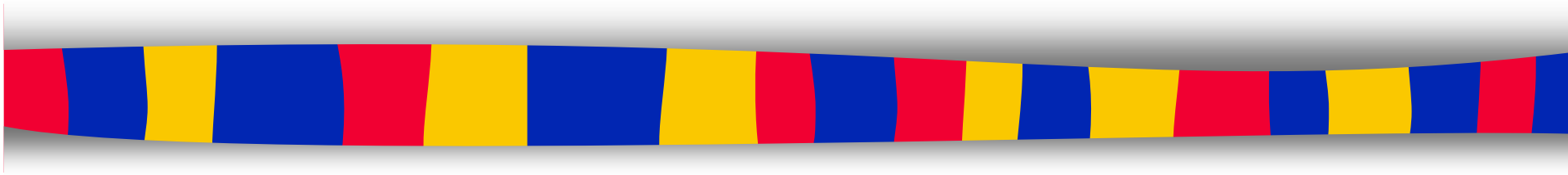
EECS 4314

Advanced Software Engineering



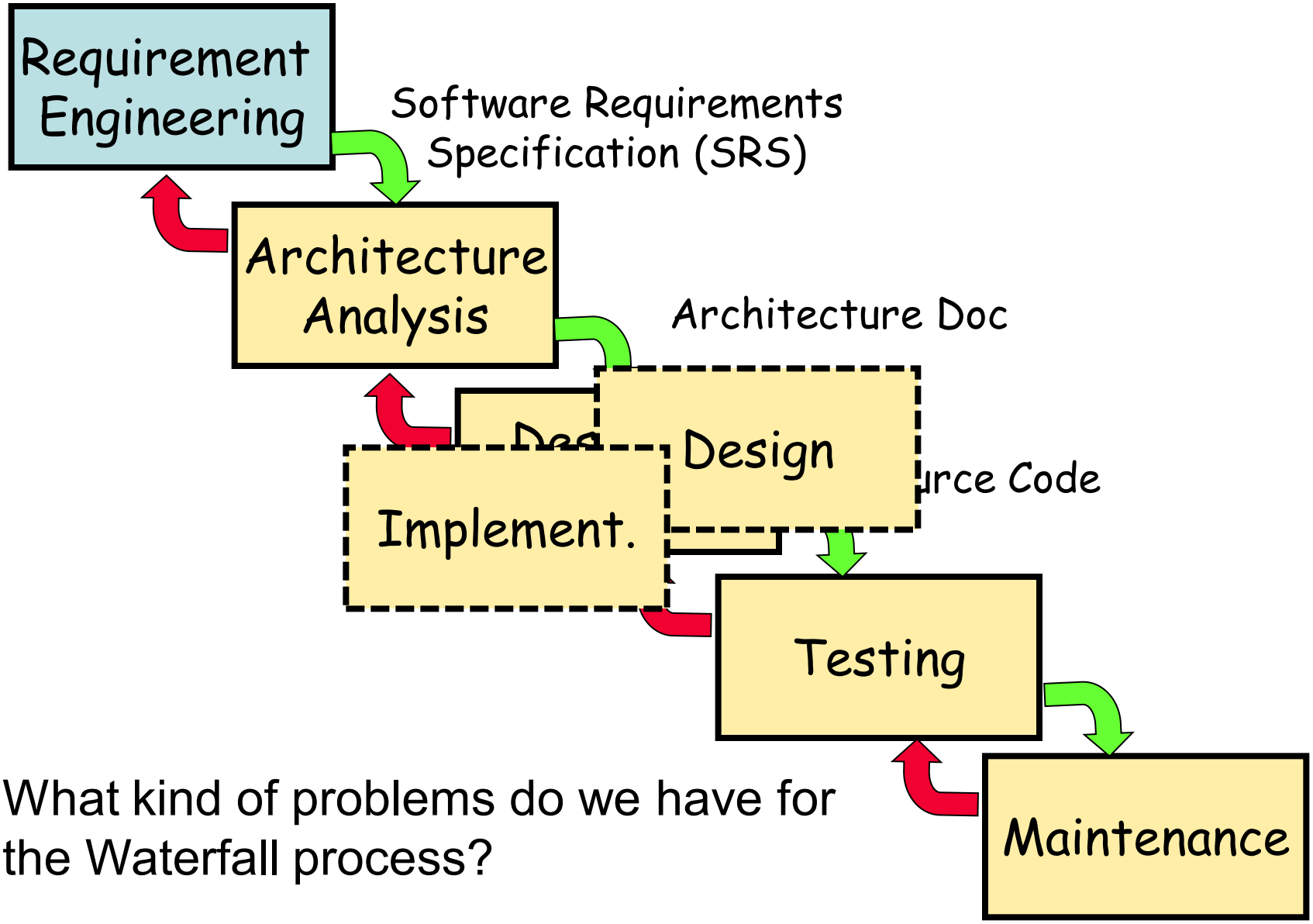
Topic 02:

Course Overview

Zhen Ming (Jack) Jiang

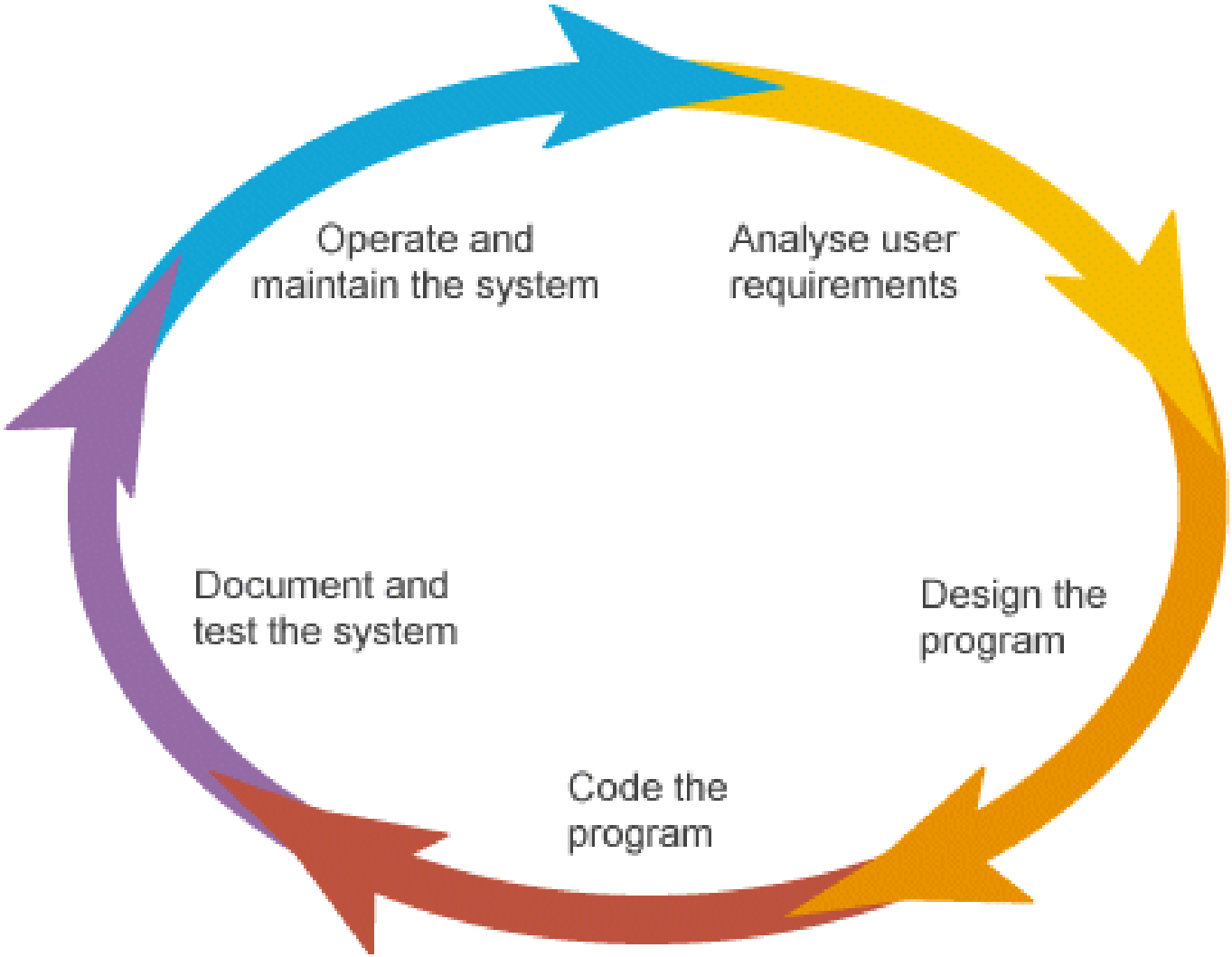
Software Process

Waterfall Development Process

