# Verification as Part of the Development Process

CSCE 747 - Lecture 25 - 04/12/2016

# Life Cycle of Software

Any product - software included - has a life cycle: a timeline that can be split into the required phases of existence.

**What are the phases of a software lifecycle?**

**Project Planning**

Requirements Definition

Software Design

Implementation

Testing

Release

Operation/ Maintenance

# The Need for Planning

**Why do we get stuck in the code & fix loop?**

We know the phases of the lifecycle. We know there are activities that must be performed:

- Specification, Design, Coding, Testing, Evolution

Lack structure and guidance:

- When are we done? When do we move on?
- Activities must be planned and modeled if they are to be managed.

# Risk Management

The principle task of a manager is to minimize (avoid or mitigate) risk.

- The “risk” in an activity is a measure of the uncertainty of the outcome of that activity.
  - Risk is related to the amount and quality of available information.
  - What are the risk factors? What will be their impact? How likely are they to arise?

# Defining a Process

**Process:** a flow of events that describes how something works.

- In our case - defines a timeline of human activities required to build software.
- Structures who is doing what, when, and how.
- Many different software processes:
  - Traditional: Strict, regimented phases. We move to a new phase only once one is done.
  - Agile: Short bursts of development where specification, design, testing, and coding are mixed.

# Risk Management

High-risk activities cause schedule and cost overruns.

**A visible process provides the means to track, assess, and mitigate risk.**

Processes provide quality and predictability by removing risk.

# The Quality Process

# Quality Process

- “Quality” is not something that can be added in a final step before delivery.
- A testing phase is part of development...
  - … but quality-assuring activities (testing, verification) should be part of all stages of the life cycle.
  - Quality must be a part of all development phases, not just during analysis and testing.
- Quality process is our plan for ensuring quality in the final release.
  - Intertwined with the overall process.

# Quality Process

- A framework for selecting and scheduling activities towards a particular goal.
  - Considers trade-offs and interactions with other important goals.
- All activities involve trade-offs and impose constraints on development.
  - Dependability vs time-to-market.
  - Better, faster, cheaper - pick two.
  - A good process allows planners to choose optimal trade-offs.

# Quality Process Structure

- Should be structured for:
  - **Completeness**
    - Appropriate activities are planned to detect each important class of faults.
  - **Timeliness**
    - Faults are detected as early as possible.
  - **Cost-effectiveness**
    - Choose activities based on their balance of cost versus effectiveness.

