.
- Constant testing of the design is needed.
- If requirements may change the Waterfall model may not work.
- Difficult to estimate time and cost for each stage of the development process.
- Adjust scope during the life cycle can kill a project.
- High amount of risk and uncertainty.
- This model is not suitable to handle dynamic changes in the requirements

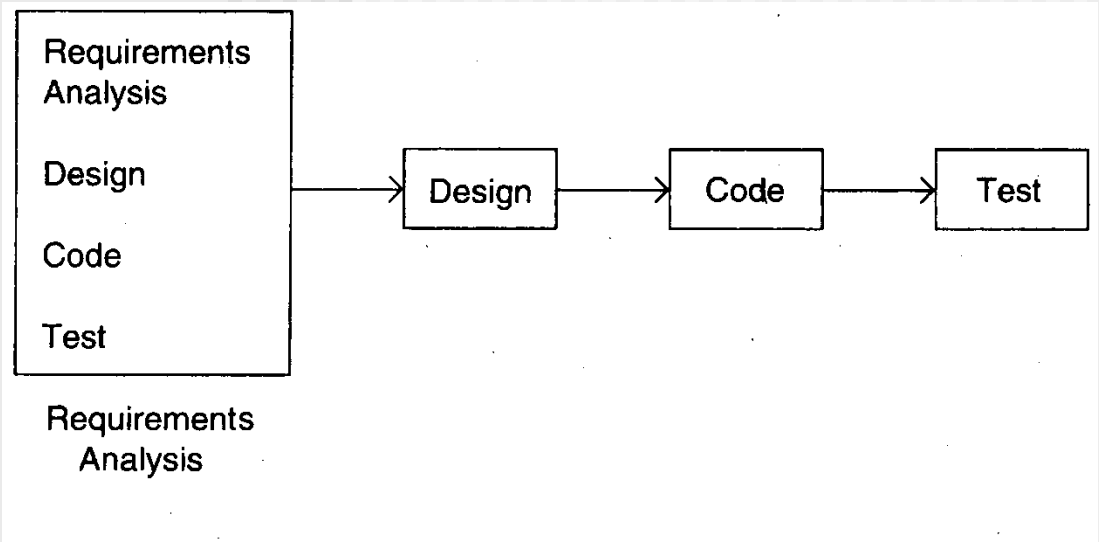
Prototype Model

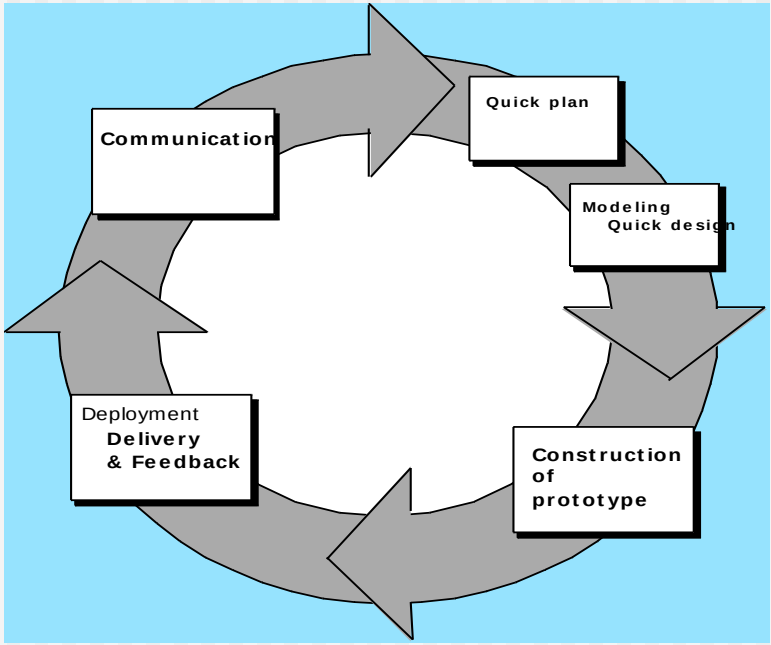
- Addresses the requirement specification limitation of waterfall
- Evolutionary process model – iterative.
- Instead of freezing requirements only by discussions, a prototype is built to understand the requirements
- Helps alleviate the requirements risk
- To accommodate a product that evolves over time
- A small waterfall model replaces the requirements stage