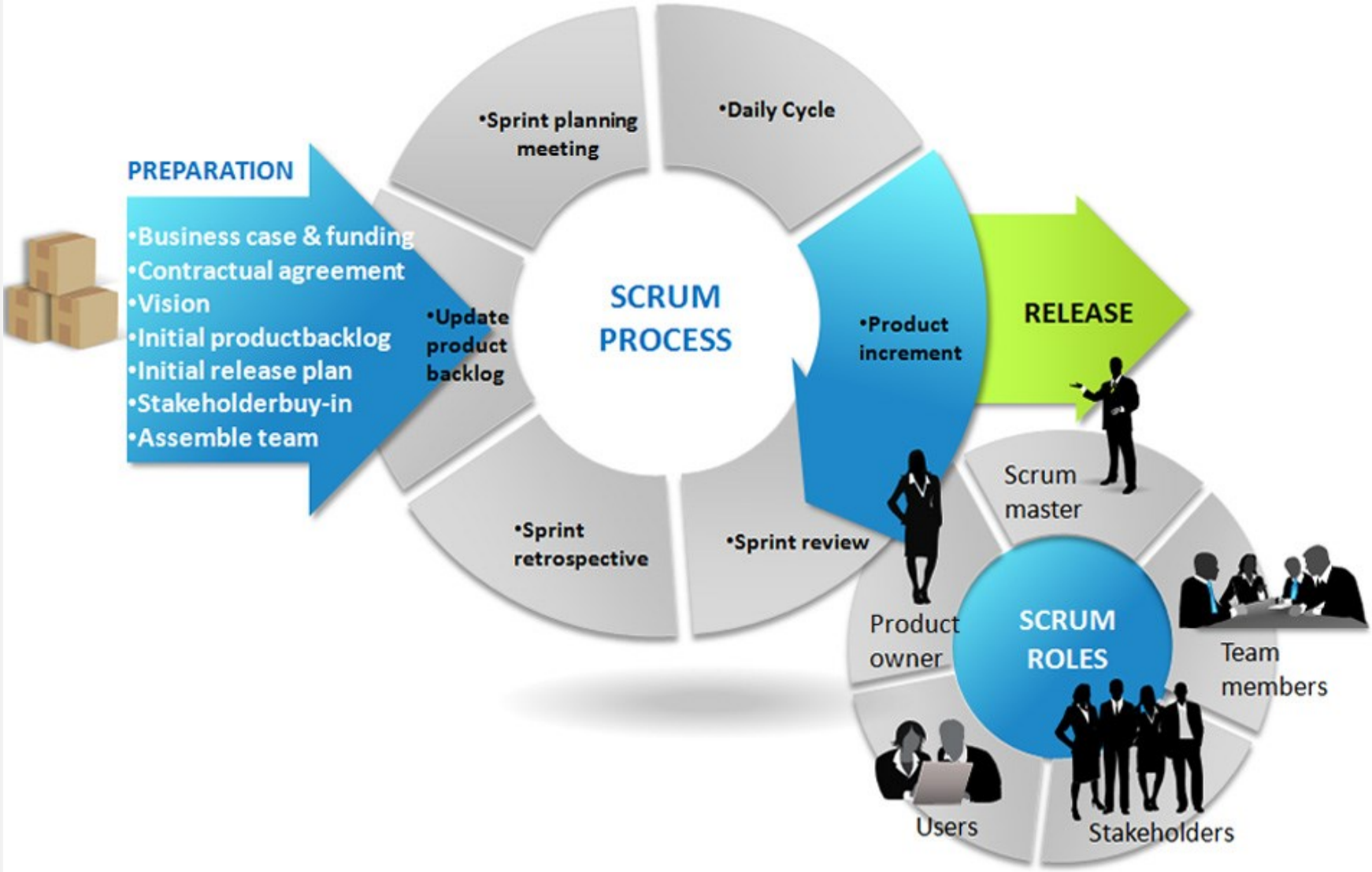


What kind of problems do we have for the Waterfall process?