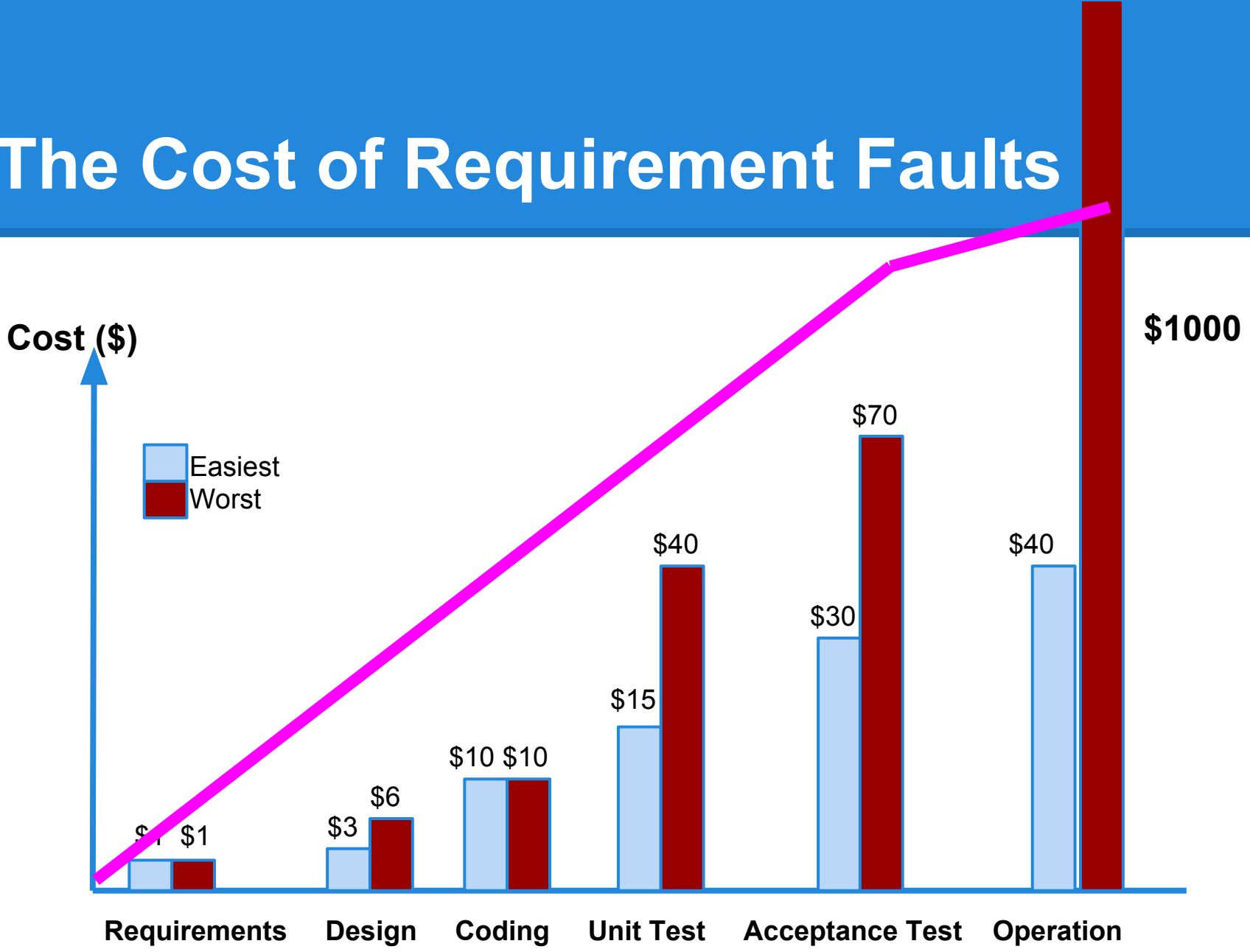
# Quality as Part of Overall Process

- Quality activities intertwine with other development activities.
  - Architectural design has an impact on the cost and types of testing possible.
    - … and integration tests can be planned once the architectural design is available.
  - An architectural model can be analyzed before code is written, used to perform verification.
- Quality activities should not be reserved for later in the development process.

# Quality as Part of Overall Process

- Mutual benefits between quality and other activities.
  - Planning tests while specifying requirements identifies faults in requirements, allows refinement of vague or contradictory requirements.
  - Planning tests during design suggests interfaces and structures, identifies optimizations to structural dependencies.
- Best predictor of cost to repair a fault is time between introduction and detection.

# The Cost of Requirement Faults




(From “Extra Time Saves Money”, Warren Kuffel)

# Quality Goals

- Properties that the software must exhibit to be “high quality.”
- Must be measurable.
- Must also be broken down into a set of reasonable tasks that can be completed.
  - Balancing cost against attainment.
- Can be divided into **external** and **internal** qualities.

# Internal Quality Goals

- Primarily affect the development organization.
  - Maintainability - the software can be updated over time without degradation.
  - Reusability - parts of the software can be reused in future projects with minimal changes.
  - Traceability - developers can trace code to related requirements.
- Can impact external customers as well.

# External Quality Goals

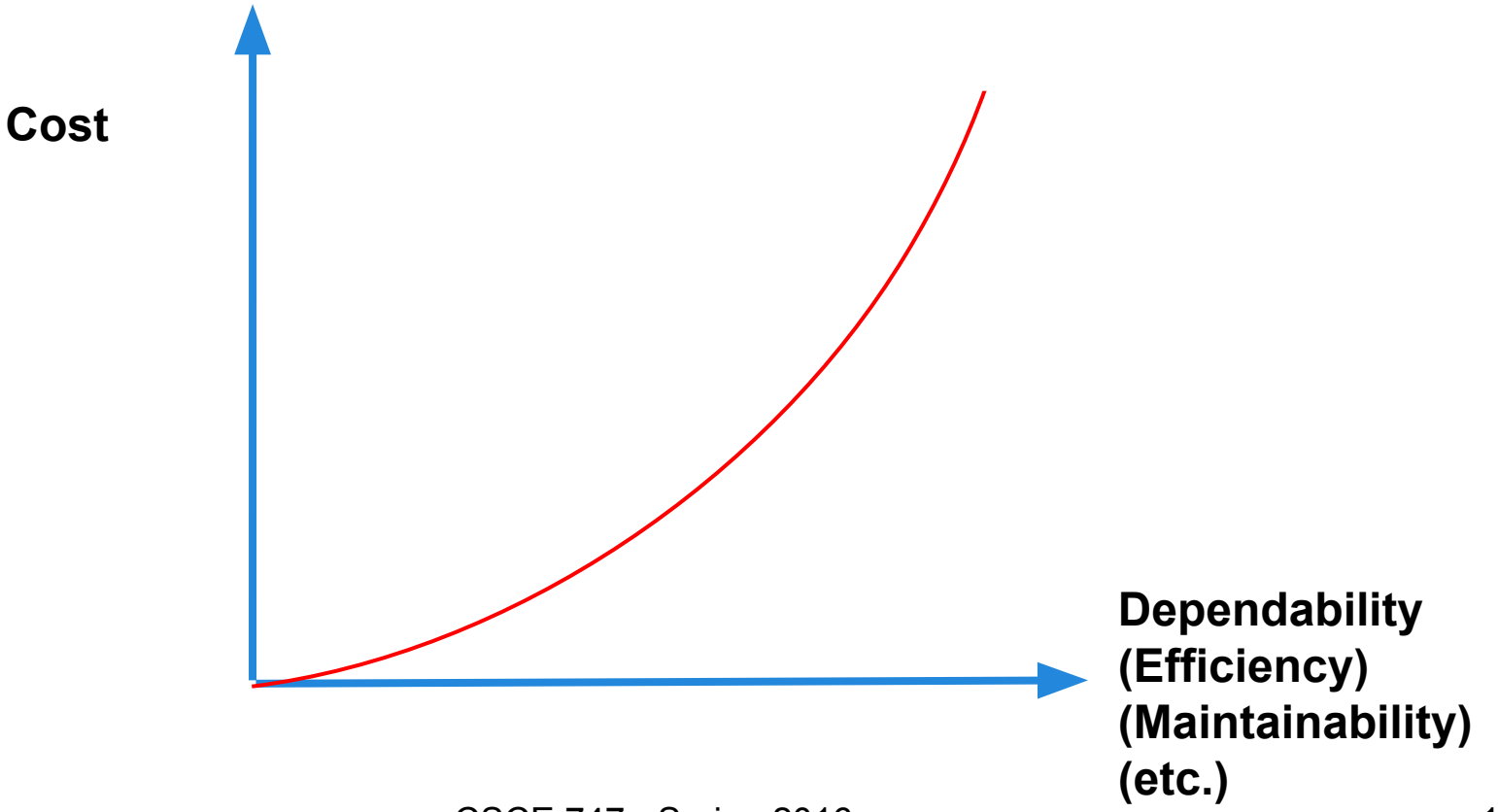
- Visible to the customer.
  - Dependability - how regularly does the system function without crashing?
  - Latency - how long does it take to get output?
  - Usability - how easy is the software to use?
  - Safety - is the software able to avoid situations where critical losses could occur?
- Can be divided into **dependability** and **usefulness** properties.

