CENG 111

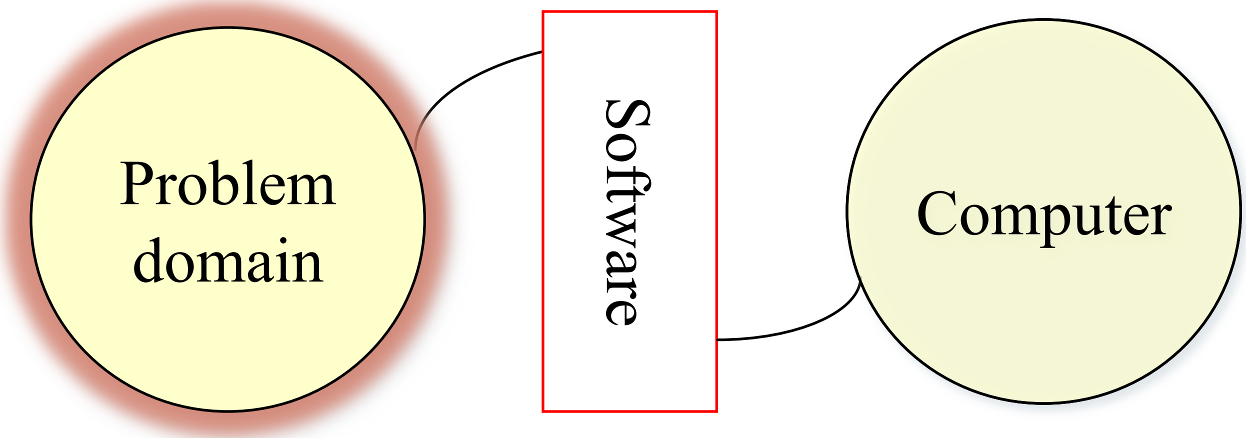
Lecture – Software Engineering

Outline

- What is Software Engineering
  - What is software
  - Art, Science, Engineering
  - In the context of Computing
  - Essential
  - A short history
- Software Engineering Life Cycles and Processes
  - Ad hoc, Waterfall, Evolutionary
  - Software Engineering Life Cycle Processes
- Concepts – Software Architecture
  - Abstractions, tiers, layers, coupling, cohesion

What is Software

- Software is an interface between the problem domain and the computer



- Software engineering is the conduct of software process that is, applying means to produce and maintain a solution to a real-world problem in the software domain.

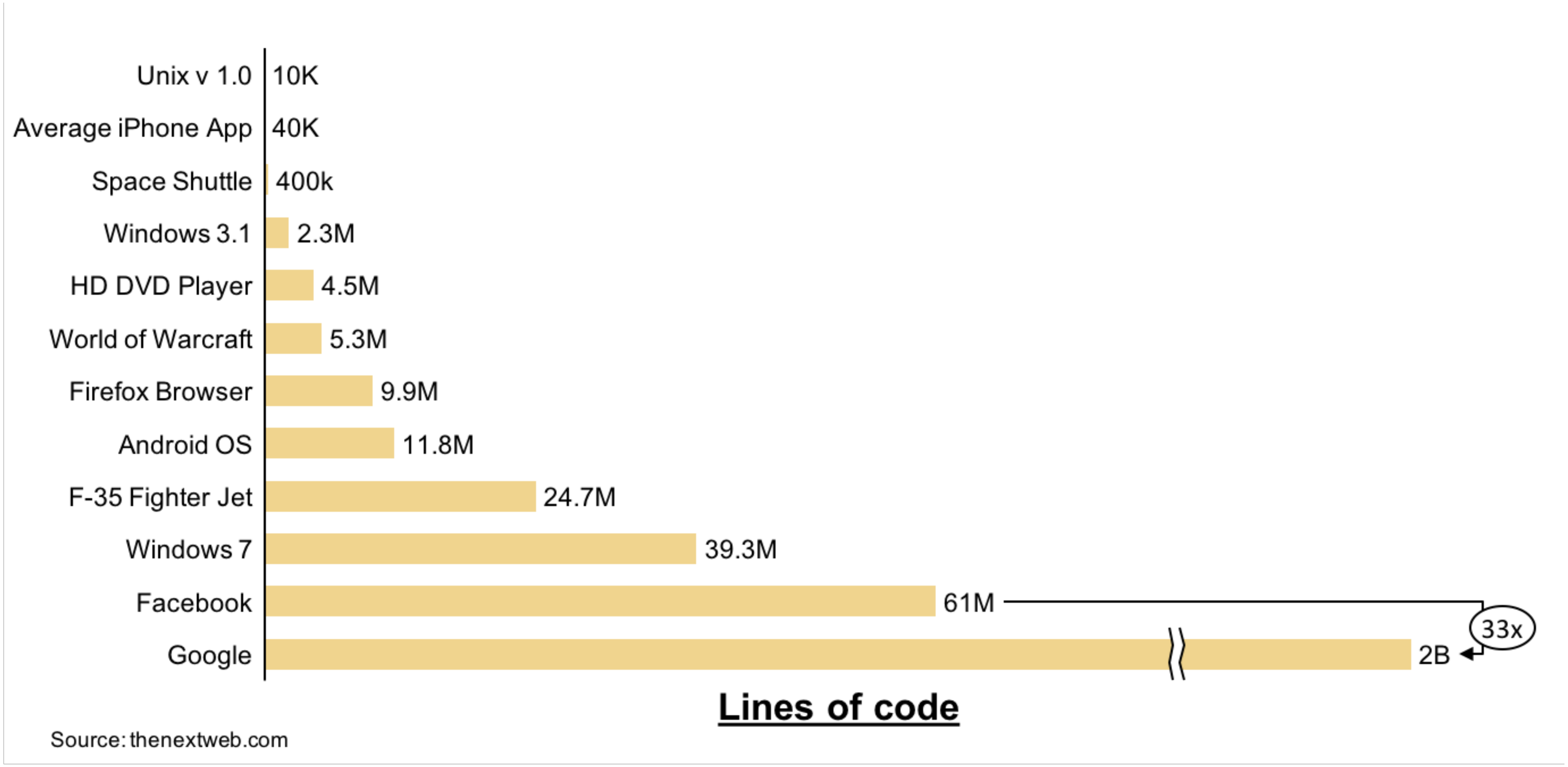
Why ?

- The demand for software products are increasing exponentially.
- Expectations from software products are increasing exponentially.

"The only thing you can do with an F-22 without its software is to take its picture"