Development of prototype

- Starts with initial requirements
- Only key features which need better understanding are included in prototype
- No point in including those features that are well understood
- Feedback from users taken to improve the understanding of the requirements

Prototype Model





Prototyping

- The prototyping paradigm begins with *communication* where requirements and goals of Software are defined.
- Prototyping iteration is *planned* quickly and modeling in the form of quick design occurs.
- The *quick design* focuses on a representation of those aspects of the Software that will be visible to the customer “Human interface”.
- The quick design leads to the *Construction of the Prototype*.
- The prototype is *deployed* and then *evaluated* by the customer.
- *Feedback* is used to refine requirements for the Software.

Prototyping

- Iteration occurs as the prototype is tuned to satisfy the needs of the customer, while enabling the developer to better understand what needs to be done.

- The prototype can serve as the “first system”. Both customers and developers like the prototyping paradigm as users get a feel for the actual system, and developers get to build Software immediately

Prototyping

- Cost can be kept low
  - Build only features needing clarification
  - “quick and dirty” – quality not important, scripting etc can be used
  - Things like exception handling, recovery, standards are omitted
  - Cost can be a few % of the total
  - Learning in prototype building will help in building, besides improved requirements

Prototyping

- Advantages
  - Requirement will be more stable and more likely to satisfy user needs
  - Early opportunity to explore scale/performance issues
  - Ability to modify or cancel the project early
  - Enhanced user engagement
- Disadvantages:
  - Potential hit on cost and schedule
  - Potential false sense of security if prototype does not focus on key (high risk) issues

