Introduction to Software Engineering

Ruchita Shah

Software Crisis

- In 1960s, Many software projects
    - Failed
    - Over-budget
    - Delayed
    - Unreliable software were delivered
    - Un-useful projects
- As hardware capability increased, Software became complex
- Larger software systems
    - Difficult to maintain
    - Expensive

Software Crisis

- Demand for larger system that is
  - Reliable
  - Within budget
  - On time
  - Useful
  - Easy to maintain
- These all lead to Software Engineering

Engineering

- Application of Scientific and Practical knowledge in order to invent, design, build, maintain and improve systems, processes etc.

Software

- Is a collection of Integrated Programs
- Consists of carefully organized instructions and codes written by programmes using some application software
- Consists of computer program and associated documents such as requirements, design models and user manuals

Software Engineering

- 1970 : Evolution of Software Engineering principle
- 1980: Automation of Software Engineering & CASE tools
- 1990 : More weightage on ‘Management’ of the projects

  Use of Standard Quality

  Process models such as ISO 9001

  Capability Maturity Models (CMM)

Models helped to develop & manage software development processes

Software Engineering - Definition

- 1969 : Fritz Bauer : “The establishment and use of sound engineering principles in order to obtain economical software that is reliable and works efficiently on real machines”
- 1990 : IEEE standard 610.12 : “Software engineering is (1) Application of systematic, disciplined and quantifiable approach to the development, operation and maintenance of software (application of Engineering to software) (2) the study of approaches as in (1)

Software Engineering - Definition

- Boehm : “Software engineering involves - The practical application of scientific knowledge to the design and construction of computer programs and the associated documentation required for developing, operating and maintaining them ”
- Combining all the 3 definitions: “Software engineering is a technological and managerial discipline concerned with systematic production and maintenance of software products that are developed and modified on time and within cost estimates.”

Software Engineering - Goal

- To improve quality of software product
- To increase the productivity
- To give job satisfaction to Software engineers

Difference between Software Engineering and Classical Engineering

- Intangible – No physical properties such as mass, volume, colour, odour etc. Program is not observable, its effect may be
- Software don’t Degrade: not as hardware degrade over time, Software do fail due to failures caused by design and implementation errors
- Obscurity between software modules : difficult to design software modules that fit seamlessly with each other, no side effects when they invoke other moduls

Difference between Software Engineering and Classical Engineering

- Classical engineering disciplines use tools and mathematical models to separate product from its design
- Software engineering relies on experience and judgement rather than mathematical formula

Difference between Software Engineering and Classical Engineering

- Classical engineering disciplines use tools and mathematical models to separate product from its design
- Software engineering relies on experience and judgement rather than mathematical formula