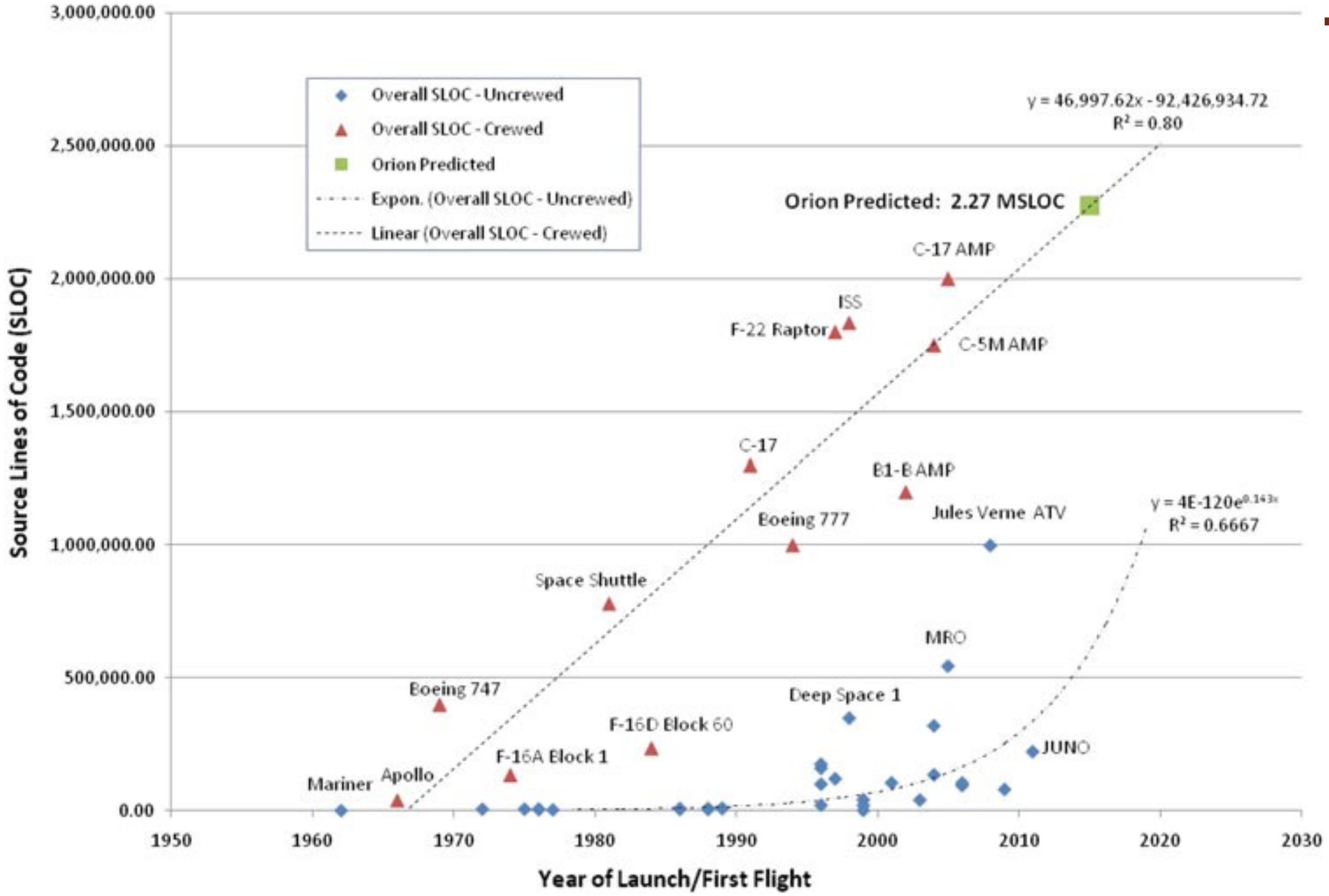
F-22 Program Director

Software is Everywhere



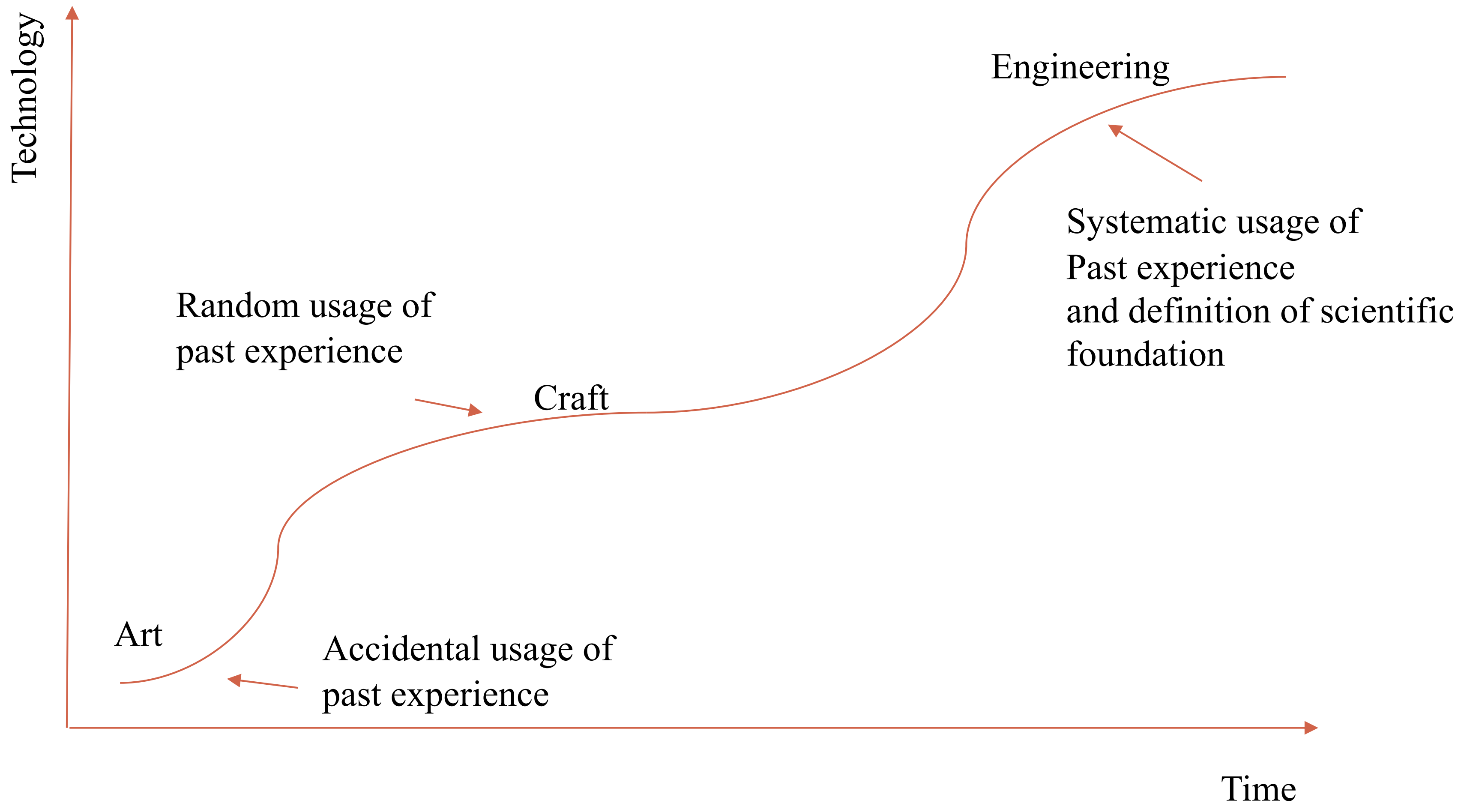
Source: thenextweb.com

Software Size Growth – Space Missions



* Judas, P., Prokop L.A, Historical Compilation of Software Metrics, Innovations in Systems Engineering, 2011

Art or Engineering

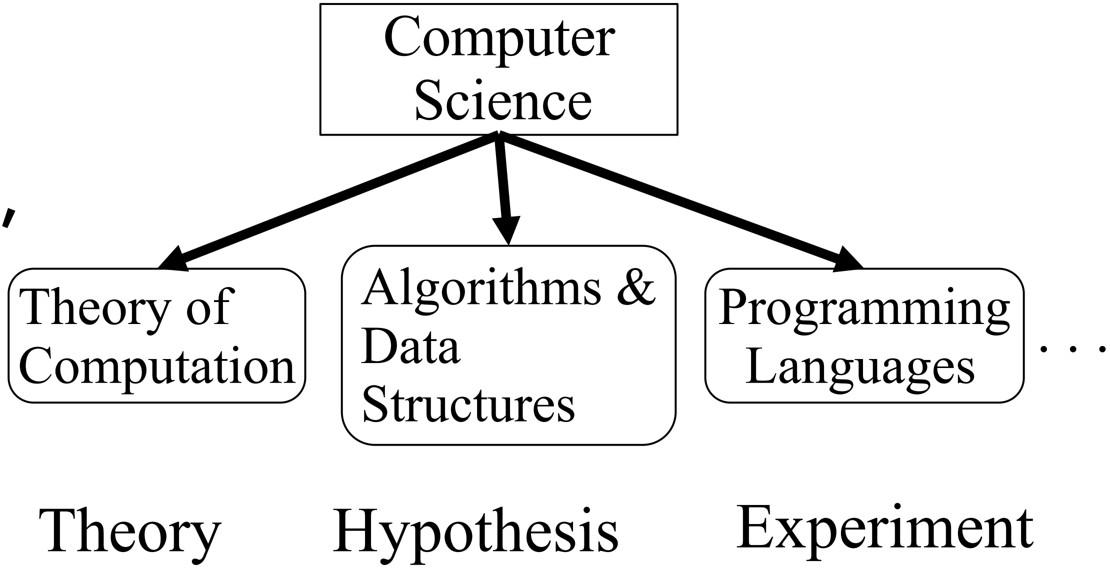


Science and Engineering

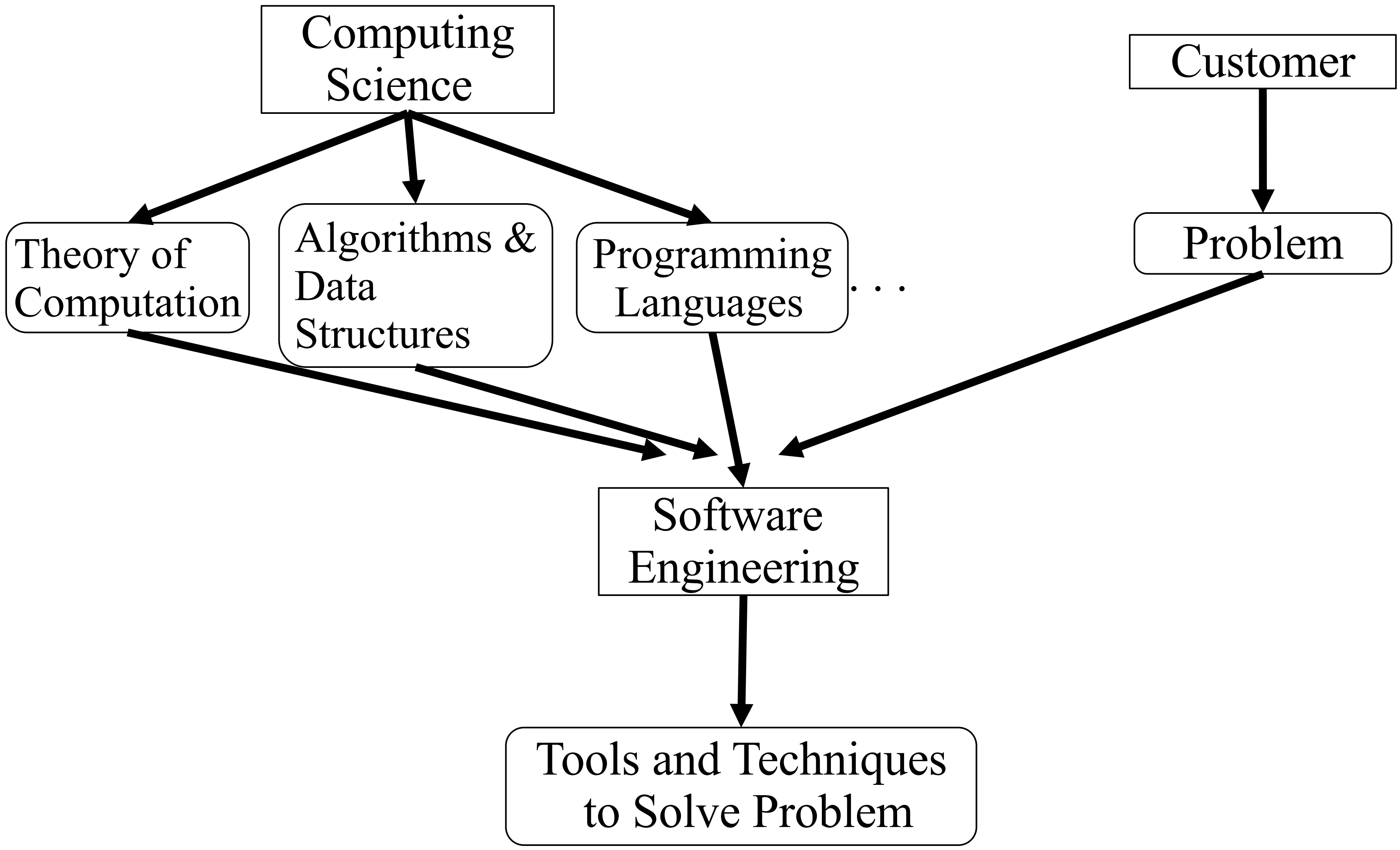
- Science is a systematic enterprise that builds and organizes knowledge in the form of testable explanations and predictions about the universe.
  - For example, biology is the natural science that studies life and living organisms
- Engineering is the creative application of science, mathematical methods, and empirical evidence to the innovation, design, construction, operation and maintenance of structures, machines, materials, devices, systems, processes, and organizations for the benefit of humankind.
  - For example, petrolium engineering might use the results of chemistry to come up with a better way of refining gasoline

Computer Science

- Theory – testable explanations:
  - Computability, automata, discrete computational structures
  - Algorithm analysis
- Hypothesis – predictions:
  - That a certain algorithm will solve a problem
- Experiment – testability:
  - Run a program implementing the algorithm



Software Engineering and Computer Science



Computing Science

Theory of Computation

Algorithms & Data Structures

Programming Languages

. . .