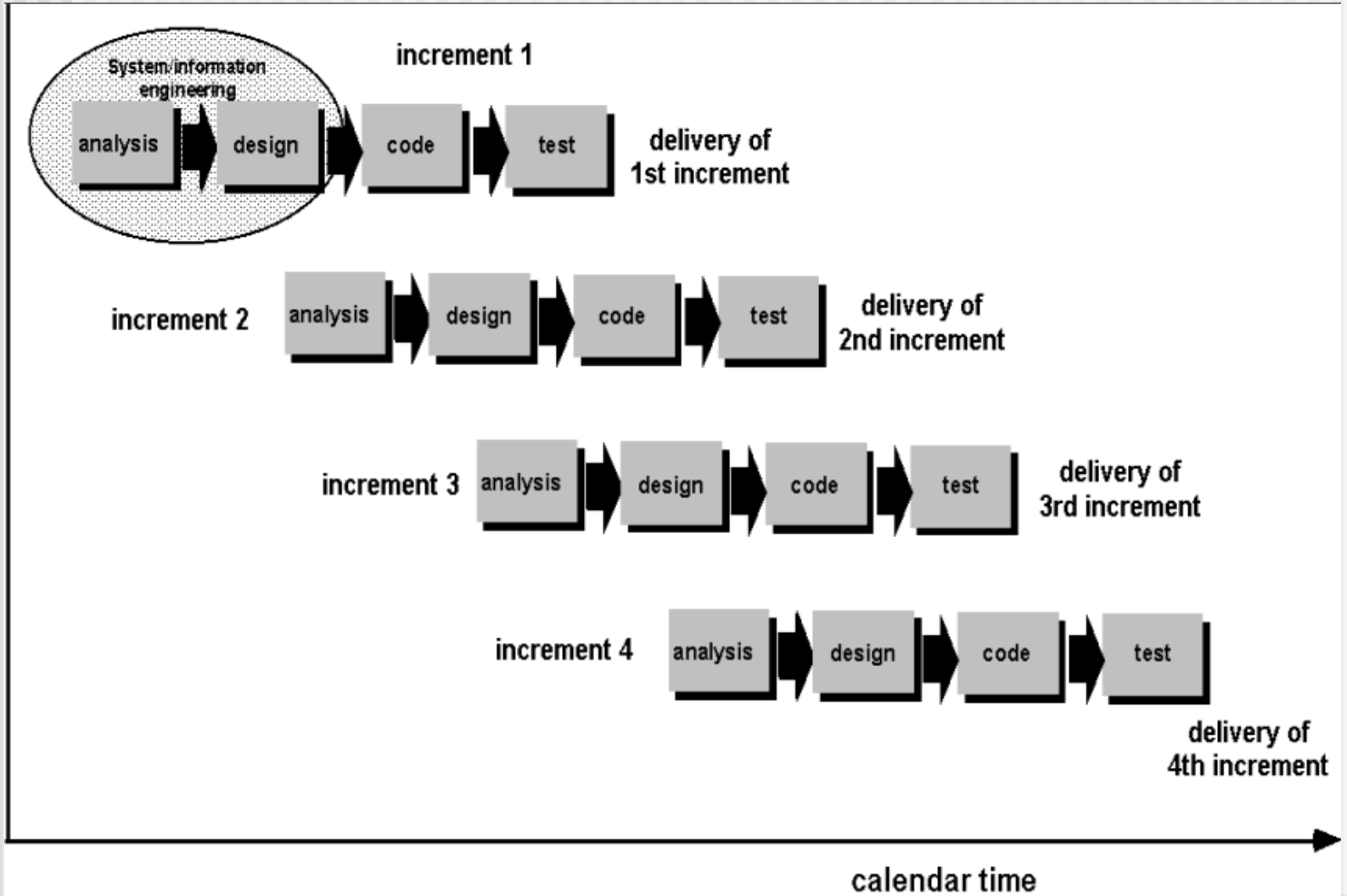
Prototyping

- Applicability:
  - When requirements are hard to elicit
  - When confidence in requirements is low
  - Where requirements are not well understood
  - When design is driven by user needs
- Variants
  - Paper Prototypes
  - UI Prototypes
  - Technology Proving
  - Rapid Prototyping environments

Incremental process Model

- Rather than deliver the system as a single delivery, the development and delivery is broken down into increments with each increment delivering part of the required functionality.
- User requirements are prioritised and the highest priority requirements are included in early increments.
- Once the development of an increment is started, the requirements are frozen though requirements for later increments can continue to evolve.

Incremental process Model



Incremental process Model

- Development occurs as a succession of releases with increasing functionality.
- Combines elements of the water fall model applied in an iterative fashion.
- Each linear sequence produces deliverables of the software.
- Customers provide feedback on each release, used in deciding requirements and improvements for next release.
- There is no “maintenance” phase – each version includes both problem fixes as well as new features.

Incremental Model Advantages

- Customers get usable functionality earlier than with waterfall.
- Early feedback improves likelihood of producing a product that satisfies customers.
- The quality of the final product is better
  - The core functionality is developed early and tested multiple times (during each release)
  - Only a relatively small amount of functionality added in each release: easier to get it right and test it thoroughly
  - Detect design problems early and get a chance to redesign