# External Quality Goals

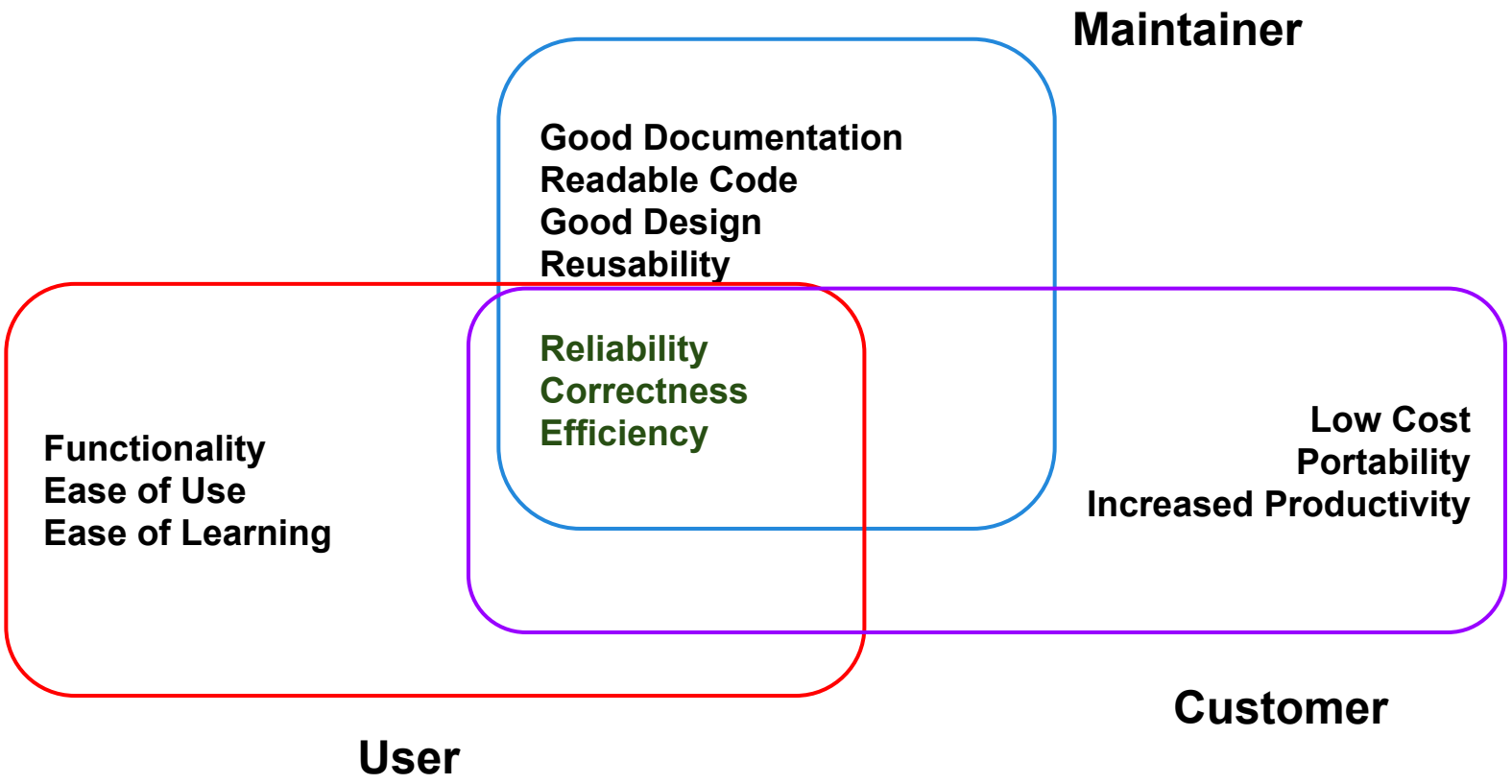
- **Dependability** - does the software do what it was intended to do?
  - If it is not dependable, it has a fault.
- **Usefulness** - can the software be used for its intended job?
  - The software can be reliable and useless.
  - May be slow, have a bad interface, be missing documentation or features.

# Expensive to Maximize Goals

Costs rise exponentially if very high levels of an goal are required.