Customer

Problem

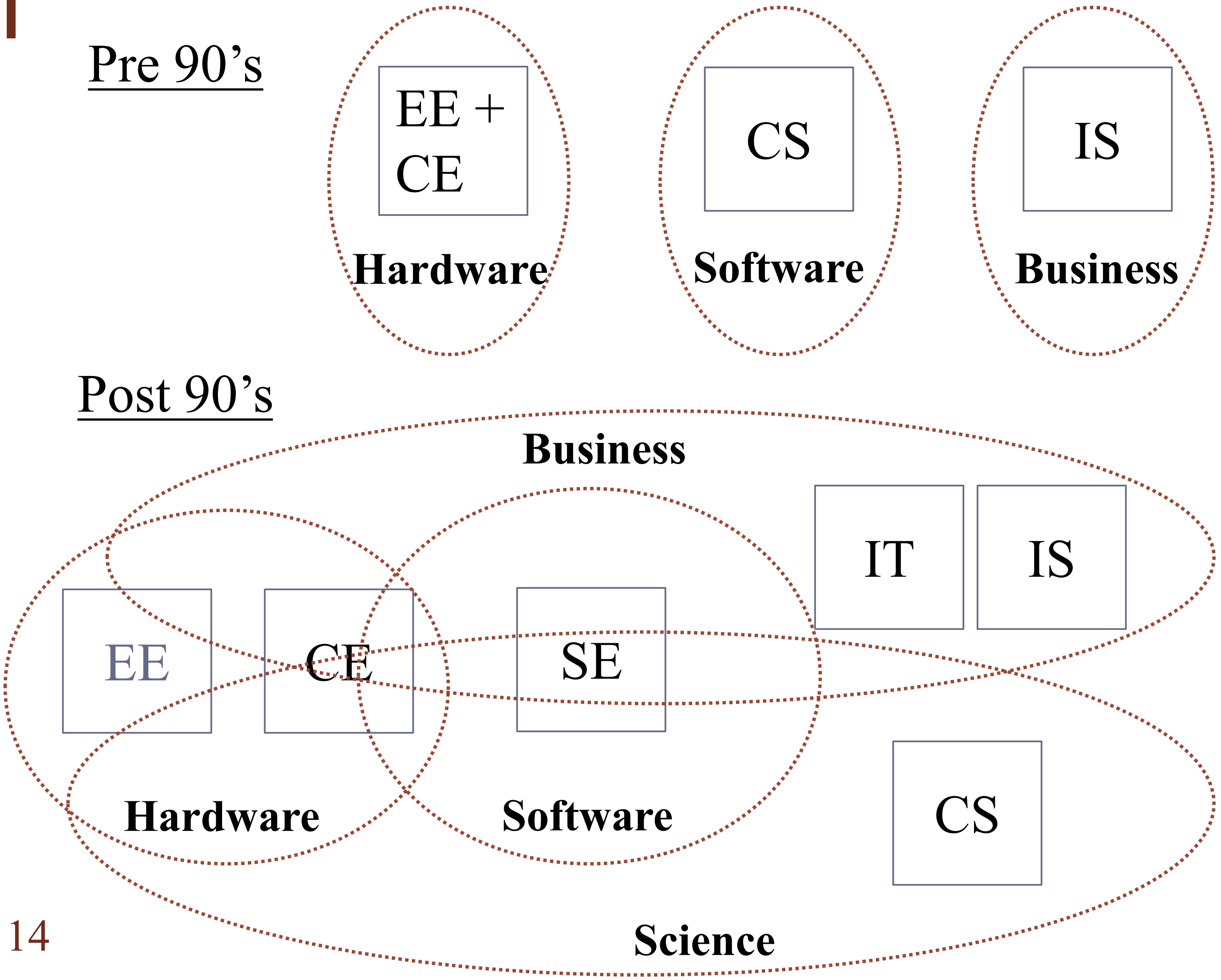
Software Engineering

Tools and Techniques to Solve Problem

Computing Disciplines

- Computer science
  - concerns with the theory and experimentation that form the basis for the design and use of computers and applications on them.
- Computer engineering
  - concerned with the design of digital systems including communications systems, computers and devices that contain computers.
- Information systems
  - concerned with the information that computer systems can provide to aid an organization in defining and achieving its goals.
- Software engineering
  - concerned with developing and maintaining software systems that behave reliably and efficiently, are affordable to develop and maintain, and satisfy all the requirements that customers have defined for them.

Computing Disciplines



- CS: Computing Science
- CE: Computer Engineering
- SE: Software Engineering
- IS: Information Systems
- IT: Information Technology

- EE: Electrical Engineering

What is Software Engineering

- Application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software; that is, the application of engineering to software.

“systematic”

- Following a well-defined sequence of activities,
  - in which desired outputs (deliverables) are well-defined
  - by using well-defined inputs
  (i.e. documented syntax, semantics, context and other relevant properties of the input)
  - in a well-defined process
  (e.g. using organizational standards for interprocess communication, data formats, error handling etc.)
  - whose outputs are in turn used similarly as inputs in subsequent process(es),
  - until the final output is achieved,
  - and where the correctness of the output is verifiable.

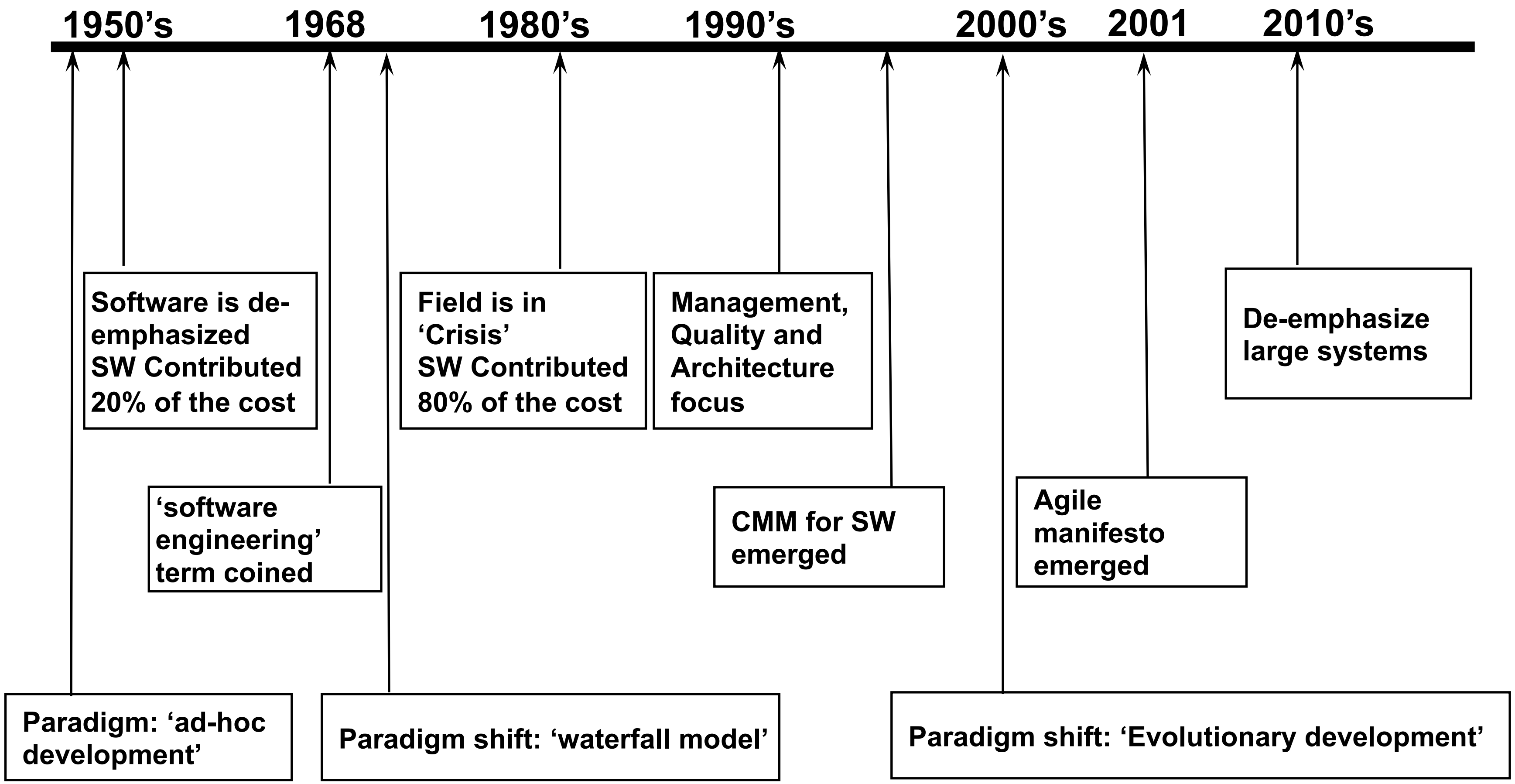
“disciplined”

- Each process is followed using organizational principles (e.g. who manages whom, who is responsible for what?),
- Intermediate results are carefully documented, as well as final results,
- Actions are traceable as to their causes, individuals involved, time of occurrence and circumstances.

“quantifiable”

- Measured and predictable
  - The size and extent of the required effort
    (size of output code, data, documentation, manpower, duration, budget for development, expected error rate and user support)
  - The quality of the work products and processes

Software Engineering Timeline



1950's

1968

1980's

1990's

2000's

2001

2010's

Software is de-emphasized
SW Contributed 20% of the cost

'software engineering' term coined

Field is in 'Crisis'
SW Contributed 80% of the cost

Management, Quality and Architecture focus

CMM for SW emerged

Agile manifesto emerged

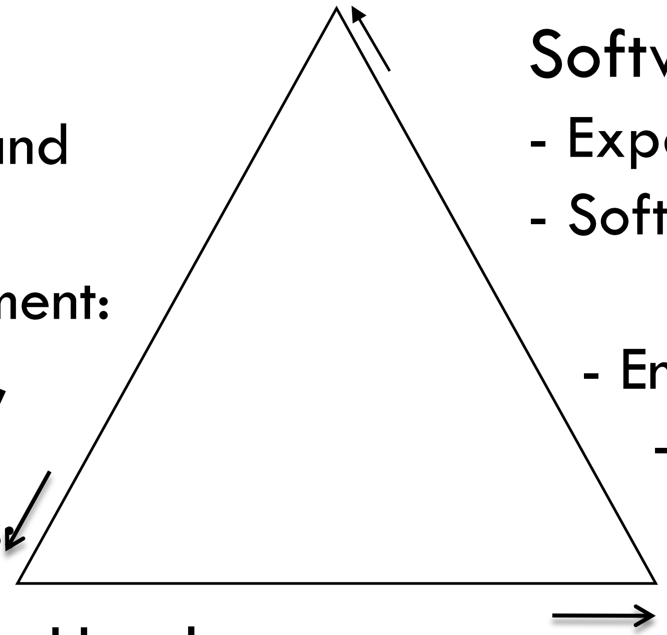
De-emphasize large systems

Paradigm: 'ad-hoc development'

Paradigm shift: 'waterfall model'

Paradigm shift: 'Evolutionary development'

Hardware, Software, Services



Services:

- Cheap hardware and storage
- Software development: Agile, decentralized, small teams
- Beautiful programs: robust, anti-fragile