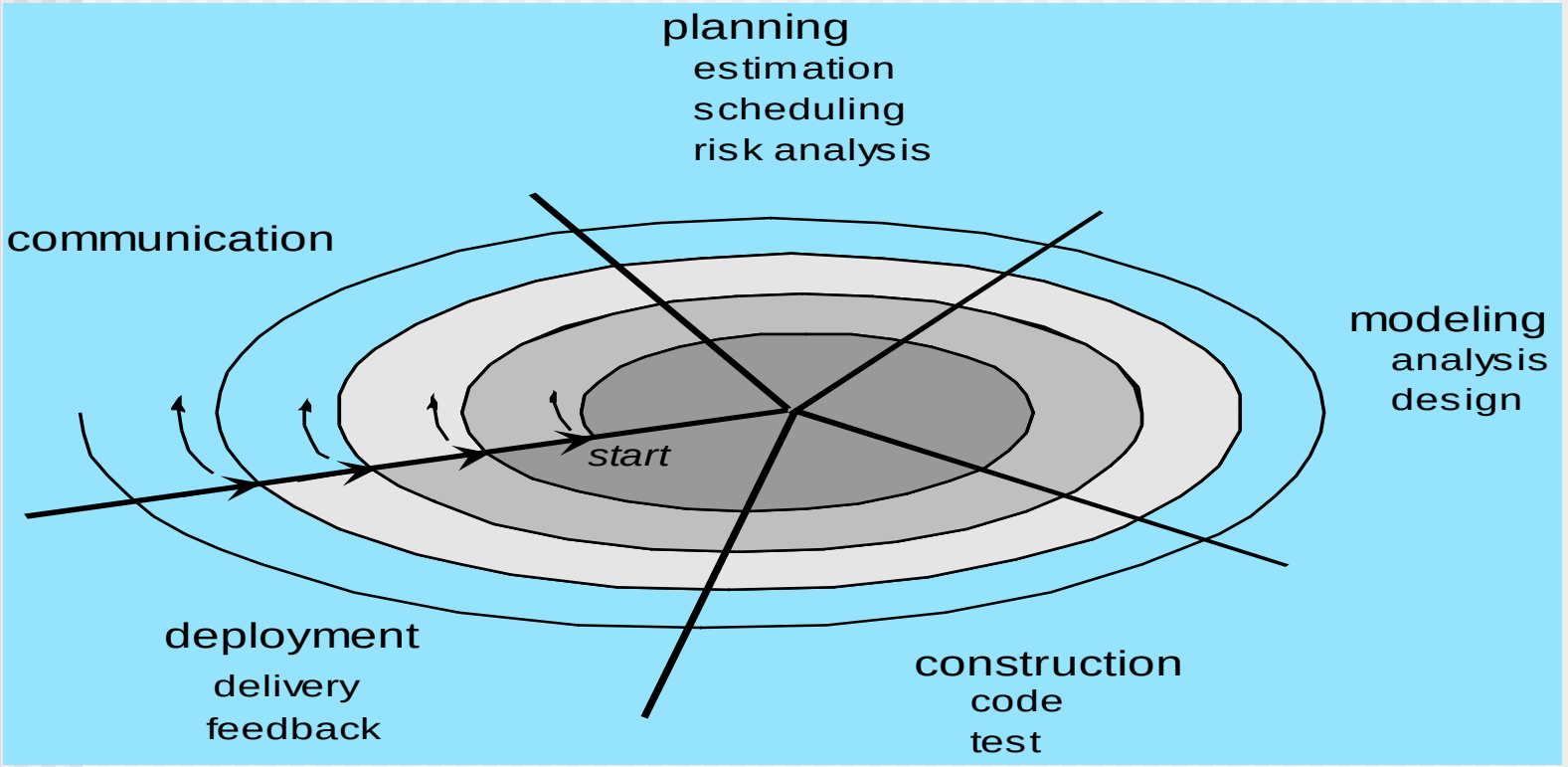
Incremental Model Disadvantages

- Needs good planning and design.
- Needs a clear and complete definition of the whole system before it can be broken down and built incrementally.
- Total cost is higher than waterfall.

Spiral Model

- A hybrid model where the development of the system spirals outward from an initial outline through to the final developed system.
- couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model.

Spiral Model



Spiral Model

- Process is represented as a spiral rather than as a sequence of activities with backtracking.
- Each loop in the spiral represents a phase in the process.
- Like the innermost loop might be concerned with system feasibility, next loop with system requirements, next loop with system design and so on.
- No fixed phases such as specification or design - loops in the spiral are chosen depending on what is required.
- Risks are explicitly assessed and resolved throughout the process.