# Quality is in the Eyes of Beholders

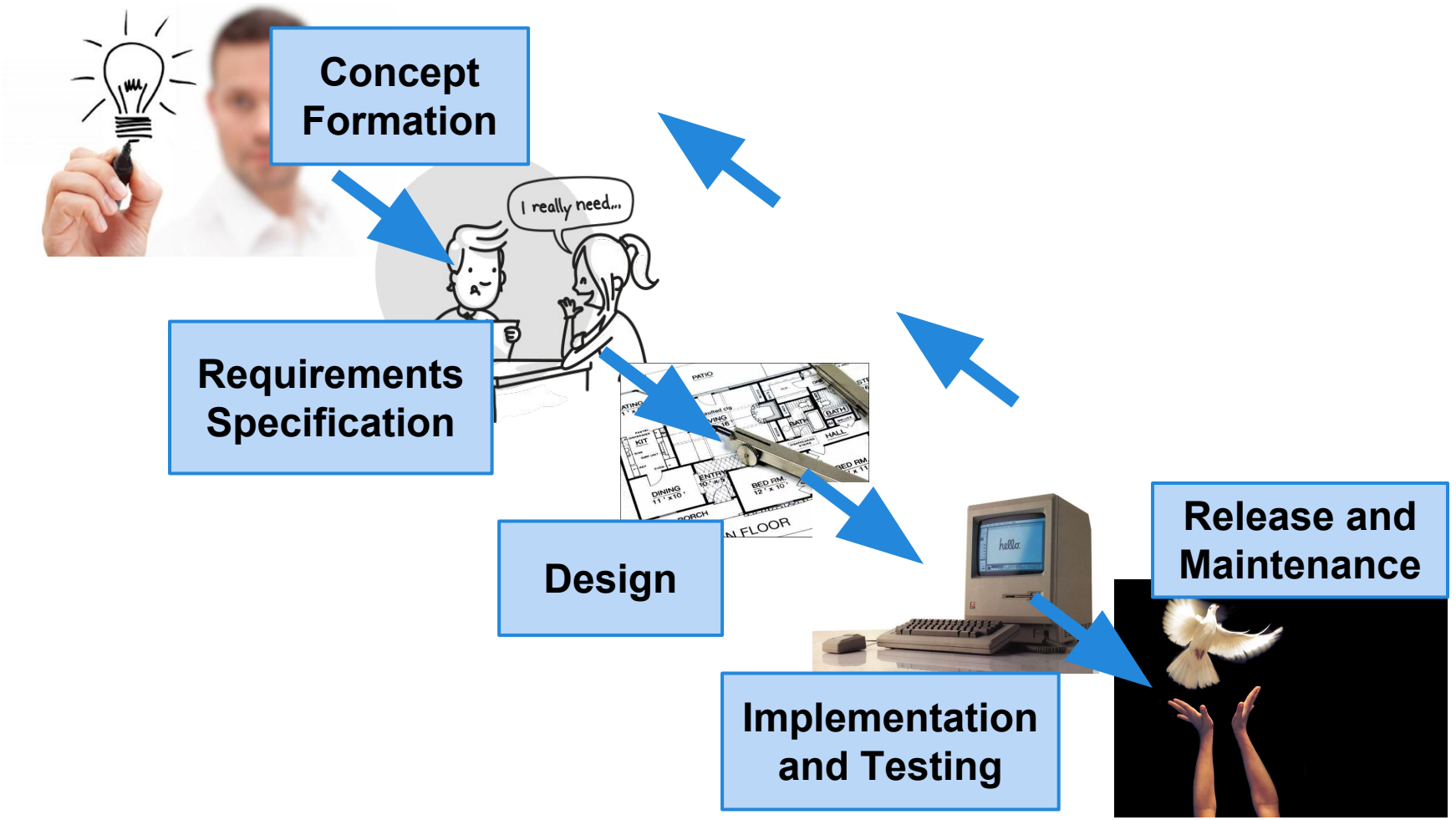


# Planning and Monitoring

- Quality process requires coordination of many different activities.
- Planning is needed to order, provision, and coordinate all activities supporting a goal.
- Monitoring of a process is needed to measure completion of a plan and to steer and adjust the process.