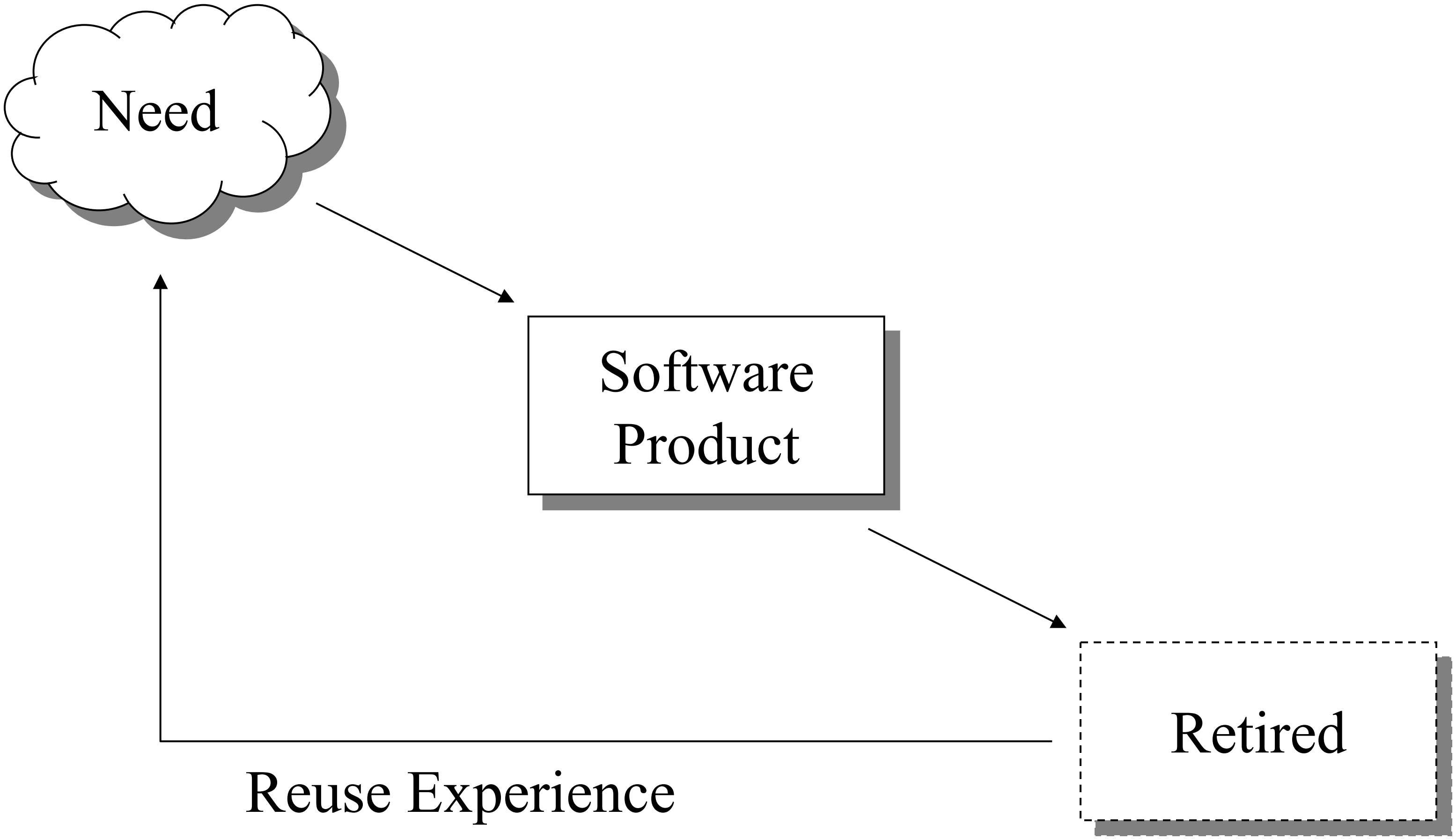
Software:

- Expensive software
- Software development: systematic, process based
- Engineering approach
- Beautiful programs: other programmers can understand.

Hardware:

- Cheap software
- Software development: individual activity
- Beautiful programs: utilize the hardware elegantly.

Software Life Cycle



Need

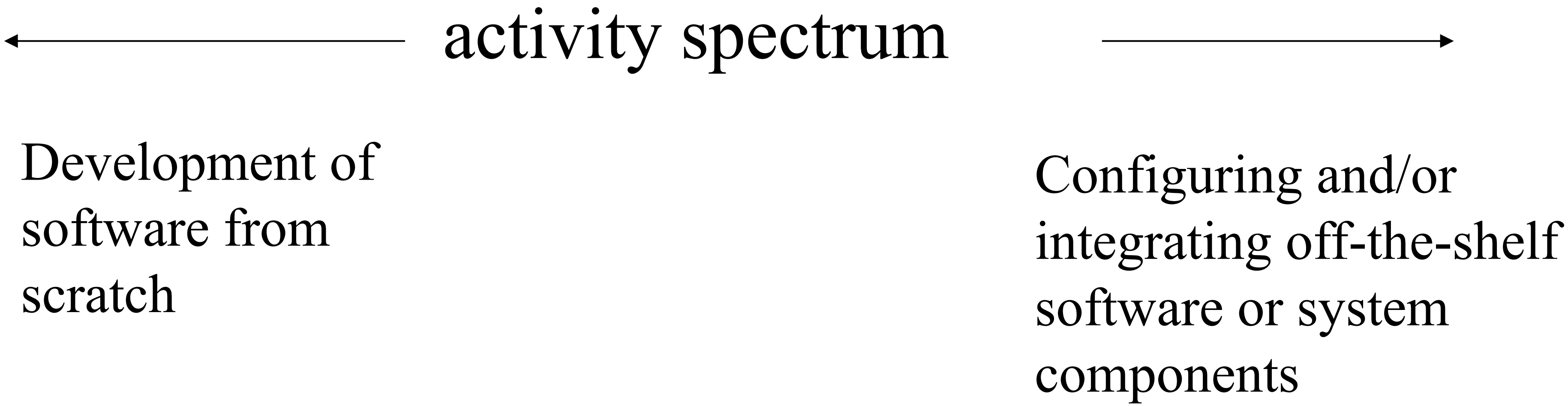
Software Product

Retired

Reuse Experience

The Software Process Activities

- The activities may involve the development of software;
  - from scratch in a programming language,
  - by extending or modifying existing systems,
  - by configuring and integrating off-the-shelf software or system components.



activity spectrum

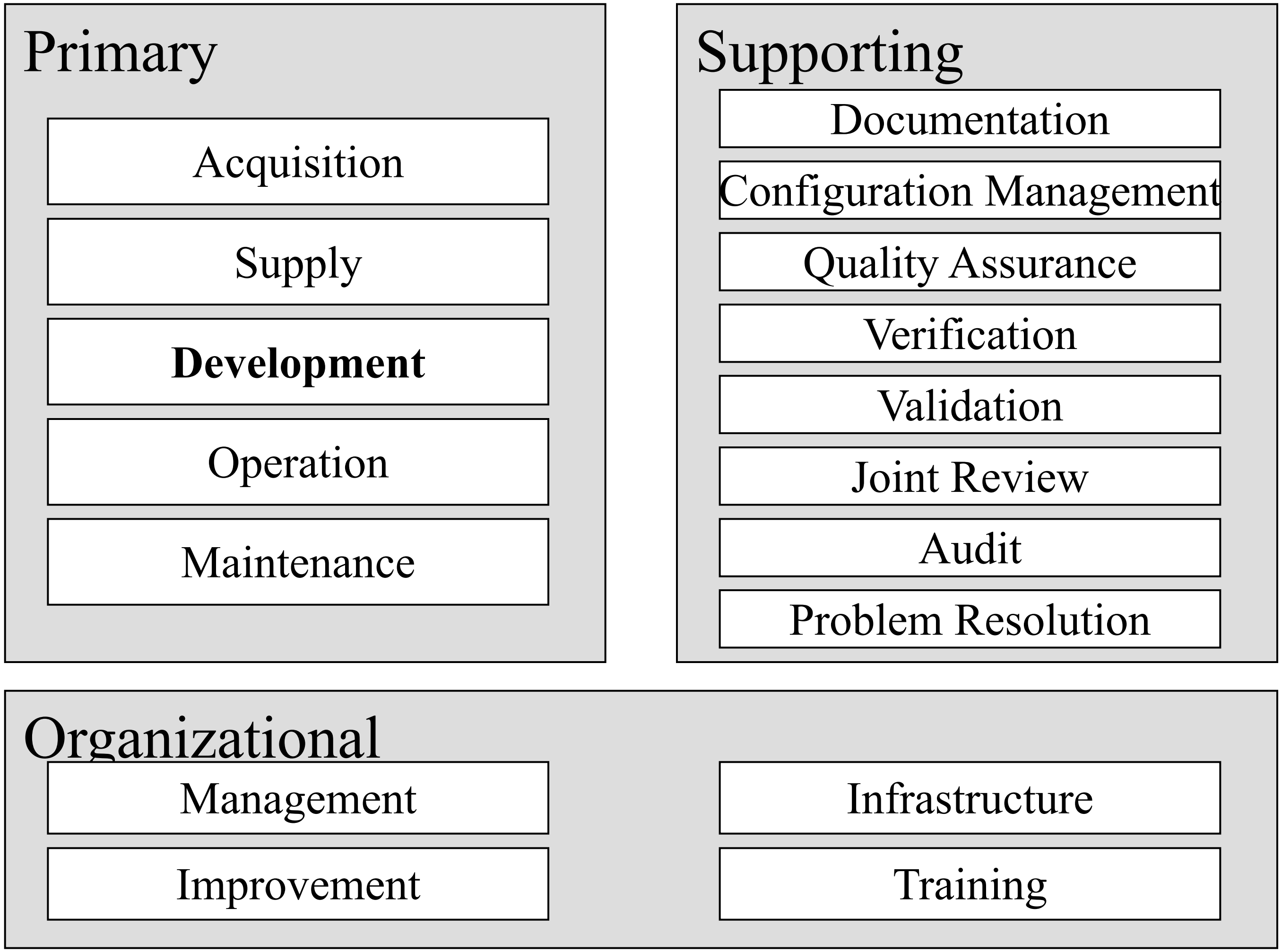
Development of software from scratch

Configuring and/or integrating off-the-shelf software or system components

The Software Life Cycle Process

- ISO/IEC 12207:1995 (or corresponding IEEE/EIA 12207:1996) establishes a common framework for software life cycle processes
- IEEE/IEA 12207: 1996 includes the original ISO/IEC 12207:1995 as well as guidelines for application
- Newer versions:
  - ISO/IEC 12207:2008, which was published in February 2008
  - ISO/IEC/IEEE 12207:2017 is the newest version, published in November 2017

ISO/IEC12207 Software Life Cycle Processes



Primary

Acquisition

Supply

Development

Operation

Maintenance

Supporting

Documentation

Configuration Management

Quality Assurance

Verification

Validation

Joint Review

Audit

Problem Resolution

Organizational

Management

Improvement

Infrastructure

Training

Software Development Process

- Software development process is a subset of processes that a software organization perform.
- Describe what is intended to be done, that is the mode of operation, without having to provide all implementation details
- It is a roadmap that helps you create a timely, high-quality software
  - It describes the structure within which software engineers operate