Spiral Model

- Each loop in the spiral is split into four sectors:
- Objective setting • Specific objectives for the phase are identified.
- Risk assessment and reduction • Risks are assessed and activities put in place to reduce the key risks.
- Development and validation • A development model for the system is chosen which can be any of the generic models.
- Planning • The project is reviewed and the next phase of the spiral is planned.

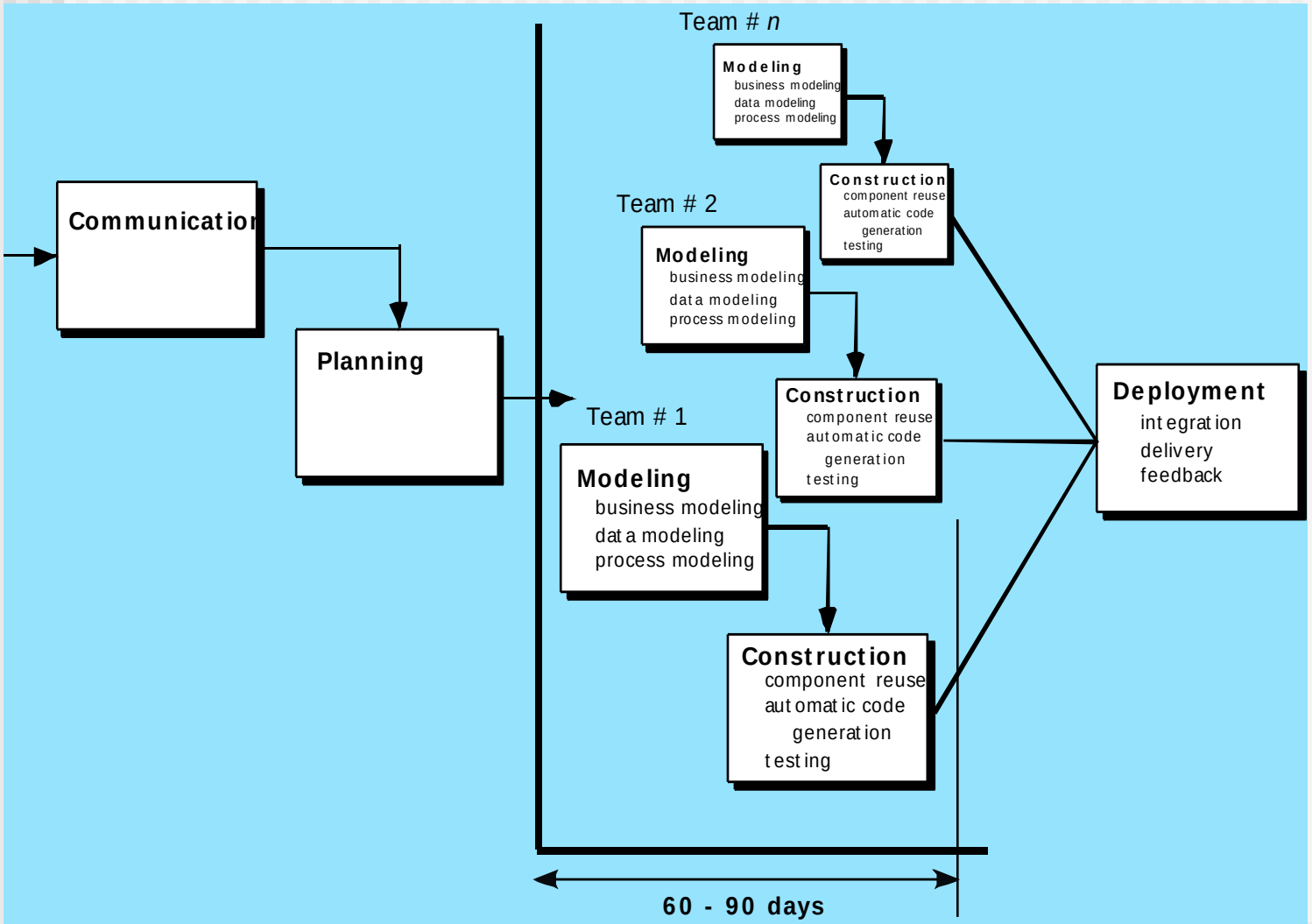
Spiral Model

- Advantages
  - Explicit consideration of risks (alternative solutions are evaluated in each cycle).
  - More detailed processes for each development phase.
- Disadvantages
  - Cost is high.
  - Sometime difficult to implement or too time consuming.