The Agile Iterative Software Development Process



SCRUM PROCESS

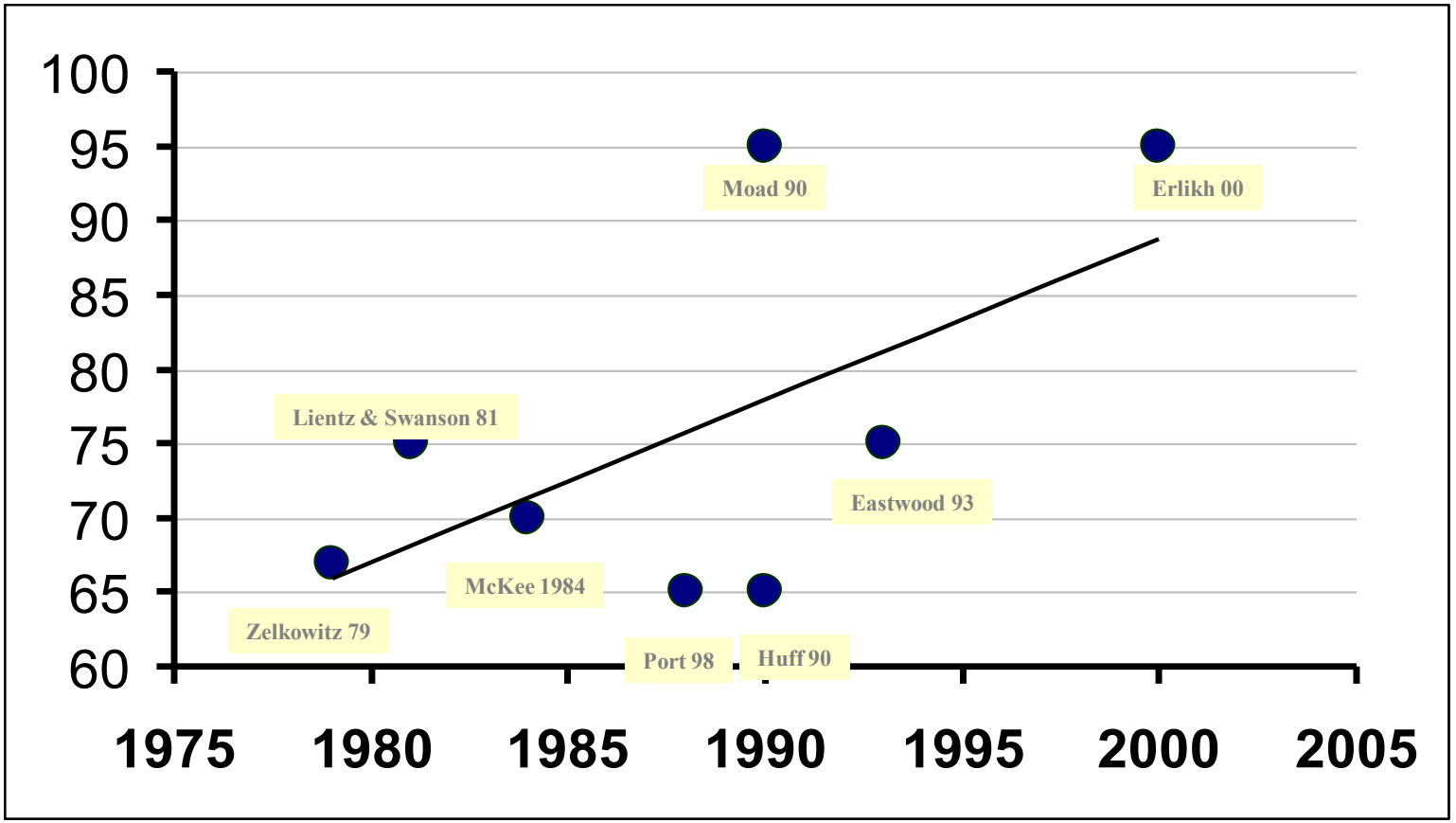


Different Phases in Software Development Cycle

1. Requirements
2. Architecture
3. Design
4. Implementation
5. Testing
6. Maintenance

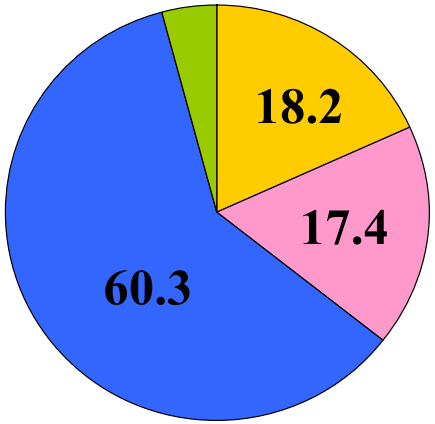
- Most emphasis during undergrad is on phases 3-5
- Which phase lasts the longest and costs the most?
- The problems in which phase is the mostly costly and time consuming to fix?

Percentage of Project Costs Devoted to Maintenance

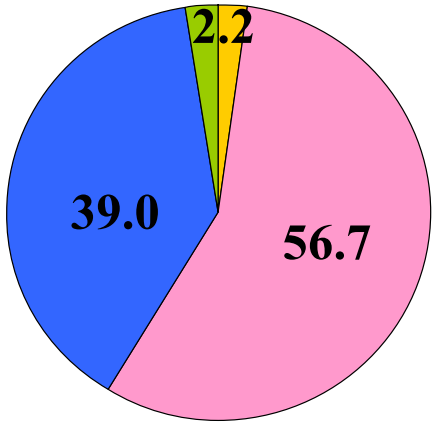


Survey of Software Maintenance Activities

- **Perfective:** add new functionality
- **Corrective:** fix faults
- **Adaptive:** new file formats, refactoring



Lientz, Swanson, Tompkins [1978]
Nosek, Palvia [1990]
MIS Survey



Schach, Jin, Yu, Heller, Offutt [2003]
Mining ChangeLogs
(Linux, GCC, RTP)

Cost of Incorrect or Incomplete Requirements

- [1981] ~ 75 – 85% of all errors found in SW can be traced back to the requirements and design phases
- [2000] Based on a survey of the cost of maintaining 500 major projects, 70 – 85% of total project costs are due to requirement errors and new requirements

The High Cost of Requirements Errors

- The errors discovered during the design of a development project could fall into one of two categories:
  1. Errors that occurred when the development staff created a technical design from a correct set of requirements, or
  2. Errors that should have been detected as requirements errors somewhat earlier in the process but that somehow "leaked" into the design phase of the project.
- It's the second category of errors that turn out to be particularly expensive .. Why?

Because ...

1. The errors are misleading. Everyone is looking for design errors during the testing or inspection activities while in fact they are in the requirements.
2. By the time the requirements error is discovered, time and effort have been lost in faulty design. So, the design have to be thrown away or reworked.