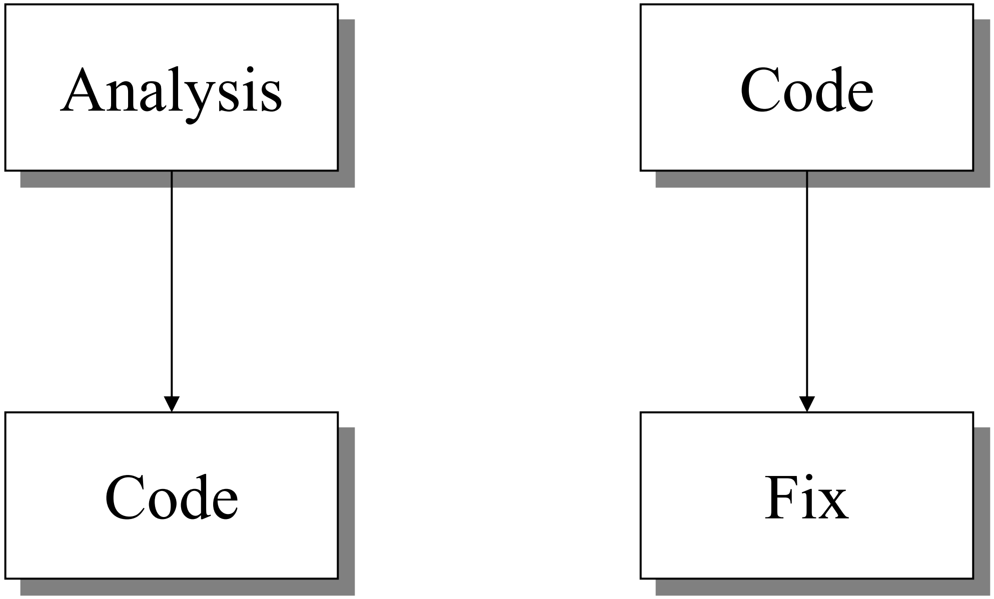
Software Development Process

- System requirements analysis
- System architectural design
- Software requirements analysis
- Software architectural design
- Software detailed design
- Software coding and testing
- Software integration
- Software qualification testing
- System integration
- System qualification testing
- Software installation
- Software acceptance support

Software Development Life Cycle Models

- Code and Fix
- Basic Waterfall
- Evolutionary
- Incremental
- Spiral
- Rapid Application Development
- Formal Systems Development
- Reuse Based Development
- Component Based Development
- ….

Analyze and Code / Code and Fix



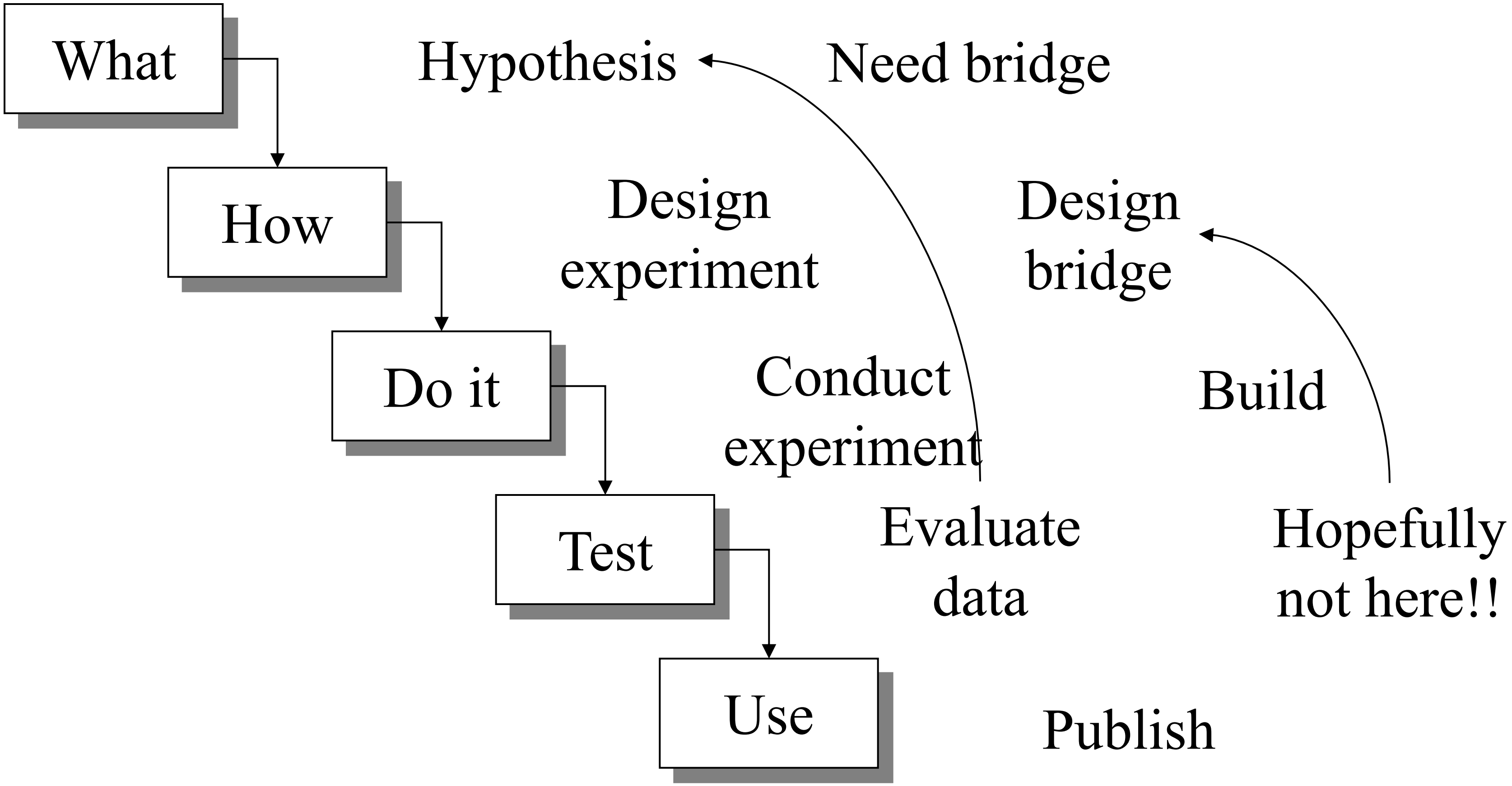
Advantages

- All that is required if the effort is sufficiently small and if the final product is to be used by the one who built it
- Most customers are happy to pay, since effort in both steps directly contributes to the product

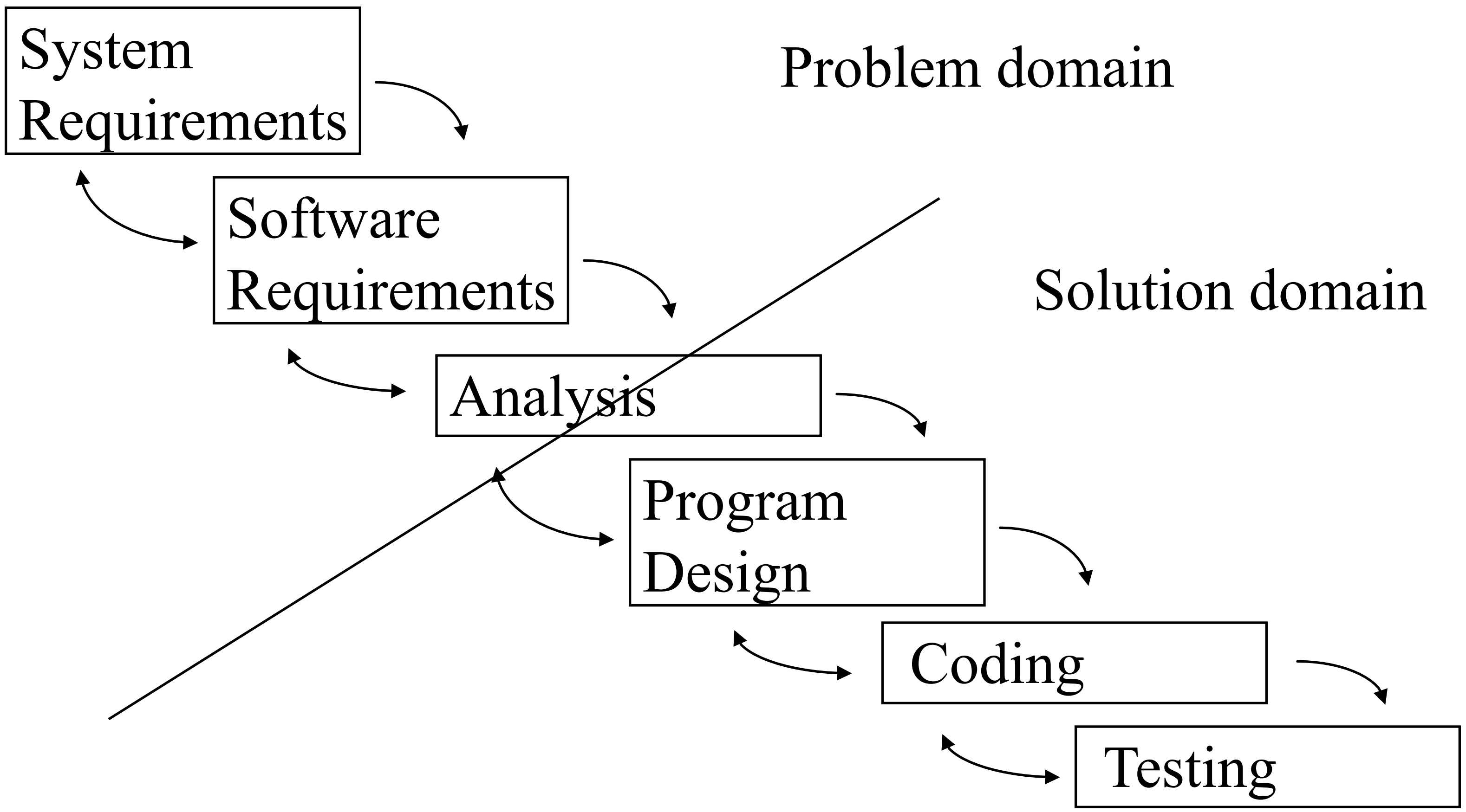
Difficulties

- After a number of fixes, the code becomes poorly structured
- Difficult to maintain because of poor preparation for modification
- No customer involvement can result in poor match to users’ needs
- Suitable for individual programmers but not for teams

Basic Problem Solving Process



Basic Waterfall Model



System Requirements

Software Requirements

Analysis

Program Design

Coding

Testing

Problem domain

Solution domain

Advantages

- There is an iteration with the preceding and succeeding steps but rarely with the more remote steps
- The change process is divided into manageable units
- After the requirements step is complete there exist a firm baseline
- Maximize the extend of early work

Difficulties

- The testing phase occurs at the end of the development cycle. The errors might signal major redesign.
- If the product is totally original, some experimental tests might be required before building the system.
- What a software is going to do is subject to wide interpretation even after previous agreement.
  - Requirements frequently changes as the system developed.
- Emphasis on documentation as completion criteria:
  - Possible for some problem domains (compilers, OS, etc.) difficult for others such as interactive end-user applications.
    - 1M$ HW -> 30 pages specification
    - 1M$ SW -> 1500 pages specification
- There are many other feedback points beside the end of the phase.