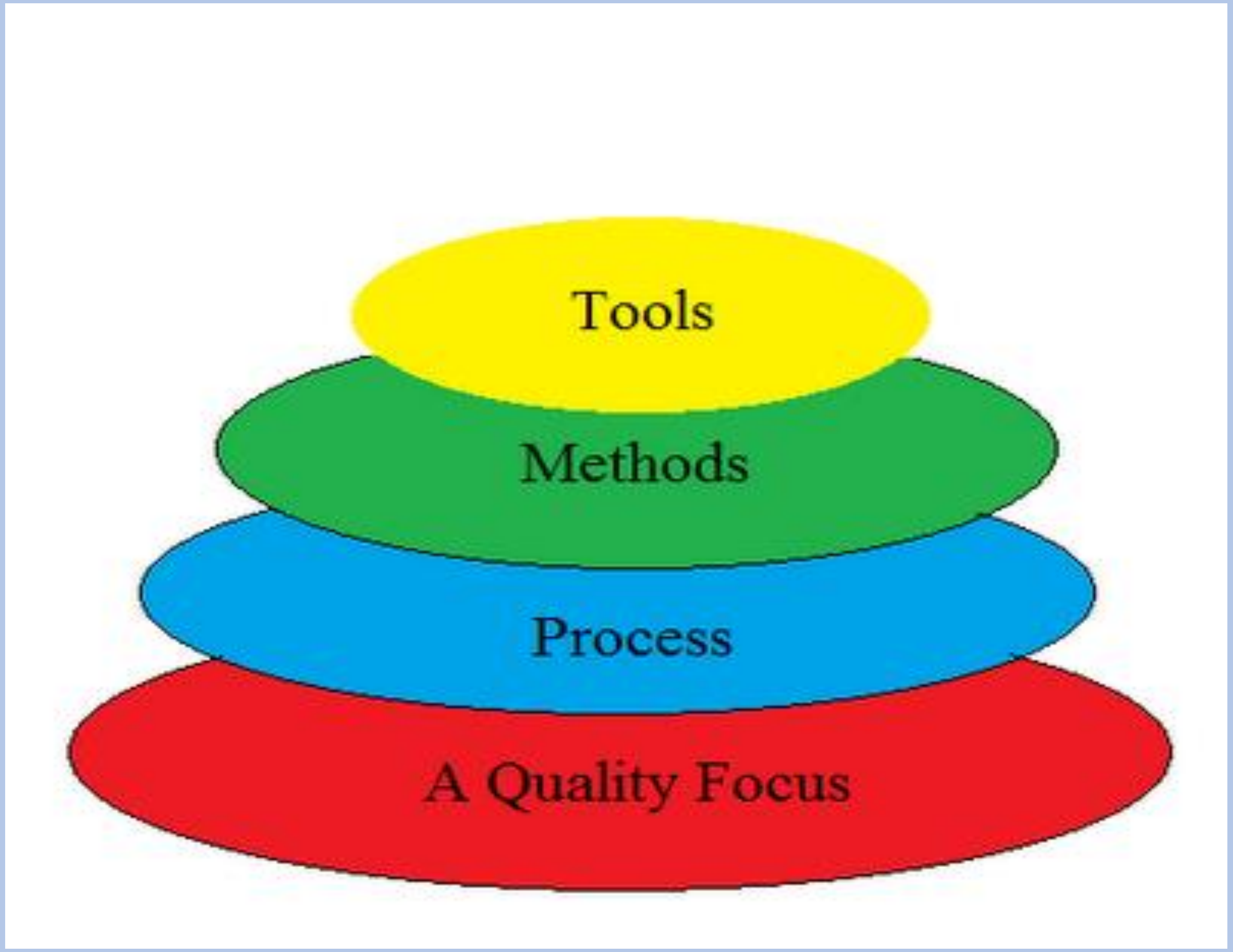
Evolving Role of Software

- Dual roles – A Product and A vehicle to deliver a product
- It is information transformer – produce, manage, acquire, modify, display & transmit information
- Delivers most important product – Information, transforms data
- Significant change in role of software

Evolving Role of Software

- Why it takes so long to get software finished?
- Why development costs so high?
- Why cant we find all the errors before handing software to customers?
- Why spend so much time & efforts to maintain existing programs?
- Why it is difficult to measure progress of a software that is being built?