The High Cost of Requirements Errors

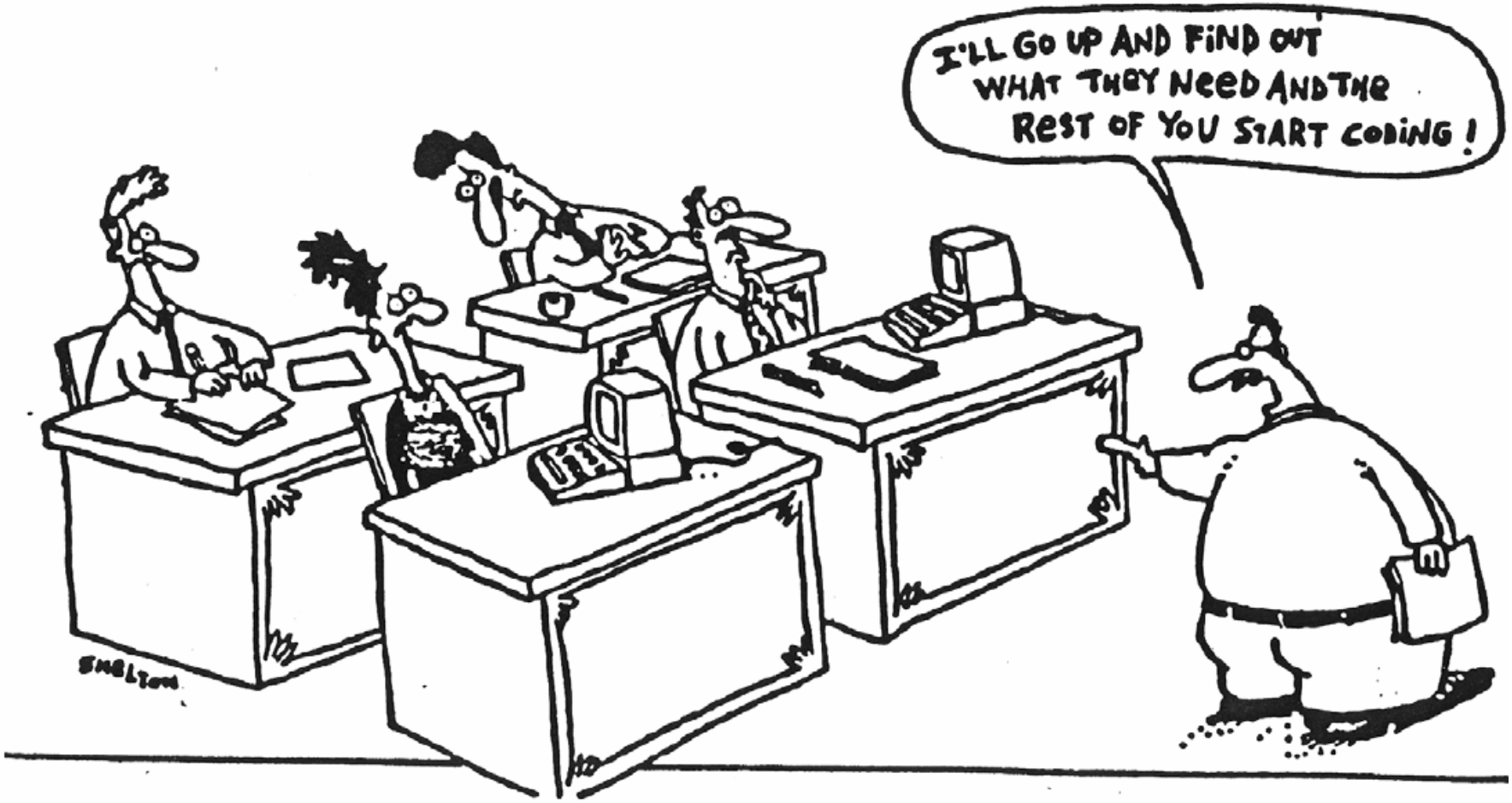
- In order to repair a defect, we are likely to experience costs in some or all of the following areas:
  - Respecification, Redesign, Recoding, Retesting,
  - Change orders: replacing defected systems by corrected one,
  - Corrective action: undoing whatever damage may have been done and refund.
  - Scrap: useless code, design and test cases.
  - Recall of defective software (could be embedded)
  - Warranty costs.
  - Product liability: customer can sue for damages
  - Service costs for reinstallation.
  - Documentation