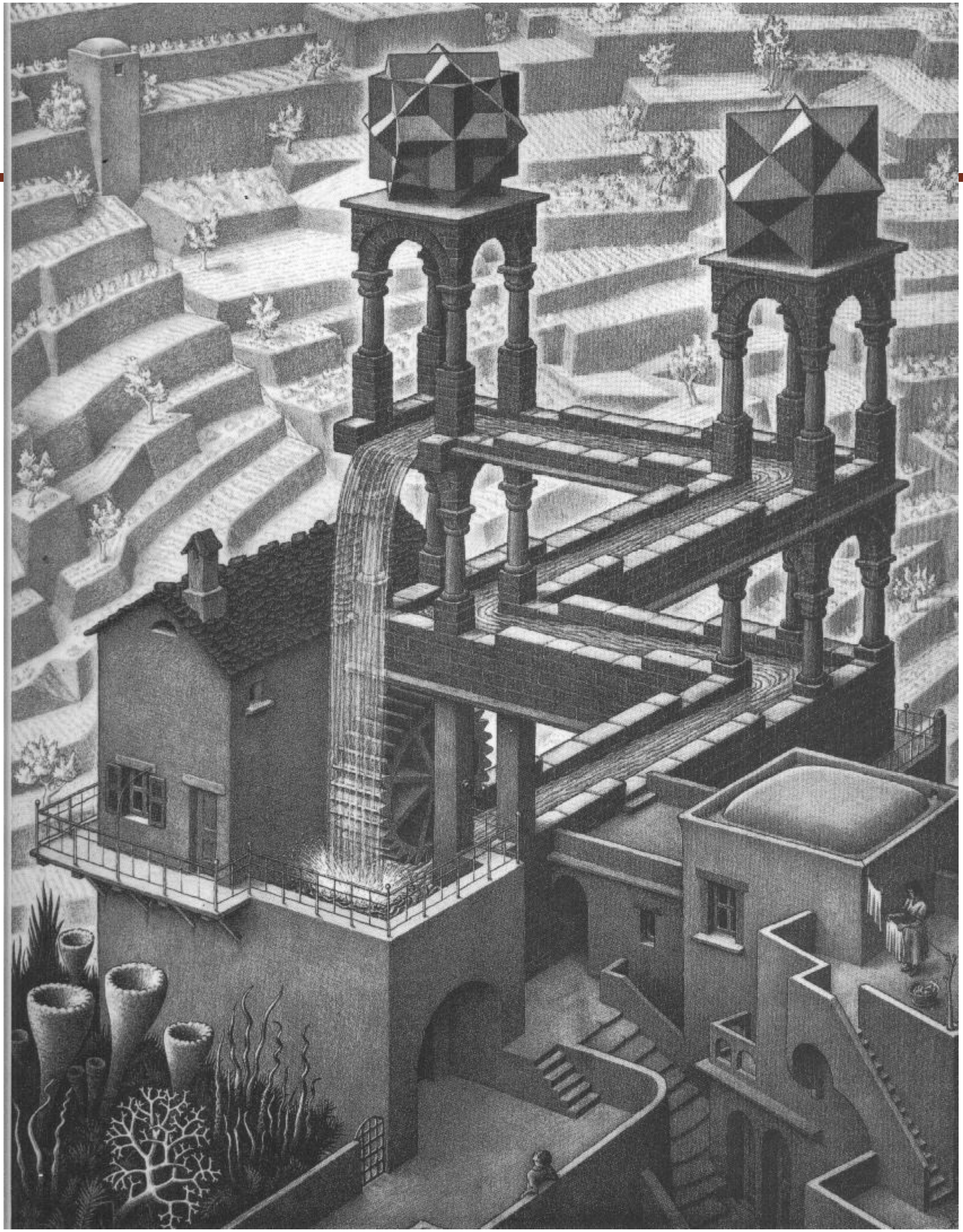
HW & SW Development Processes

- HW Process
  - HW Requirements
  - Preliminary Design
  - Detailed Design
  - Manufacture
  - Test
  - Operations & Maintenance

- SW Process
  - SW Requirements
  - Preliminary Design
  - Detailed Design & Code
  - Duplication of CDs
  - Test
  - Operations & Maintenance

Hardware vs. Software

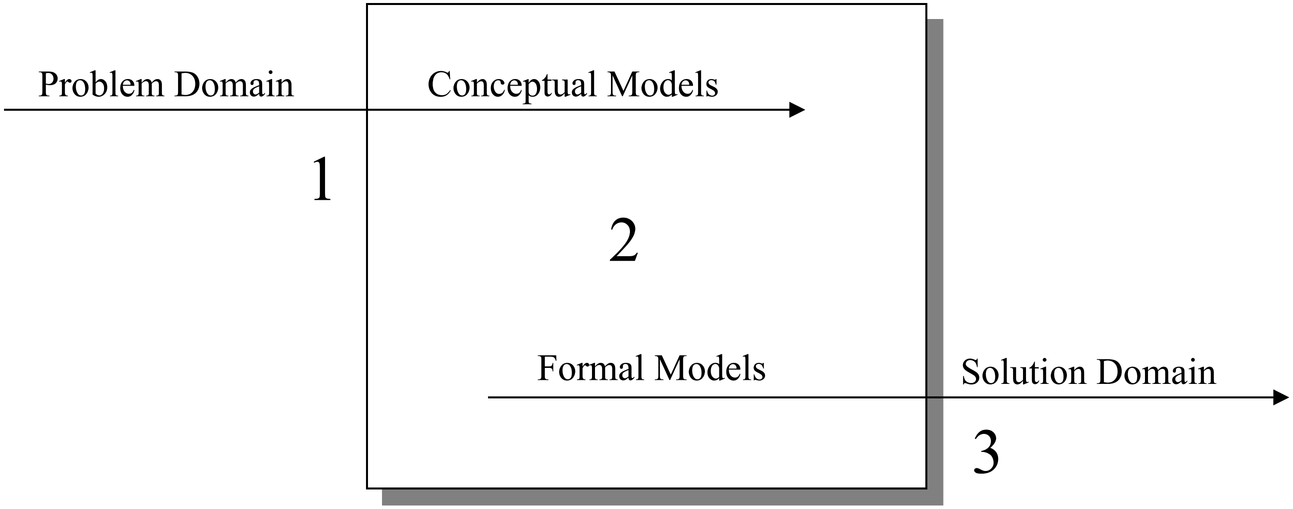
- Hardware requires manufacture, whereas software does not
  - HW: Focus on manufacture
  - SW: Focus on Design & Code
- Hardware has physical models to use in evaluating design decisions
- Hardware is composed of parts that wear out, whereas software does not degrade from use
  - HW Reliability: a measure of failure of components
  - SW Reliability: a measure of number of remaining defects



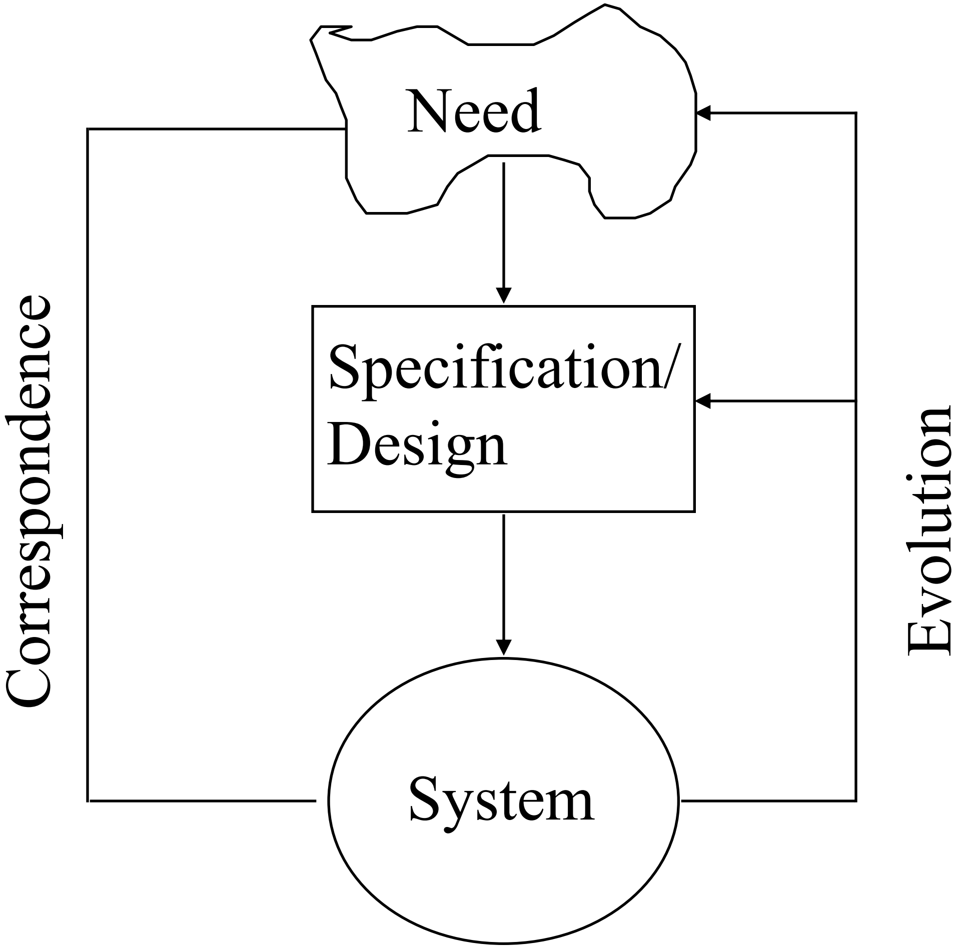
Waterfall - 1961 (M.C. Esher)

The Inherent Process

- In effect we transform an abstract need to a successively more solid product
- As we move from the definition of the need to a software solution we move from problem domain to solution domain
  - The process takes place in at least two different domains
  - Knowledge of the problem domain is equally important to the knowledge of the solution domain (software tools etc.)