# Planning the Process

# The Software Lifecycle



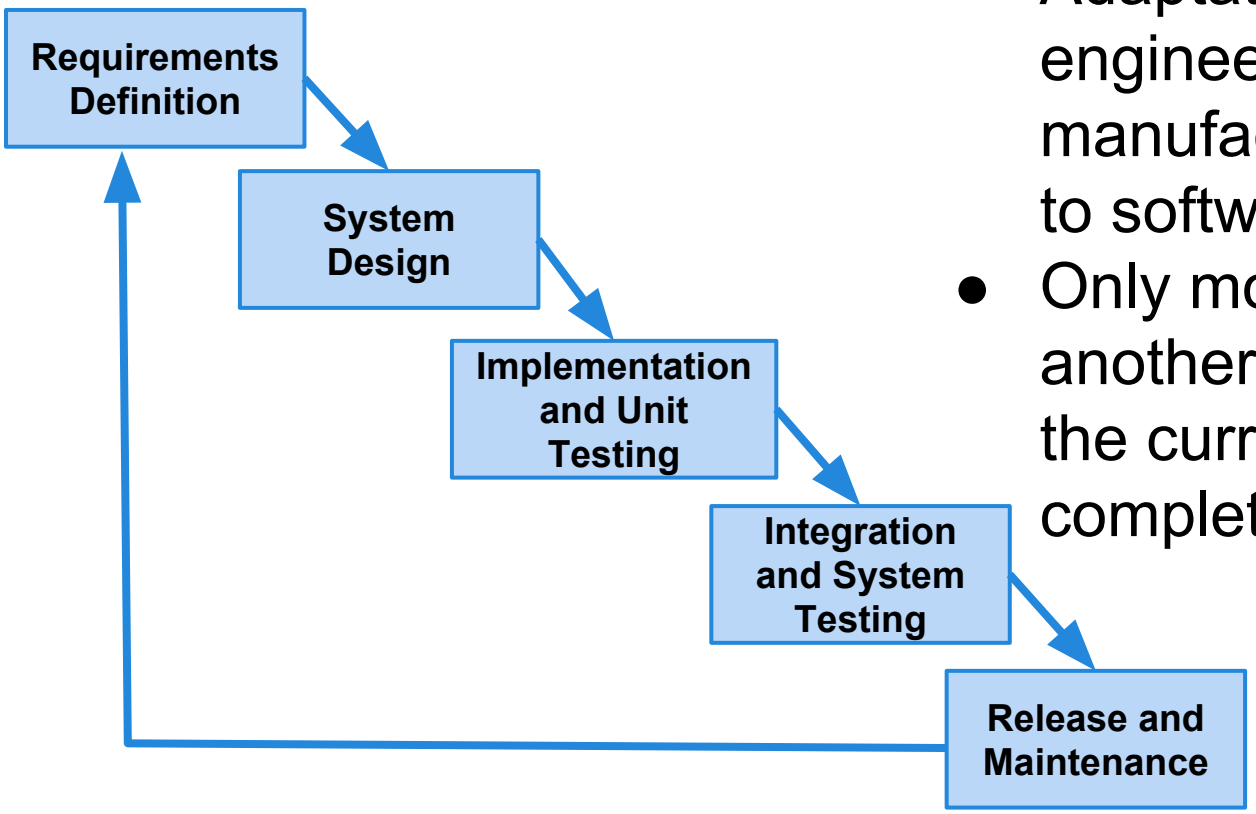
# Planning the Process

- Planning involves scheduling activities, allocating resources, and devising milestones.
- Quality activities must be coordinated with other development processes.
  - May constrain order that activities are completed.
  - May shape tasks to facilitate coordination.
- Quality planning begins at project inception and follows cycles of formulation and execution.

# Planning the Process

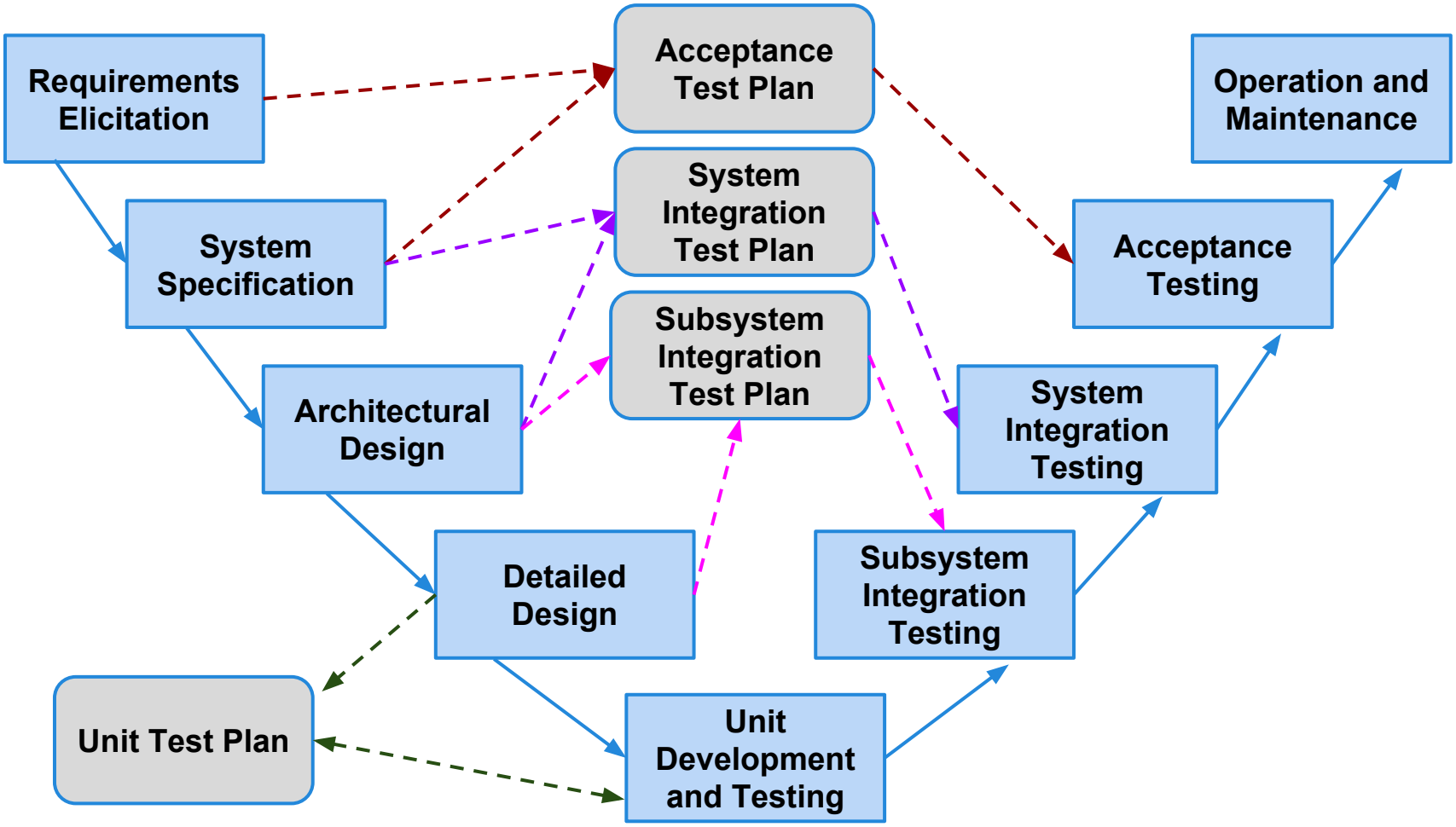
- The quality process should follow a form similar to the overall process:
  - Traditional - strictly separated phases, where an activity only begins once the previous one is “done”.
  - Agile - development proceeds incrementally, and activities are mixed.
    - (but focused on the current increment)
  - Mixed - increments or spirals with distinct phases.
  - As faults are cheaper to fix when detected earlier, each development step in the process should be paired with a verification step.