Requirements

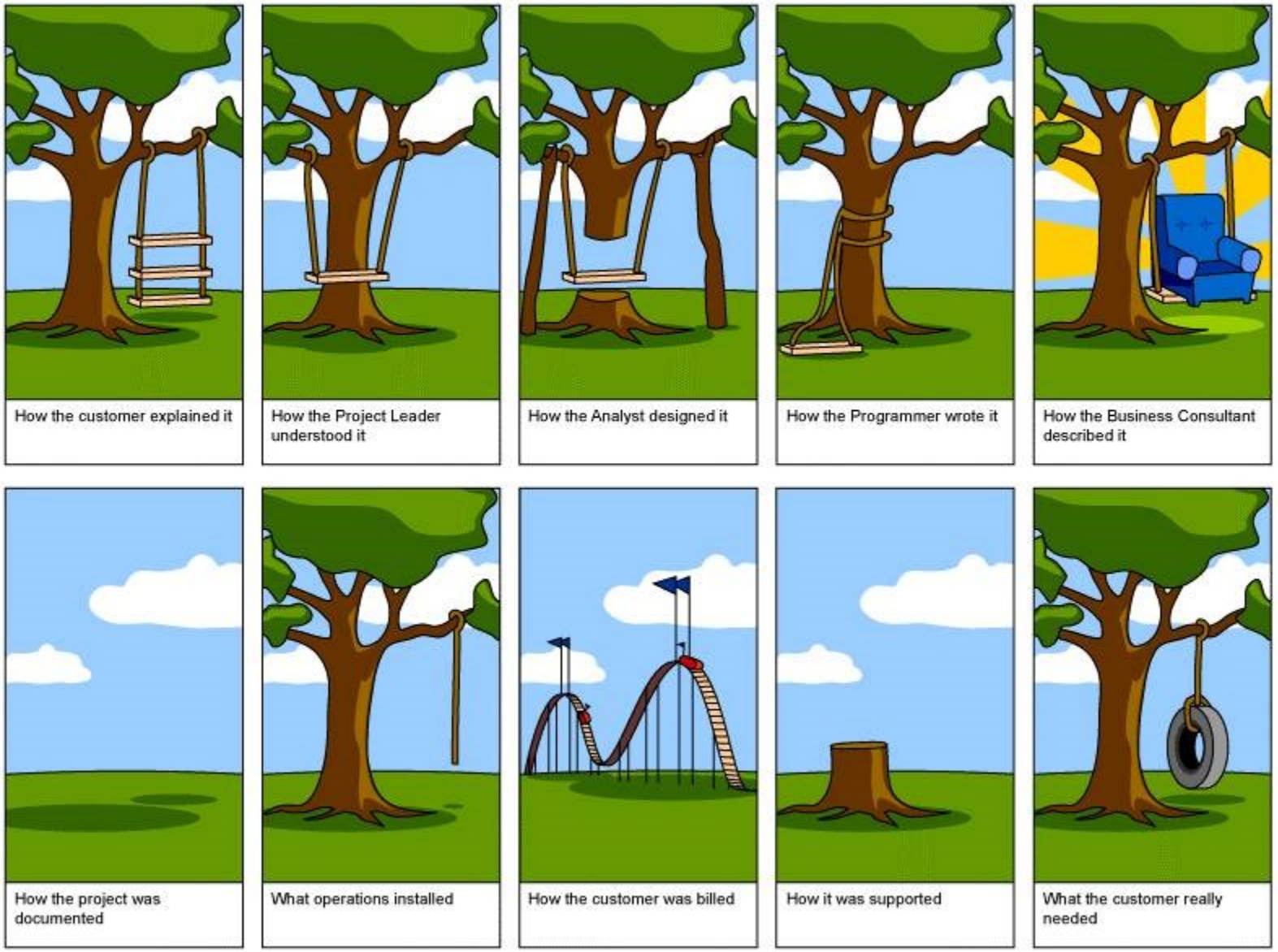
Where Do Requirements Come From?

- Requirements come from users and stakeholders who have demands/needs
- An analyst/requirement engineer:
  - Elicits these demands/needs (raw requirements)
  - Analyzes them for consistency, feasibility, and completeness
  - Formulates them as requirements and write down a specification
  - Validates that the gathered requirements reflect the needs/demands of stakeholders:
    - *Yes, this is what I am looking for.*
    - *This system will solve my problems.*

And Maybe This One



More Stakeholders



Developers' View of Users	Users' View of Developers
Users don't know what they want. Users can't articulate what they want. Users have too many needs that are politically motivated. Users want everything right now. Users can't prioritize needs. Users refuse to take responsibility for the system. Users are unable to provide a usable statement of needs. Users are not committed to system development projects. Users are unwilling to compromise. Users can't remain on schedule.	Developers don't understand operational needs. Developers place too much emphasis on technicalities. Developers try to tell us how to do our jobs. Developers can't translate clearly stated needs into a successful system. Developers say no all the time. Developers are always over budget. Developers are always late. Developers ask users for time and effort, even to the detriment of the users' important primary duties. Developers set unrealistic standards for requirements definition. Developers are unable to respond quickly to legitimately changing needs.