RAD Model

- an incremental software process model that emphasizes a short development cycle.
- If requirements are well understood and project scope is constrained, the RAD process enables a development team to create a fully functional system within a short period of time.
- Using Component based construction approach.

RAD Model


Team # n
Modeling
business modeling
data modeling
process modeling
Construction
component reuse
automatic code
generation
testing
Team # 2
Modeling
business modeling
data modeling
process modeling
Construction
component reuse
automatic code
generation
testing
Communication
Planning
Team # 1
Modeling
business modeling
data modeling
process modeling
Construction
component reuse
automatic code
generation
testing
Deployment
integration
delivery
feedback
60 - 90 days

RAD Model

- **Advantages**
  - RAD reduces the development time
  - Reusability of components help to speed up development.
  - All functions are modularized so it is easy to work with.

RAD Model

- **Disadvantages**
  - For large projects RAD require highly skilled engineers in the team.
  - Both end customer and developer should be committed to complete the system in a much abbreviated time frame
  - If commitment is lacking RAD will fail.
  - RAD is based on Object Oriented approach and if it is difficult to modularize the project the RAD may not work well.

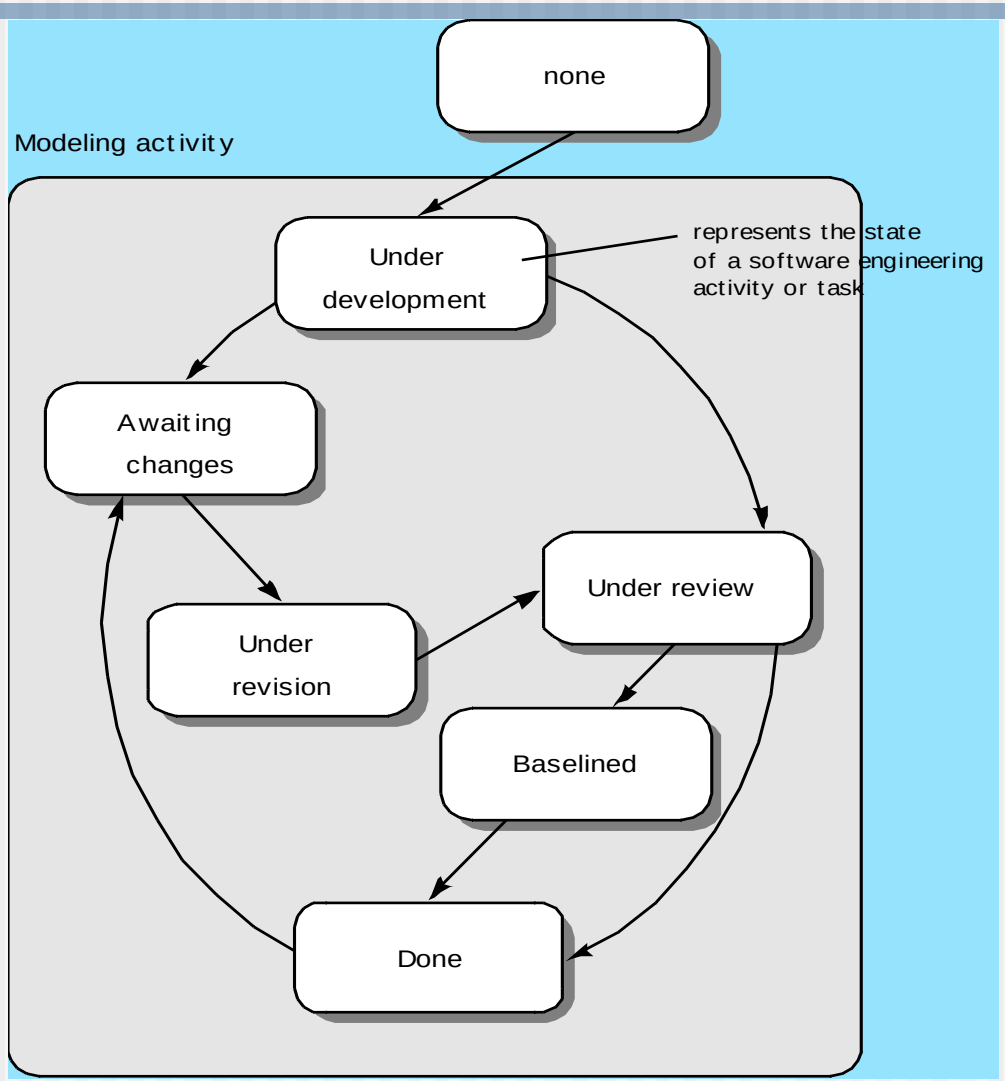
Concurrent Development Model

- Represented schematically as a series of framework activities, Software engineering actions of tasks, and their associated states.
- more appropriate for system engineering projects where different engineering teams are involved.
- All activities exist concurrently but reside in different states.

Concurrent Development Model

- early in the project the *communication* activity has completed its first iteration and exists in the **awaiting changes** state.