Software Engineering – A Layered Technology



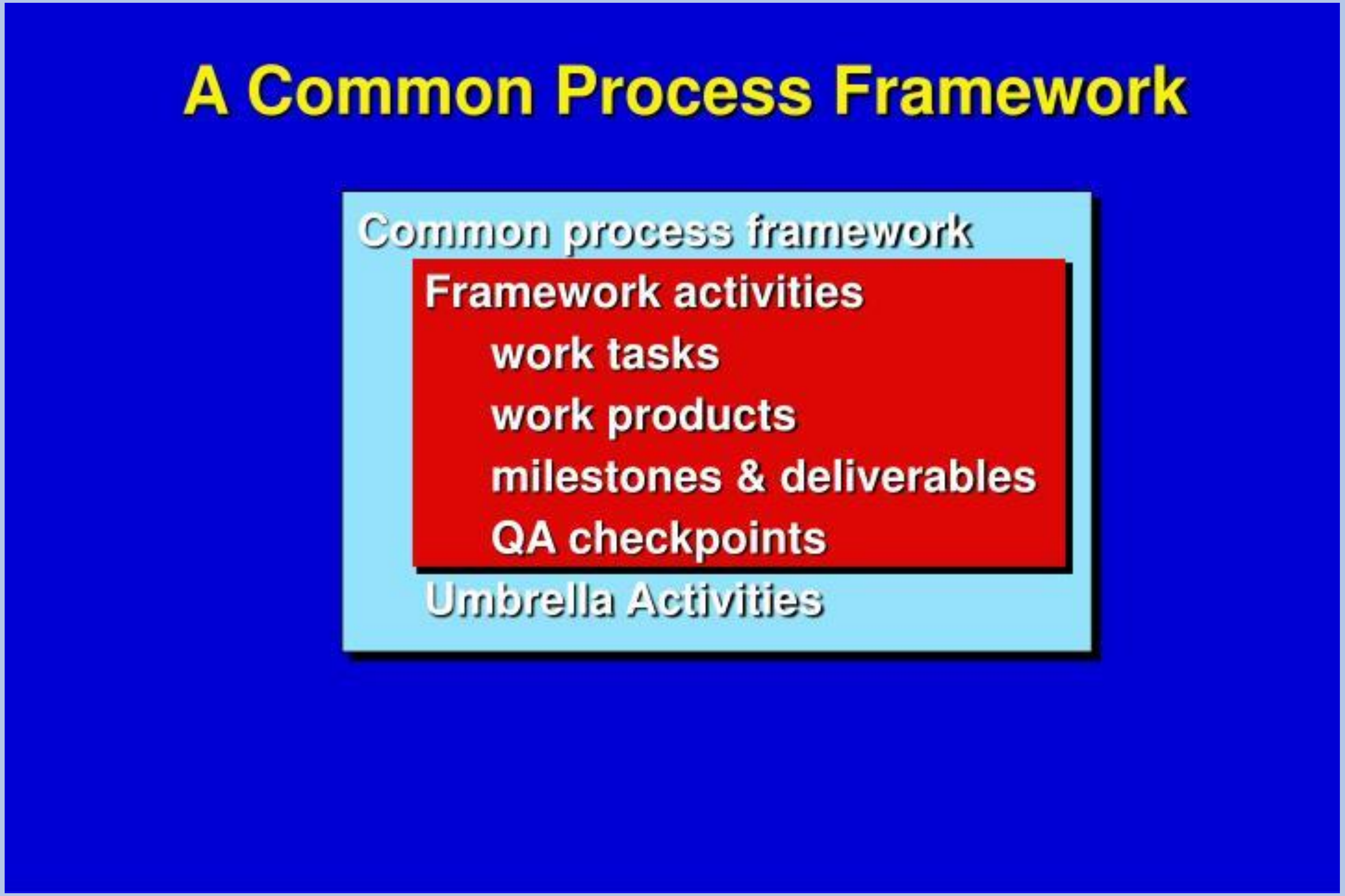
Software Engineering – A Layered Technology

- The Quality focus :
  - Bedrock
  - Any engineering approach rest on commitment for quality
  - Continuous process improvement

Software Engineering – A Layered Technology

- The Process :
  - Is the foundation of software engineering
  - Defines a framework for a set of key process areas (KPAs)
    - Effective delivery of SE technology
    - Management control of software project
    - Context of technical methods applied
    - Work products
    - Milestones, Quality ensured
    - Proper change management

Common Process Framework



A Common Process Framework

Common process framework

Framework activities

work tasks

work products

milestones & deliverables

QA checkpoints

Umbrella Activities

Software Engineering – A Layered Technology

- Methods :
    - Provide technical how-to’s for building software.
    - Encompass a broad array of tasks that include requirements analysis, program construction, testing, and support.
    - Rely on a set of basic principles that govern each area of the technology and include modeling activities and other descriptive techniques

Software Engineering – A Layered Technology

- Tools :
  - Provide automated or semi-automated support for the process and the methods.
  - CASE: computer-aided software engineering is a system for the support of software development.
    - Combines SW, HW, and a SE database (a repository containing important information about analysis, design, program construction, and testing)

A Generic View of SE

- Engineering is analysis, design, construction, verification, and management of technical (or social) entities.
- Entity -> computer software
- A software engineering process must be defined: definition phase, development phase, and support phase.