Types of Requirements

- **Functional Requirements**
  - Specify the function of the system
  - F(input, system state) → (output, new state)
- **Non-Functional Requirements (Constraints)**
  - **Quality Requirements**
    - Specify how well the system performs its intended functions
    - Performance, Usability, Maintenance, Reliability, Portability
  - **Managerial Requirements**
    - When will it be delivered
    - Verification (how to check if everything is there)
    - What happens if things go wrong (legal responsibilities)
  - **Context / Environment Requirements**
    - Range of conditions in which the system should operate

Design

Design v.s. Architecture

- **Design**
  - Inner structure of the components
  - Low level (information hidding and interfaces help it change)
  - Mostly technical concerns
  - Makes sense for systems with KLOCs
  - Late in life cycle
- **Architecture**
  - Structure of system (components and connectors)
  - High level and hard to change (better get it right!)
  - Concerned with technical and non technical requirements (e.g., Security, Legal, Outsourcing)
  - Makes sense for systems with MLOCs
  - Very early in life cycle

Design Patterns

- Good designers know not to solve every problem from first principles. They reuse solutions.
- Practitioners do not do a good job of recording experience in software design for others to use.
- A **Design Pattern** systematically names, explains, and evaluates an important and recurring design.
- We describe a set of well-engineered design patterns that practitioners can apply when crafting their applications.

Software Architecture

Software Architecture (IEEE Definition)

- *Architecture is the fundamental **organization** of a **system** embodied in its **components**, their **relationships** to each other, and to the **environment**, and the principles guiding its design and evolution. [IEEE 1471]*

What is a system? (*[IEEE 1471]*)

- ***System:*** *a collection of components organized to accomplish a specific function or set of functions.*
- *A System can mean*
  - *individual applications*
  - *systems in the traditional sense*
  - *subsystems, systems of systems, etc…*
- *A system exists to fulfill one or more* ***missions*** *in its environment.*

Environments, Missions and Stakeholders (*[IEEE 1471]*)

- ***Environment****: determines the setting and circumstances of developmental, operational, political, and other influences upon that system.*

- ***Mission:*** *a use or operation for which a system is intended by one or more* ***stakeholders*** *to meet some set of objectives.*

- ***Stakeholder:*** *an individual, team, or organization (or classes thereof) with interests in, or concerns relative to, a system.*

Architectural Styles

Understanding Software Architecture

- **Live Architecture**
  - Is in head(s) of software developer(s), the "software architect"
  - May be abstract or mostly concrete
  - Is a "mental model", "wetware"; may be fuzzy, inaccurate, incomplete, incorrect …
- **Complexity**
  - Architecture simplifies the system, by concentrating on structure, not content or semantics
  - Cognitive complexity: how hard to understand or visualize
- **Reverse Engineering**
  - Extraction of design (or architecture) from implementation and from developers
  - "Design recovery"

Architectural Styles of Software Systems

- **Architectural Style**
  - Form of structure, e.g.,
    - "Pipes" between components, or
    - "Layered" system, or
    - "Bulletin board" system
  - *Analogy: Style of a building*
- It determines:
  - the vocabulary of components and connectors that can be used in instances of that style
  - a set of constraints on how they can be combined. For example, one might constrain:
    - the topology of the descriptions (e.g., no cycles).
    - execution semantics (e.g., processes execute in parallel).

More Terminology

- **Reference Architecture**
  - General architecture for an application domain
  - *Example: Common structure for compilers or for operating systems*
- **Product Line Architecture (PLA)**
  - Architecture for a line of similar software products
  - *Example: Software structure for a family of computer games*

Determining an Architectural Style

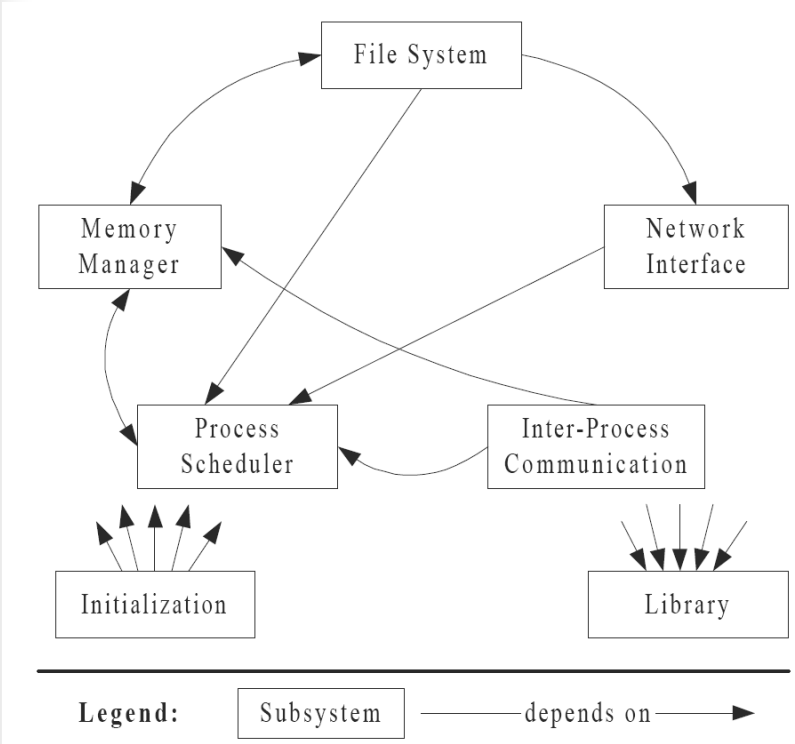
- We can understand what a style is by answering the following questions:
  - What is the **structural pattern**? (i.e., components, connectors, constraints)
  - What are the **essential invariants** of the style?
  - What are some **common examples of its use**?
  - What are the **advantages** and **disadvantages** of using that style?
  - What are some of the **common specializations** of that style?