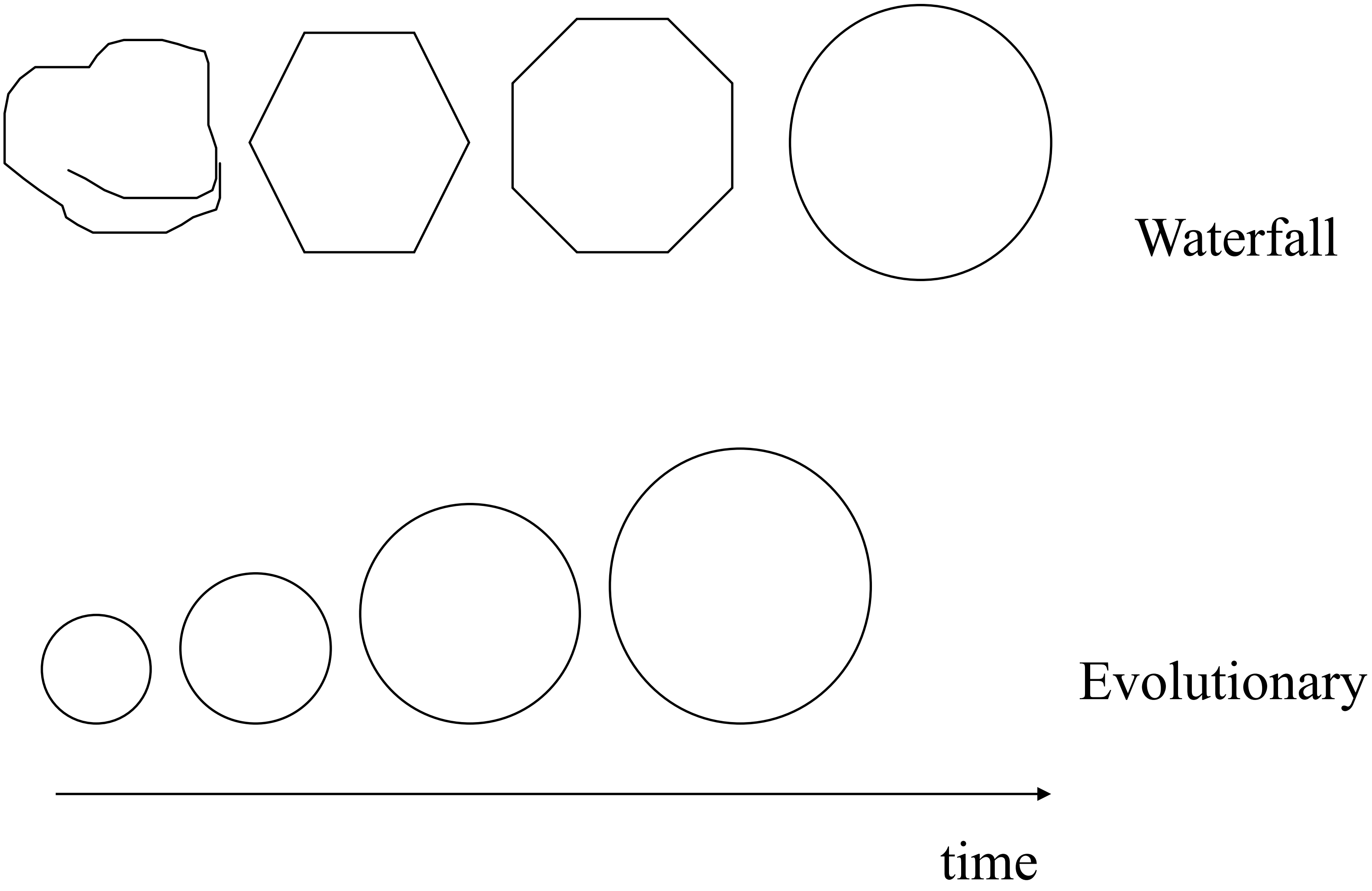
Evolutionary Models



Evolutionary Development

- Develop an initial system according to outline requirements
  - Development starts with the parts of the system which are well understood
- Evolve the system through multiple versions by customer feedback
- Specification, development, and validation activities are carried out concurrently
- Objective: Work with the customer to explore their requirements and deliver a final system

Output Progression



Waterfall

Evolutionary

time

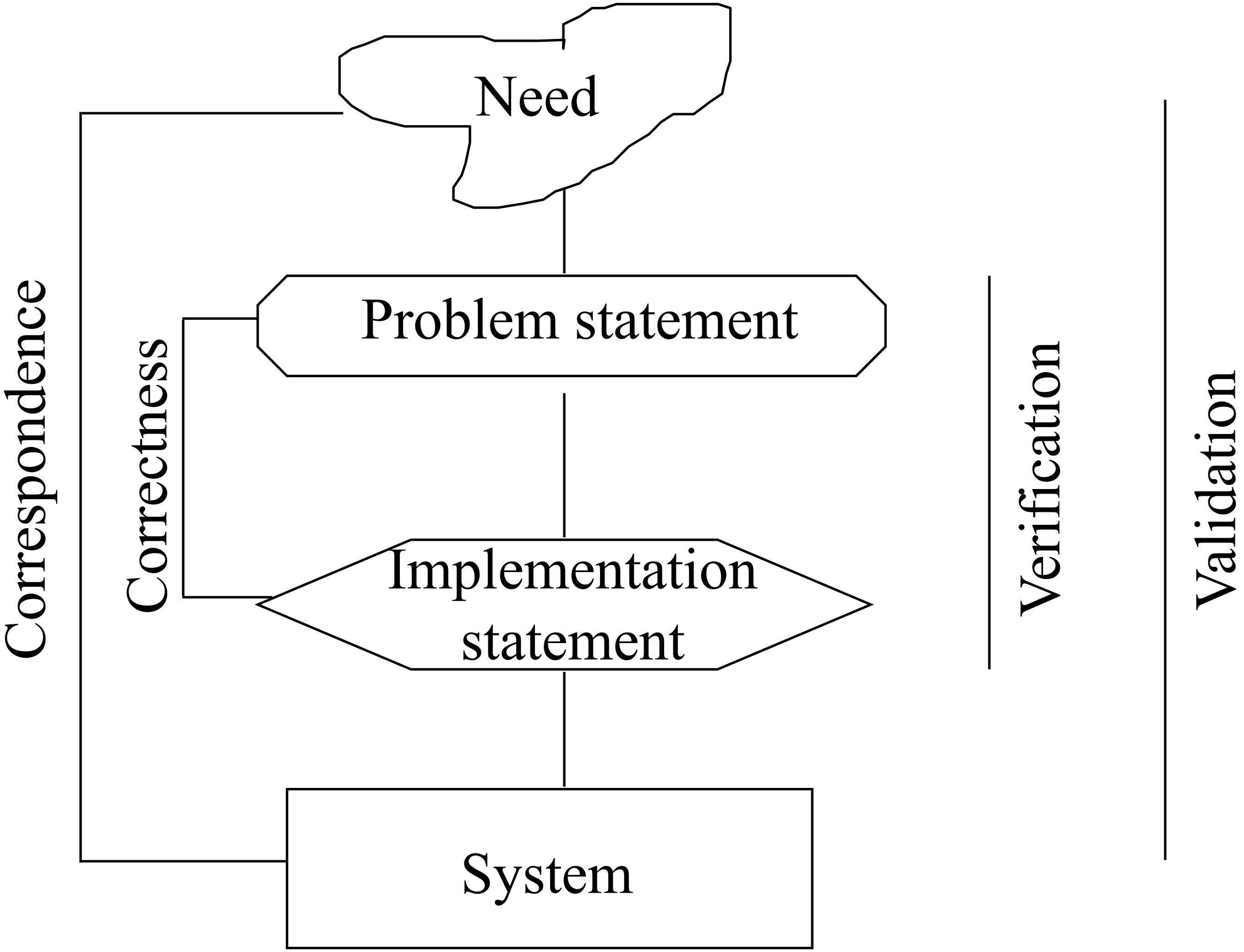
Advantages & Difficulties

- Advantage
  - Enables users to better understand their needs
  - Problems are detected earlier
- Difficulties
  - Process is not visible
  - Software deteriorates as changes are made
  - Special tools/techniques and skilled personnel are required

Applicability

- Suitable for small or medium sized systems and short-lifetime systems
- Parts of the larger system which are difficult to specify (such as user the interface) may be developed using evolutionary development

The Generic Process



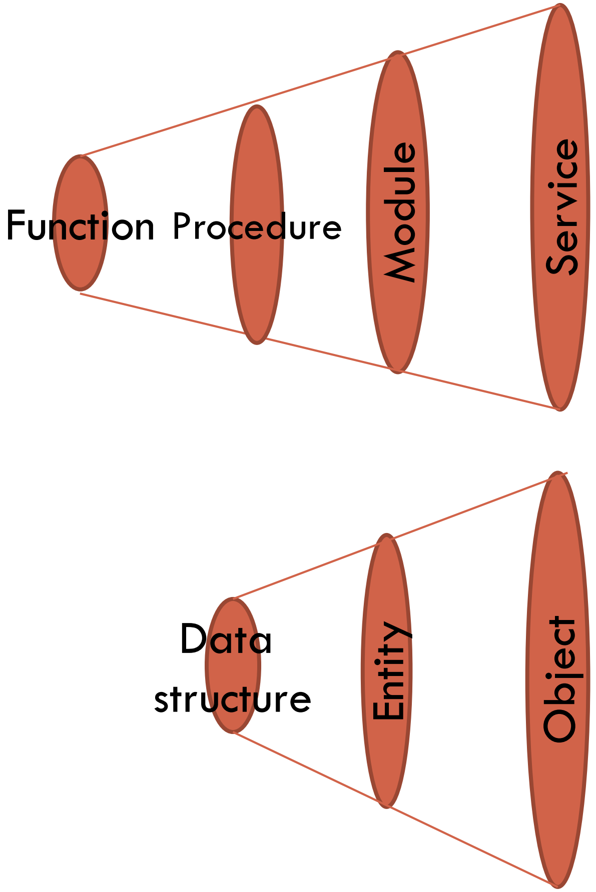
Validation & Verification

- Verification
  - “Am I building the product right?”
  - Determine correctness with respect to its specification; Can begin after specification
  - Objective (if specification is formal)
- Validation
  - “Am I building the right product?”
  - Predict correspondence; Can begin as soon as project starts
  - Subjective

Separation of Concerns

- Separation of concerns:
  - To solve a large problem we decompose it into smaller related pieces.
  - Each of these pieces addresses a concern or a specific part of the problem.