# The Waterfall Model

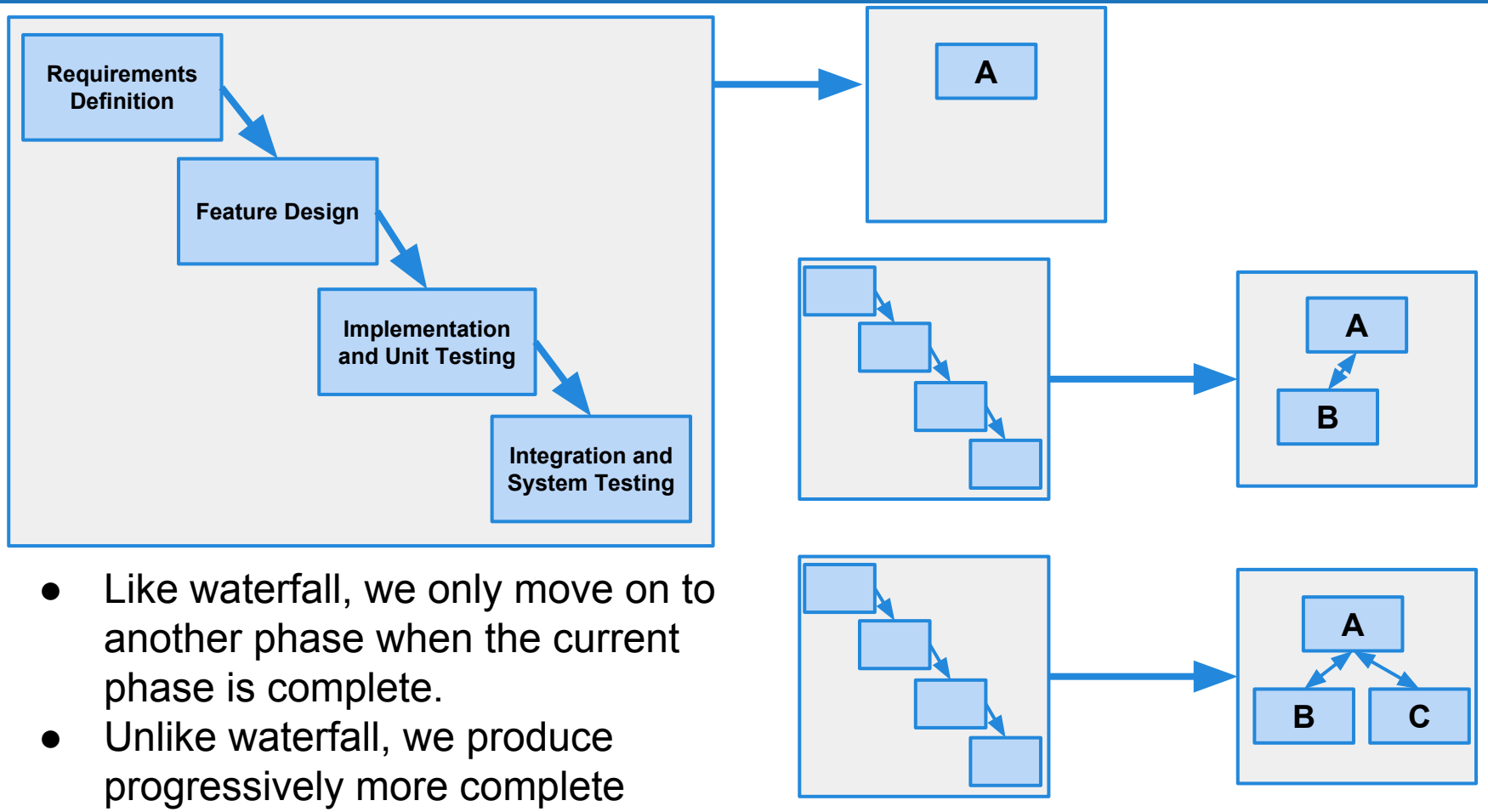



- Adaptation of engineering manufacturing process to software.
- Only move on to another phase when the current phase is complete.