Definition Phase

Focuses on “what”

- What information is to be processed?
- What function and performance are desired?
- What system behavior can be expected?
- Interface, design constraints, validation criteria $\rightarrow$ identify key requirements of the system and SW.

Development Phase

Focuses on “how”

- How data are to be constructed?
- How function is to be implemented within a software architecture?
- Procedural details, interfaces, design → programming, testing.

Three technical tasks: software design, code generation, software testing

Support Phase

- Focuses on “change” associated with error correction, adaptations, changes.
- Four types of changes:
  - Correction : Corrective maintenance
  - Adaptation : Adaptive maintenance
  - Enhancement : Perfective maintenance
  - Prevention : Preventive maintenance, often called Software Engineering ; more easily corrected, adapted, and enhance.

Umbrella Activities

- Software project tracking and control
- Formal technical review
- Software quality assurance
- Software configuration management
- Document preparation and production
- Reusability management
- Measurement
- Risk management

A Common Process Framework

- **Common process framework**
  - **Framework activities**
    - **work tasks**
    - **work products**
    - **milestones & deliverables**
    - **QA checkpoints**
  - **Umbrella Activities**

Process Maturity

- Software Engineering Institute (SEI) has developed a comprehensive model predicated on a software engineering capabilities that should be present as organization reach different levels of process maturity.
- Called Capability Maturity Model (CMM).

CMM

- Level 1: Initial
  - The software process is characterized ad hoc and occasionally chaotic
- Level 2: Repeatable
  - Basic project management process are established to track cost, schedule, and functionality
- Level 3: Defined
  - The software process for both management and engineering activities is documented, standardized and integrated into an organization wide software process

CMM

- Level 4: Managed
  - Detailed measures of the software process and product quality are collected.
  - Both SW process and product are quantitatively understood and controlled using detailed measures.
- Level 5: Optimized
  - Continuous process improvement is enabled by quantitative feedback from the process and from testing innovative ideas and technologies.

Key Process Areas (KPAs)

- Each KPA is described by identifying the following characteristics:
  - Goals
  - Commitments
  - Abilities
  - Activities
  - Methods for monitoring implementation
  - Methods for verifying implementation

Key Process Areas (KPAs)

- Each of the KPAs is defined by a set of key practices that contribute to satisfying its goals.
- The key practices are policies, procedures, and activities.
- Key indicators: those of key practices or components of key practices that offer the greatest insight into whether the goals of a key process have been achieved.

Key Process Areas (KPAs)