- The *modeling* activity which existed in the **none** state while initial communication was completed now makes a transition into **underdevelopment** state.
- If, however, the customer indicates the changes in requirements must be made, the *modeling* activity moves from the **under development** state into the **awaiting changes** state.

Concurrent Development Model



Component Based Model

- COTS (Commercial Off The Shelf) software components, developed by vendors who offer them as products can be used when software is to built.
- Provides targeted functionality with well defined interfaces.
- Incorporates many of the characteristics of spiral model.

Component Based Model

- Regardless of technology to be used, it must follow the steps like –
  - – Available component based products are researched and evaluated for the current application.
  - – Component integration issues is to dealt.
  - – A software architecture is designed to accommodate the components.
  - – Components are integrated into the architecture.
  - – Comprehensive testing is conducted to ensure proper functionality.

Component Based Model

- This model leads to software reuse.
- Provides software engineers with a number of measureable benefits.

Formal Methods Model

- Comprises set of activities that leads to formal mathematical specification of computer software.
- Enables a software engineer to specify, develop and verify a computer based system by applying mathematical notation.
- Problems of ambiguity, incompleteness and inconsistency can be managed by this method.
- Provides a base for verification therefore enable a software engineer to discover and correct undetected errors also.

Formal Methods Model

- But using this method in current scenario is time consuming and expensive.
- Extensive training is required for applying this method as few developers have the necessary background to work with this method.
- It is difficult to use the models as a communication mechanism for technically unsophisticated customers.