# The V-Model of Development



# The Incremental Model

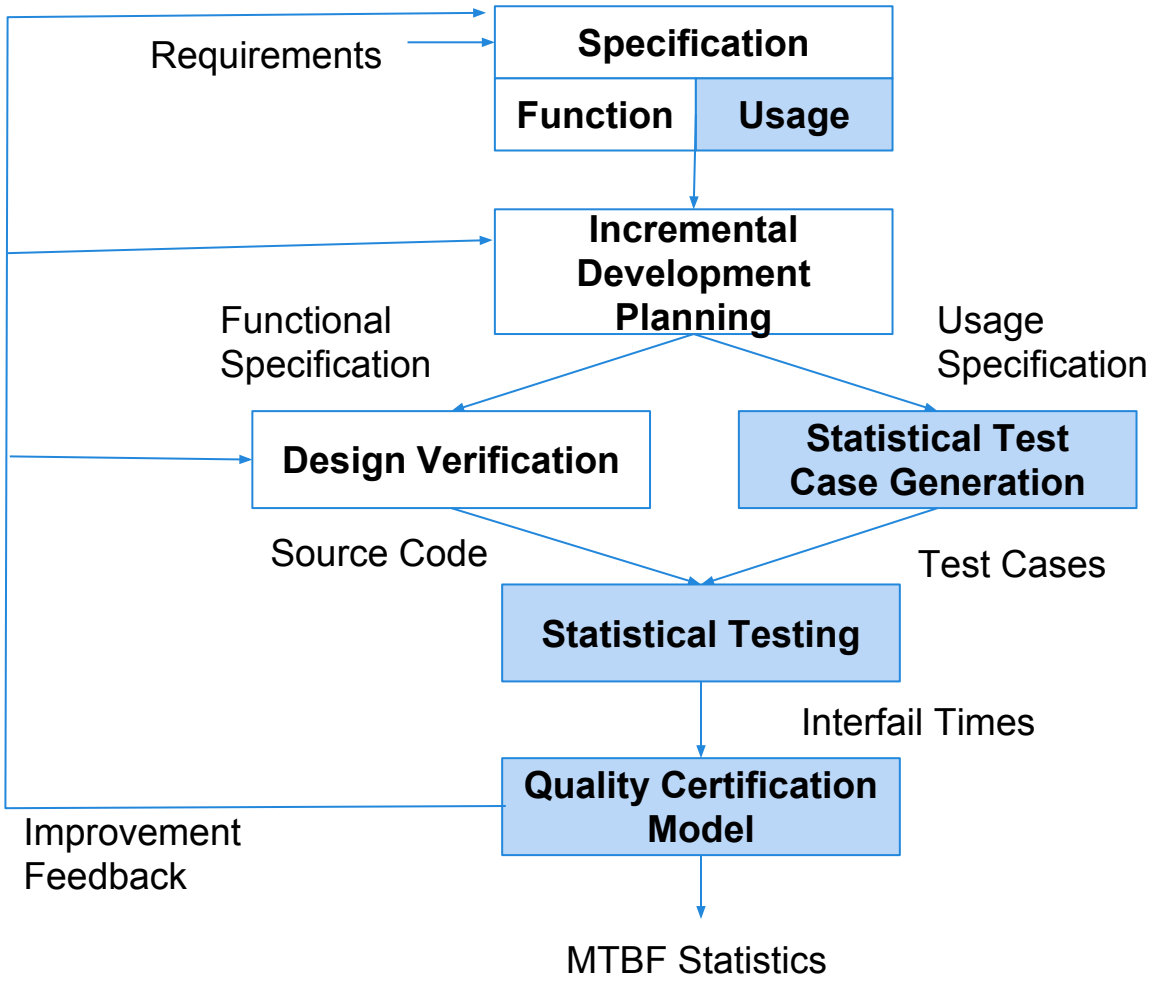



- Like waterfall, we only move on to another phase when the current phase is complete.
- Unlike waterfall, we produce progressively more complete builds of a system.

# Cleanroom Process

- Incremental process that pairs development and verification activities.
  - Stresses analysis over testing in earlier phases.
  - Testing is left for a near-release “certification” stage.
- Two cooperative teams - development and quality assurance.
- Five major activities: specification, planning, design and verification, quality certification, and feedback.