Architecture Recovery

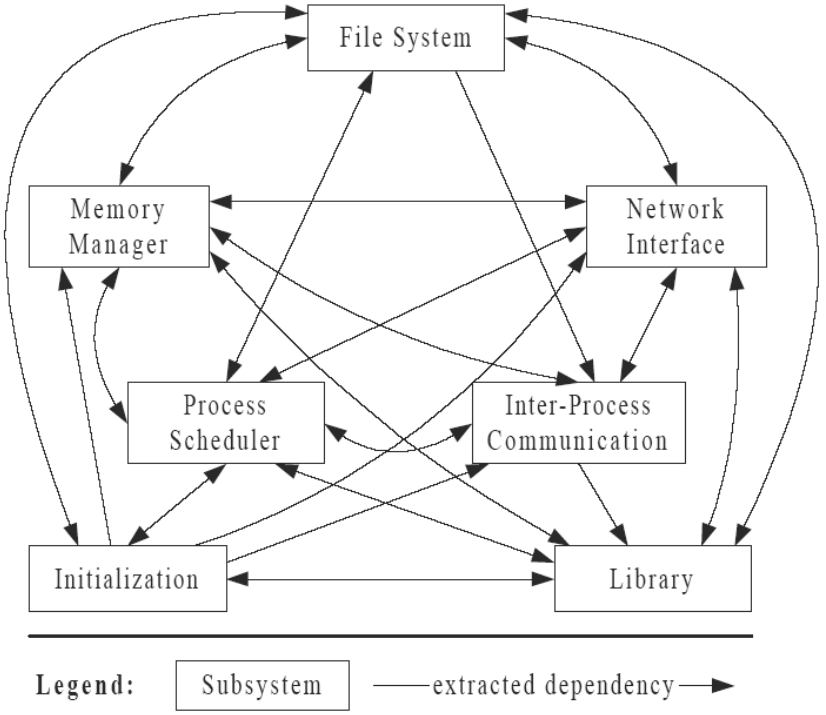
Architecture Terminology

- **Conceptual Software Architecture**
  - Abstract structure: Large piece of software with many parts and interconnections
  - *Analogy: Blueprint of house*
- **Concrete Software Architecture**
  - Actual structure: Large piece of software with many parts and interconnections
  - *Analogy: Actual structure of house*
- **Reference Architecture**
  - General architecture for an application domain
  - Example: Common structure for compilers or for operating systems
  - *Analogy: Typical architecture of a house*

Linux Architecture



Conceptual Architecture



Concrete Architecture

Project Scheduling

Project

- A project is
  - a temporary endeavour undertaken to create a "unique" product or service
- A project is composed of
  - a number of related activities that are directed to the accomplishment of a desired objective
- A project starts when
  - at least one of its activities is ready to start
- A project is completed when
  - all of its activities have been completed

Project plan

- A project plan is a schedule of activities indicating
  - The start and stop for each activity. The start and stop of each activity should be visible and easy to measure
  - When a resource is required
  - Amount of required project resources

Project Planning

- Managers should consider:
  - Resource availability
  - Resource allocation
  - Staff responsibility
  - Cash flow forecasting
- Mangers need to monitor and re-plan as the project progresses towards its pre-defined goal

Cost Estimation

Software cost estimation

- Predicting the resources required for a software development process
  - Productivity
  - Estimation techniques
  - Algorithmic cost modelling
  - Project duration and staffing

Software Metrics
(if time permits)

Example software metrics

- LOC: number of lines of code
- NOM: number of methods
- FanIn: number of other classes that reference the class
- FanOut: number of other classes referenced by the class
- CBO: coupling between objects

Design Principles:

"High cohesion and low coupling"

Software Performance
(if time permits)

Common Goals of Performance Evaluation (1)

Evaluating Design Alternatives

- Should my application service implement the push or pull mechanism to communicate with my clients?

Comparing System Implementations

- Does my application service yield better performance than my competitors?

Benchmarking

Common Goals of Performance Evaluation (2)

Performance Debugging

- Which part of the system slows down the overall executions?

Performance Tuning

- What are the configuration values that I should set to yield optimal performance?

Common Goals of Performance Evaluation (3)

Performance Prediction

- How would the system look like if the number of users increase by 20%?

what-if analysis

Capacity Planning

- What kind of hardware (types and number of machines) or component setup would give me the best bang for my buck?

Common Goals of Performance Evaluation (4)

Performance Requirements

- How can I determine the appropriate Service Level Agreement (SLA) policies for my service?

Operational Profiling

- What is the expected usage once the system is deployed in the field?

workload characterization