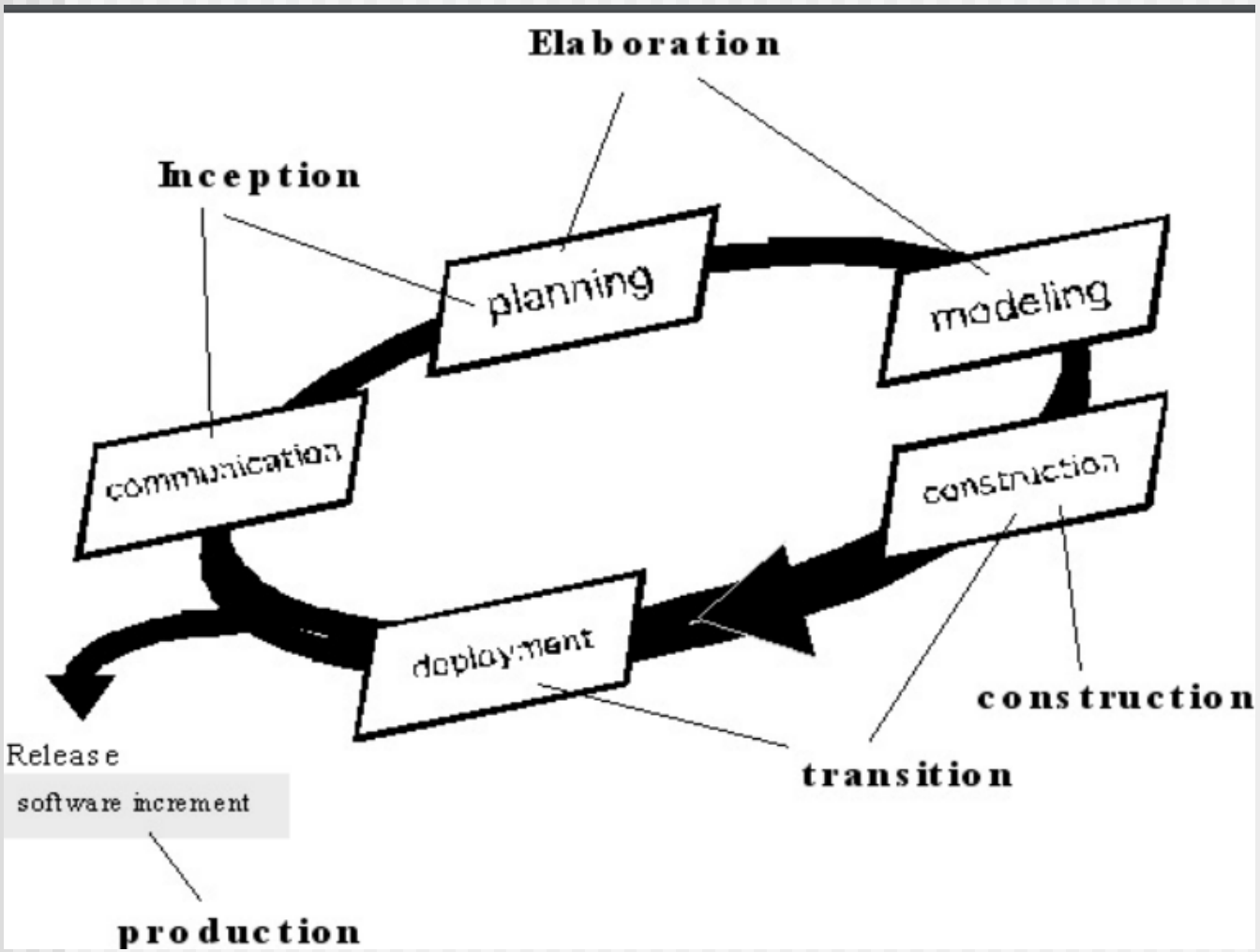
Aspect Oriented Software Model

- Complex software are being implemented as set of localized features, functions and information content referred as components.
- But it becomes crosscutting concerns when it flows across multiple systems.
- Aspectual requirements define these crosscutting concerns that have impact across the software architecture.
- AOSD provides a process and methodological approach for defining, specifying, designing and constructing aspects.

Unified Process Model

- Comprises best features and characteristics of conventional software process models.
- Emphasize importance of customer communication and streamlined methods for describing the customers view of system.
- Phases of Unified Process
- Inception = Involves customer communication and planning activities.

Unified Process Model



Unified Process Model

- Elaboration = Encompasses the customer communication and modeling activities of the generic process model. Architectural representation using Use Case Model, Analysis Model, Design Model, Implementation Model and Deployment Model.
- Construction = Develops or acquires the software components that will make each use case operational for end users.
- Transition = Software is given to end user for beta testing and user feedback reports both defects and necessary changes.

Unified Process Model

- Production = Coincides with the deployment activity of process. The on going use of software is monitored, support for the operating environment is provided, and defect reports and requests for changes are submitted and evaluated.