Decomposition Software Abstractions



- Programming abstractions:
  - Procedures
  - Modules
  - Objects
  - Components
  - Services
- Hide the details
- Decompose systems into procedures, objects, or services

Coupling and Cohesion

- Cohesive

  To have all the related code together

  - Single responsibility principle (of Robert Martin):

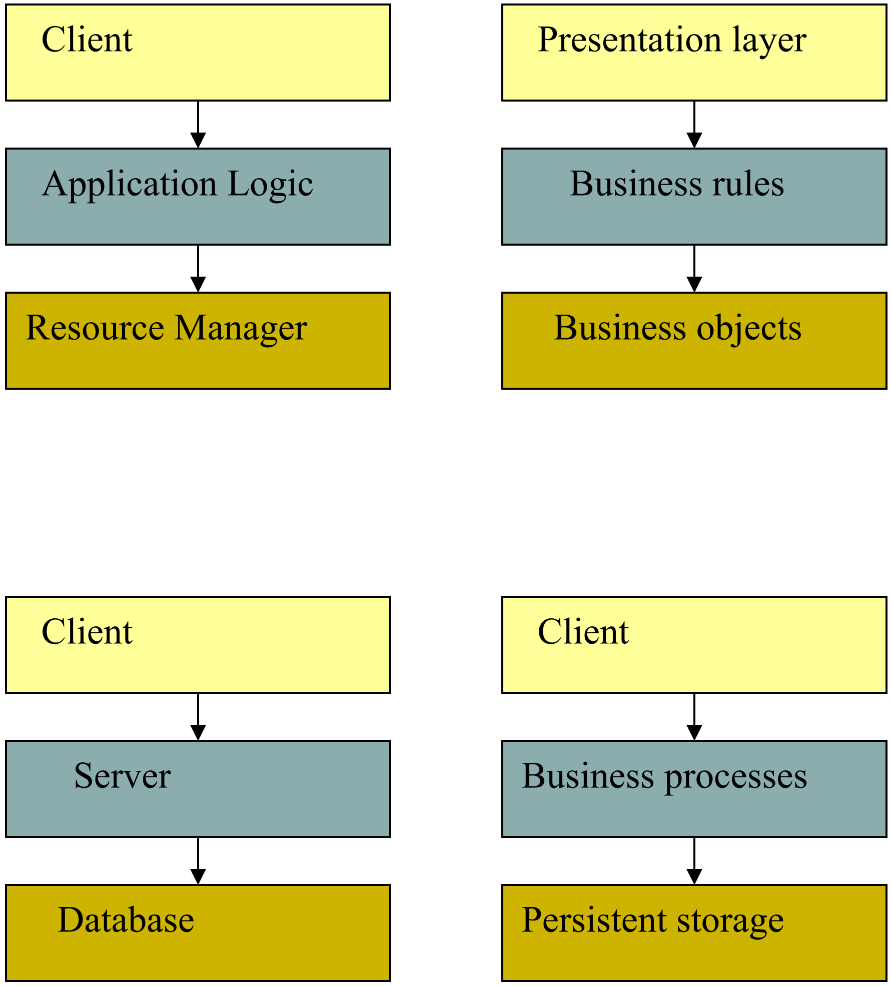
  ‘Gather together those things that change for the same reason, and separate those things that change for different reasons’

- Loosely Coupled
  - Components has, or makes use of, little or no knowledge of the definitions of other separate components
  - Can be changed independently and be deployed by themselves

Layers

- Layer is a construct to logically separate the functionality of an information system.
  - Separation of concerns in relation with the underlying technology
  - If we decompose the software into different conceptual parts we can evolve them separately
- Services and Objects enable us to decompose the software in relation with the concepts of the application domain
- Layers enables us to decompose the software independent from the properties of the application domain.

Layers



Client

Application Logic

Resource Manager

Presentation layer

Business rules

Business objects

Client

Server

Database

Client

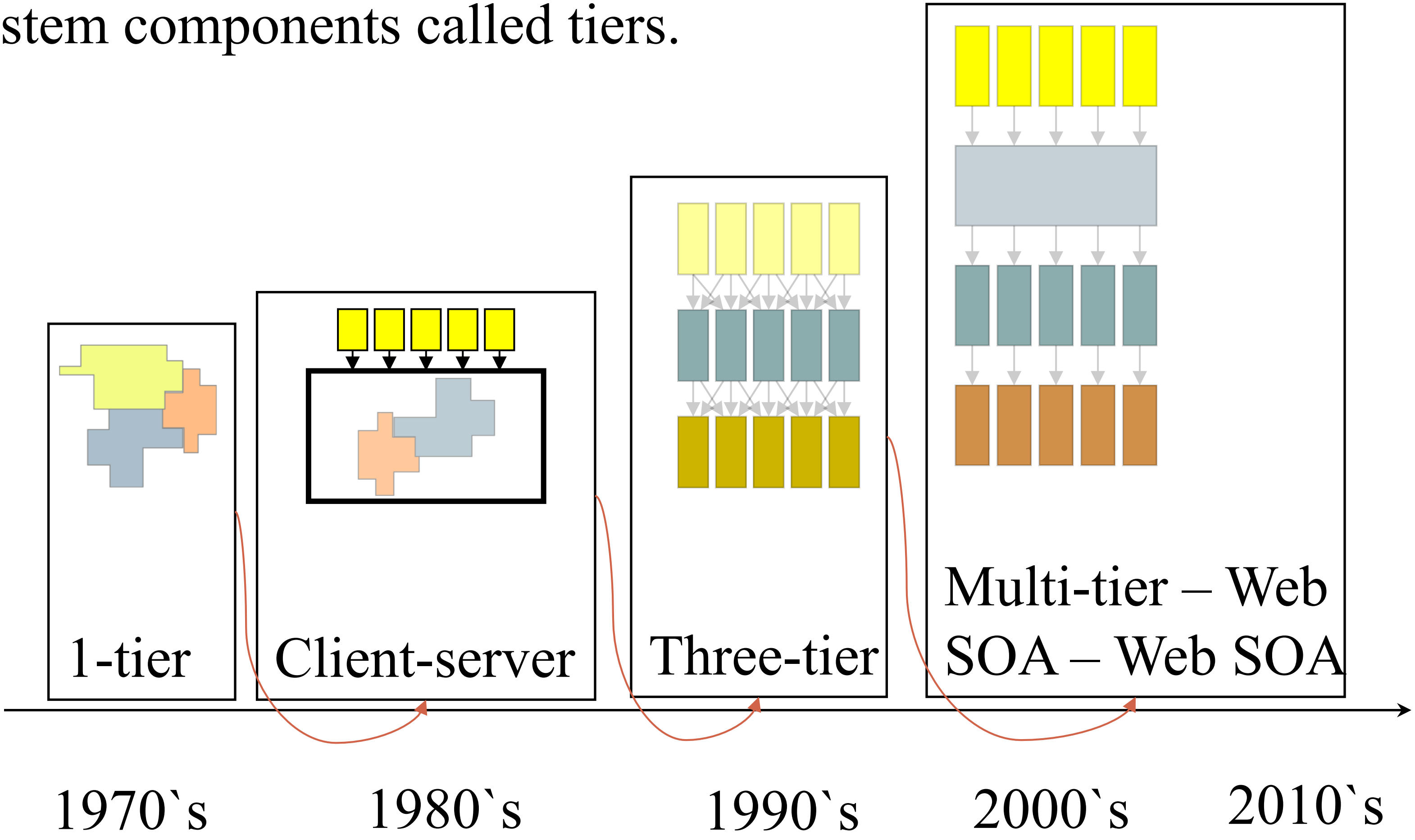
Business processes

Persistent storage

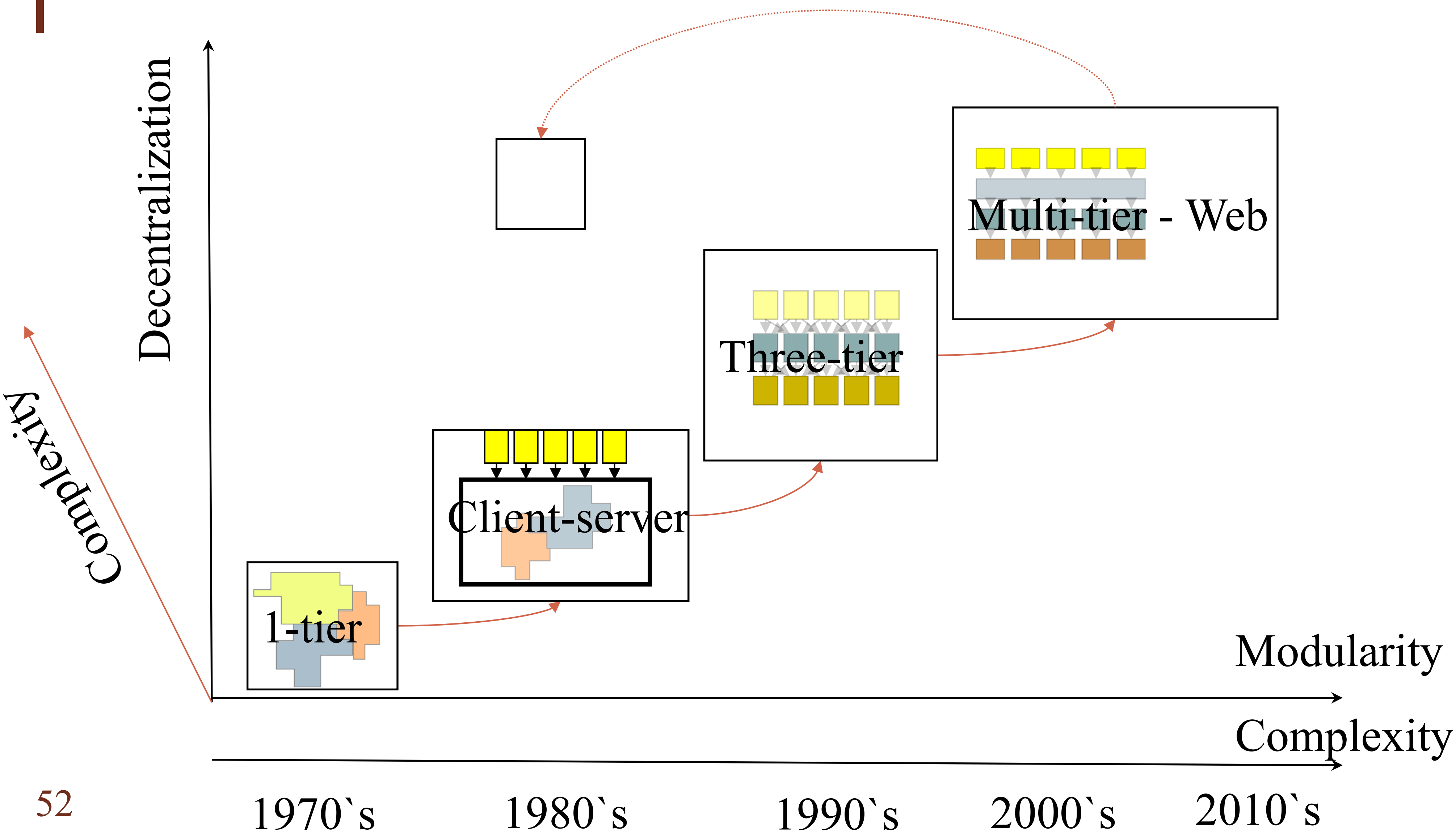
- Presentation:
  - Clients (user or program) wants to perform an operation over the system.
    - Clients interact with the system through a presentation layer.
  - Deals with creating and displaying the (user) interface.
- The application logic:
  - determines what the system actually does.
  - enforcing the business rules and establish the business processes.
  - Can take many forms: programs, constraints, business processes, etc.
- The resource management:
  - deals with the organization (storage, indexing, and retrieval) of the data necessary to support the application logic.
  - Typically a database, a text retrieval system or any other system providing querying capabilities and persistence.

Timeline – Architecture of Information Systems

When implementing real systems we map layers (logical constructs) to system components called tiers.



The Evolution


Decentralization
Complexity
1-tier
Client-server
Three-tier
Multi-tier - Web
Modularity
Complexity
1970`s
1980`s
1990`s
2000`s
2010`s