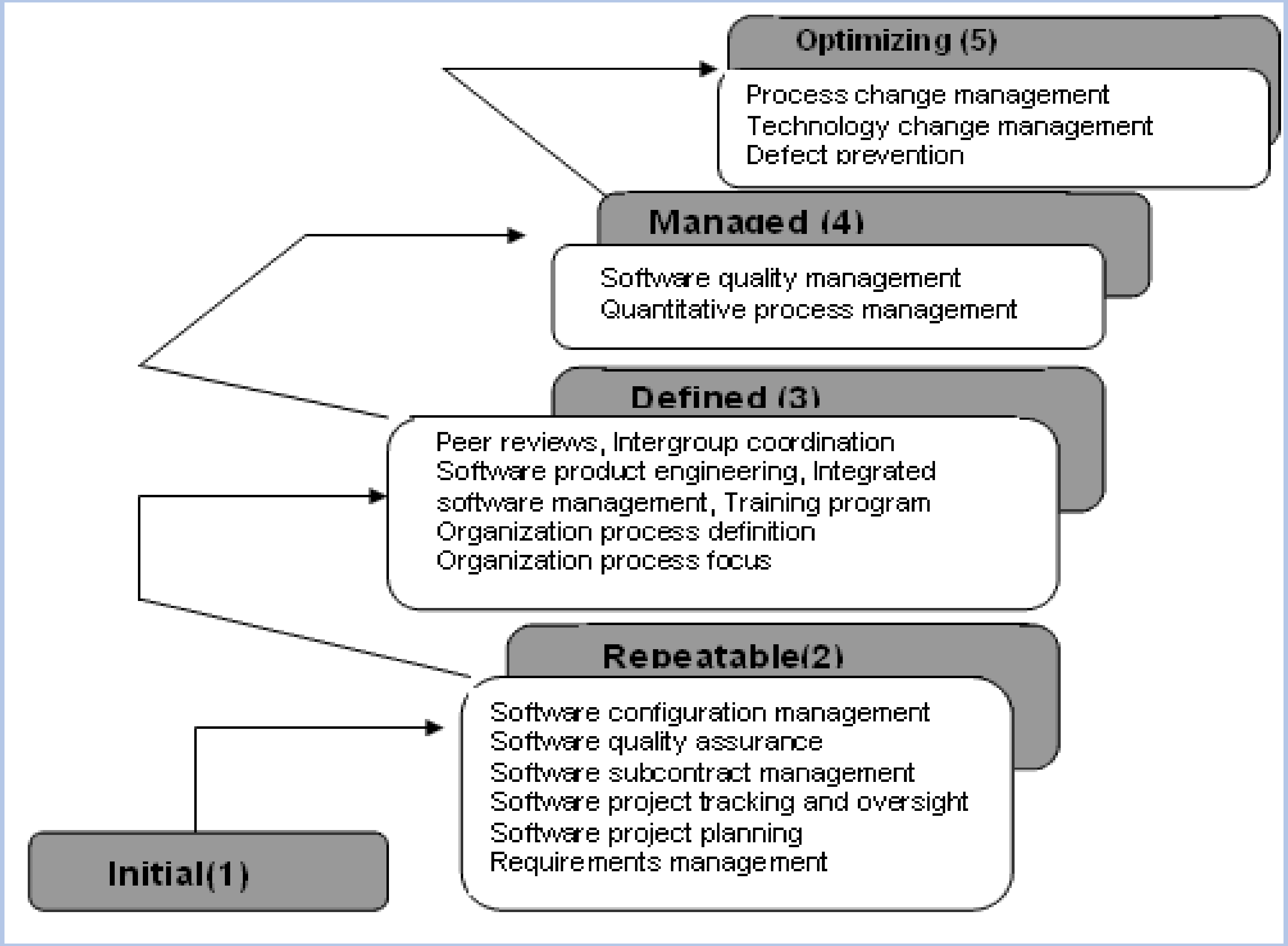
- contains the goals that must be reached in order to improve a software process
- A PA is said to be satisfied when procedures are in place to reach the corresponding goals
- A software organization has achieved a specific maturity level once all the corresponding PAs are satisfied

Key Process Areas (KPAs)

The process areas (PA’s) have the following features

- Identify a cluster of related activities that, when performed collectively, achieve a set of goals considered important for enhancing process capability.
- Defined to reside at a single maturity level.
- Identify the issues that must be addressed to achieve a maturity level

CMM


Optimizing (5)
Process change management
Technology change management
Defect prevention
Managed (4)
Software quality management
Quantitative process management
Defined (3)
Peer reviews, Intergroup coordination
Software product engineering, Integrated
software management, Training program
Organization process definition
Organization process focus
Repeatable(2)
Software configuration management
Software quality assurance
Software subcontract management
Software project tracking and oversight
Software project planning
Requirements management
Initial(1)

The different maturity levels have different process areas pre-defined as shown in the figure

Capability Maturity Models (CMM)

- Level 1: Initial

  The software process is characterized ad hoc and occasionally chaotic

- Level 2: Repeatable

  Basic project management process are established to track cost, schedule, and functionality

- Level 3: Defined

  The software process for both management and engineering activities is documented, standardized and integrated into an organization wide software process

Capability Maturity Models (CMM)

- Level 4: Managed

  Detailed measures of the software process and product quality are collected

  Both SW process and product are quantitatively understood and controlled using detailed measures

- Level 5: Optimizing

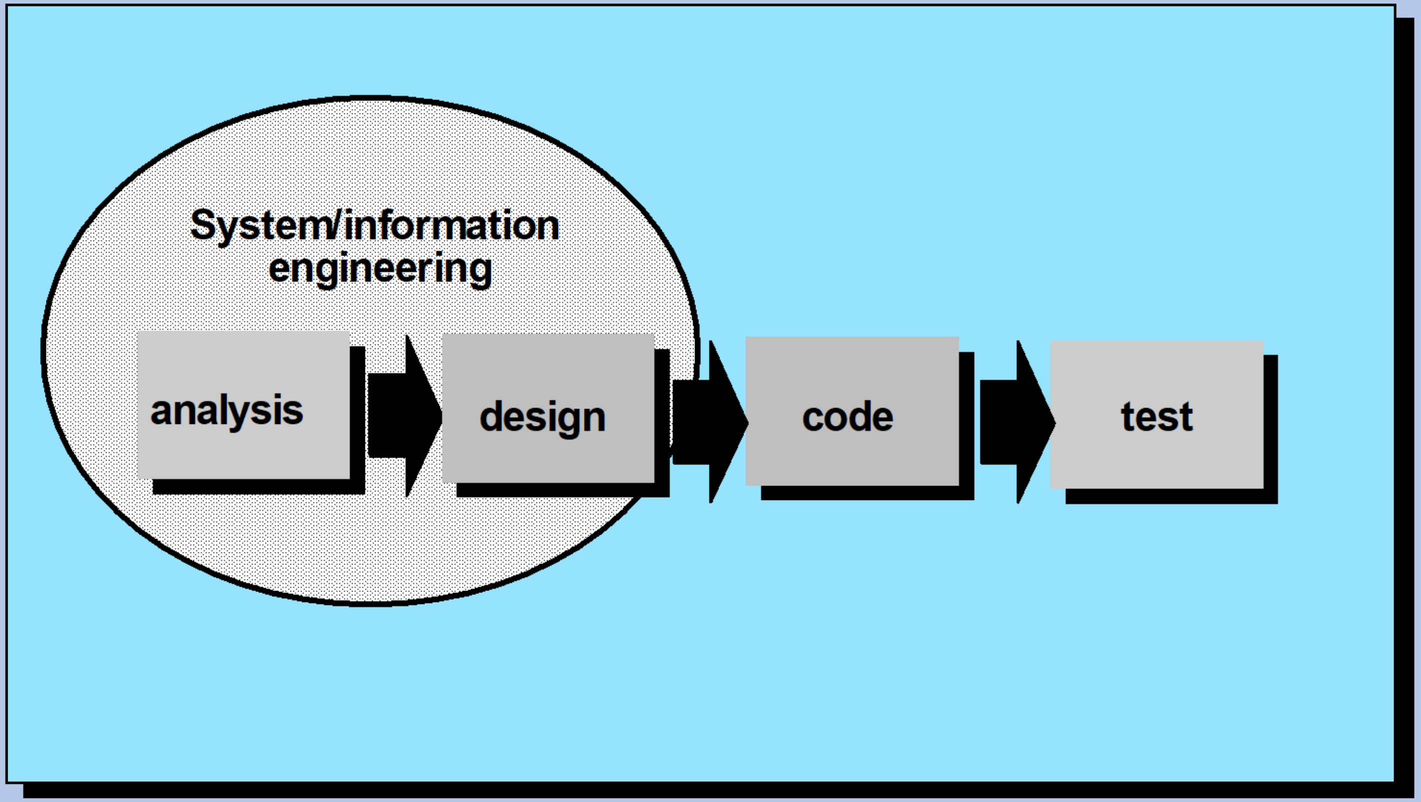
  Continuous process improvement is enabled by quantitative feedback from the process and from testing innovative ideas and technologies

Software Process Models

- A software engineer must incorporate a development strategy that encompasses the process methods, and tools layers
- This strategy is often referred to as a process model or a software engineering paradigm
- A process model is chosen based on the nature of project and application, the methods and tools to be used, and the controls and deliverables that are required

1. The Linear Sequential Model

- Sometimes called the classic life cycle or the waterfall model



1. The Linear Sequential Model

Activities:

- System/information engineering and modeling
  - Requirements gathering
  - Interface with other elements: HW, SW, database
- Software requirement analysis
  - Understand nature of SW to be built
  - Function, behavior, performance and interface
  - Documented and reviewed with the customer

1. The Linear Sequential Model

Activities:

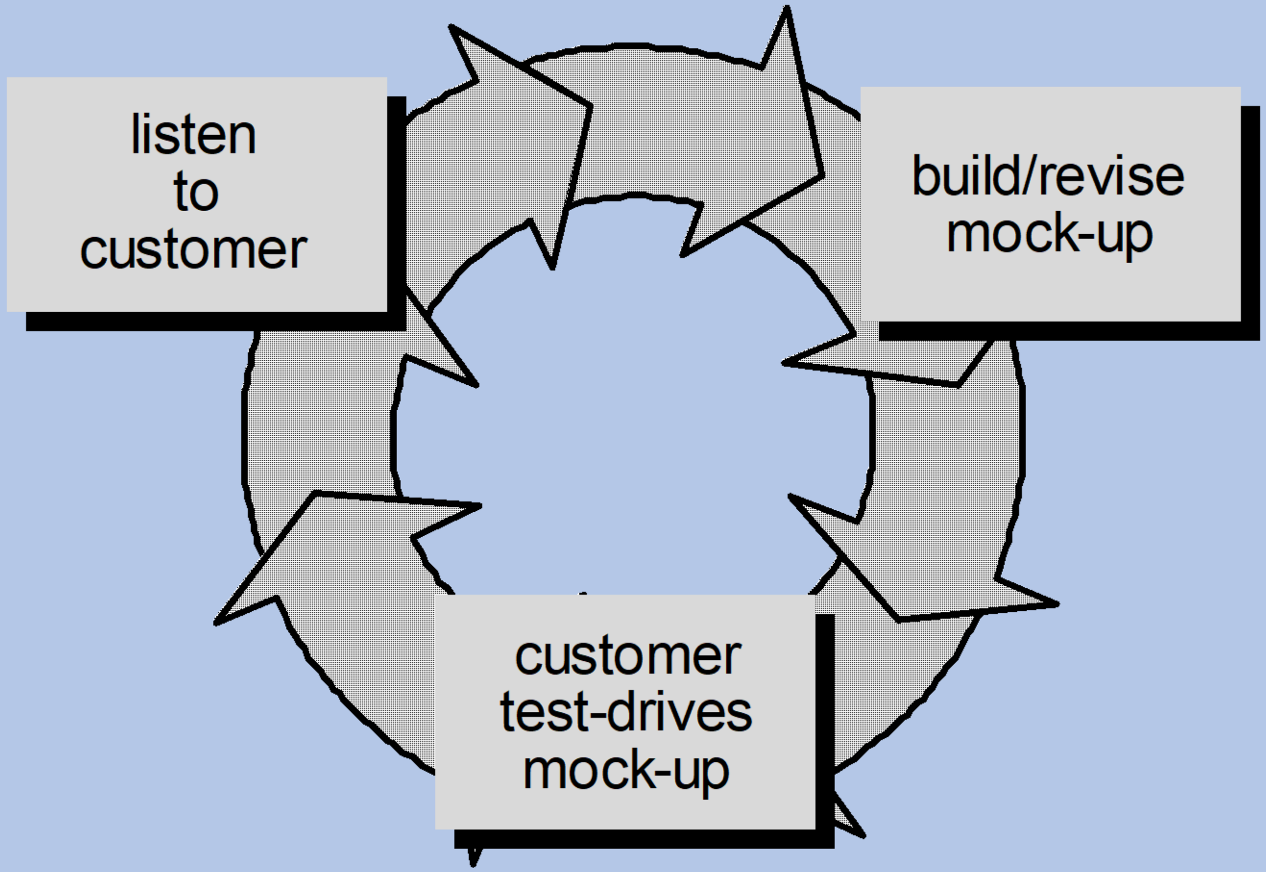
- Design:
    - A multi-step process focuses on data structure, SW architecture, interface representations, and procedural (algorithmic) detail
    - The design process translates requirements into a representation of the SW that can be assessed for quality before coding begins
    - Documented

1. The Linear Sequential Model

Activities:

- Code generation
    - Design converted in machine readable form
- Testing
    - Test all functions, both internal/external
    - Uncover errors
- Support/maintenance
    - Deal with all kinds of changes that may be occurred after SW delivery

2. The Prototype Model


listen to customer
build/revise mock-up
customer test-drives mock-up