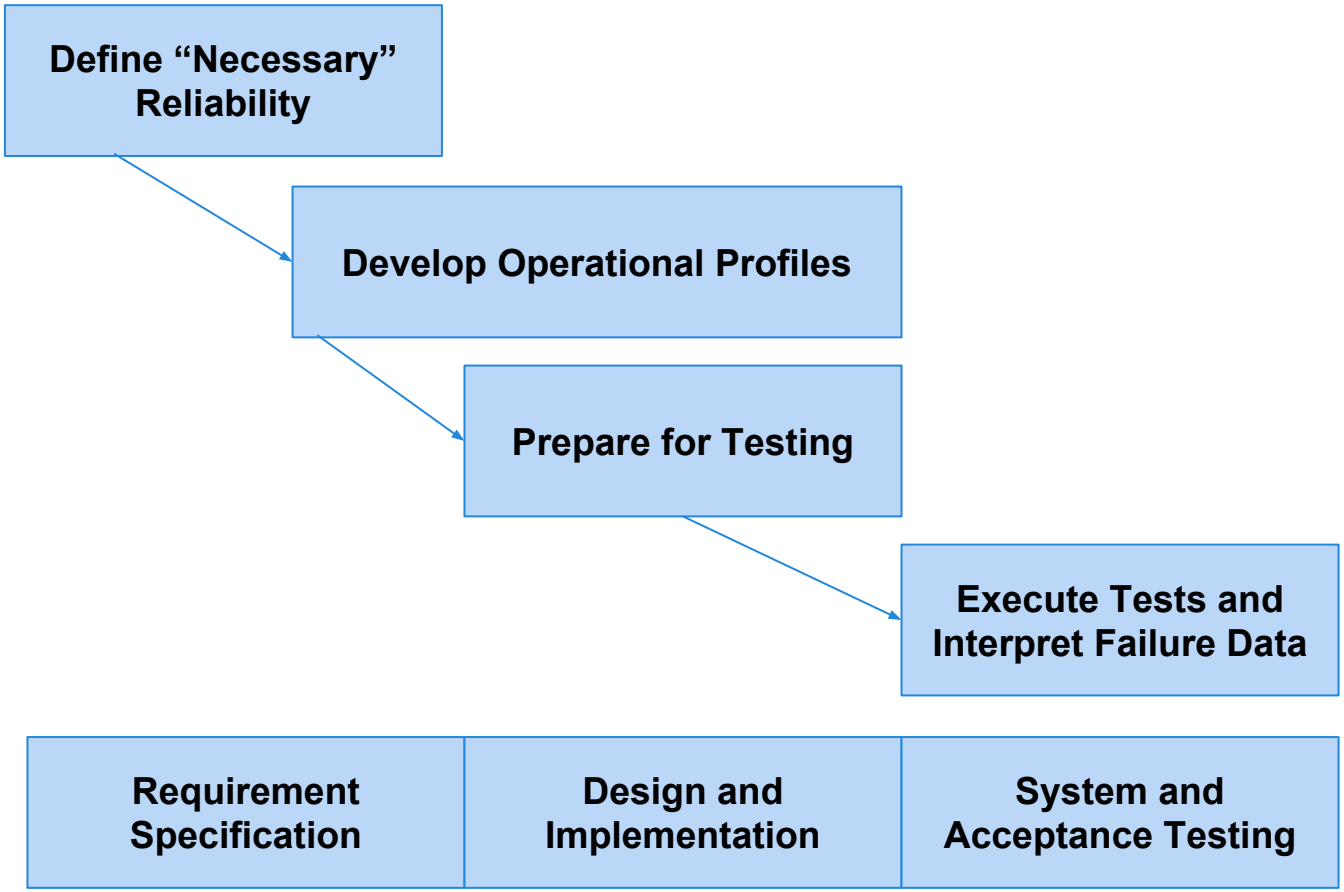
# Cleanroom Process



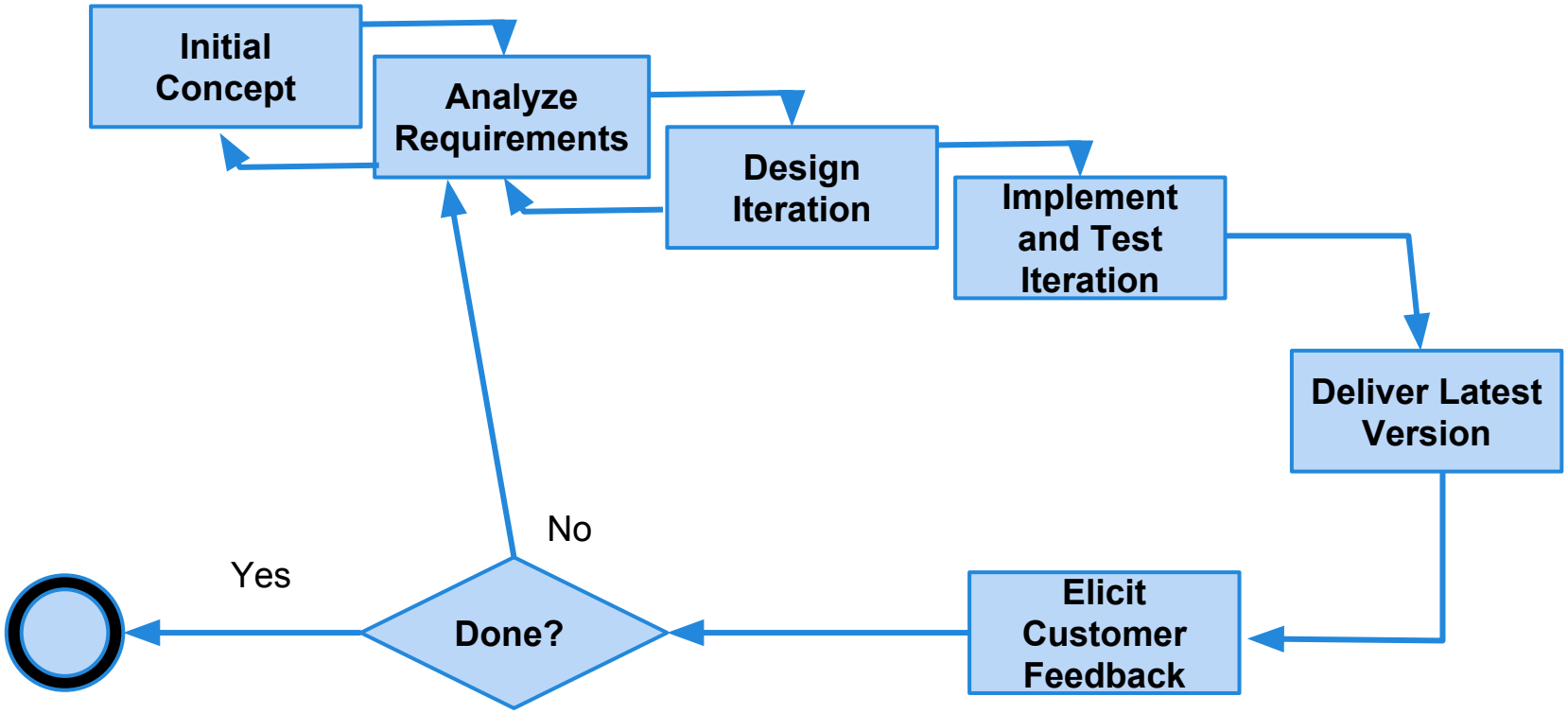
# SRET Process

- Software Reliability Engineered Testing
- Incremental process. Augments each increment with testing activities.
- Defines two types of testing:
  - Development testing - used to find and remove faults in software built on-site.
  - Certification testing - used to either accept or reject outsourced software.
- Two planning, five core steps.
  - Executed in parallel with each increment.

# SRET Process



# The Iterative/Evolutionary Model

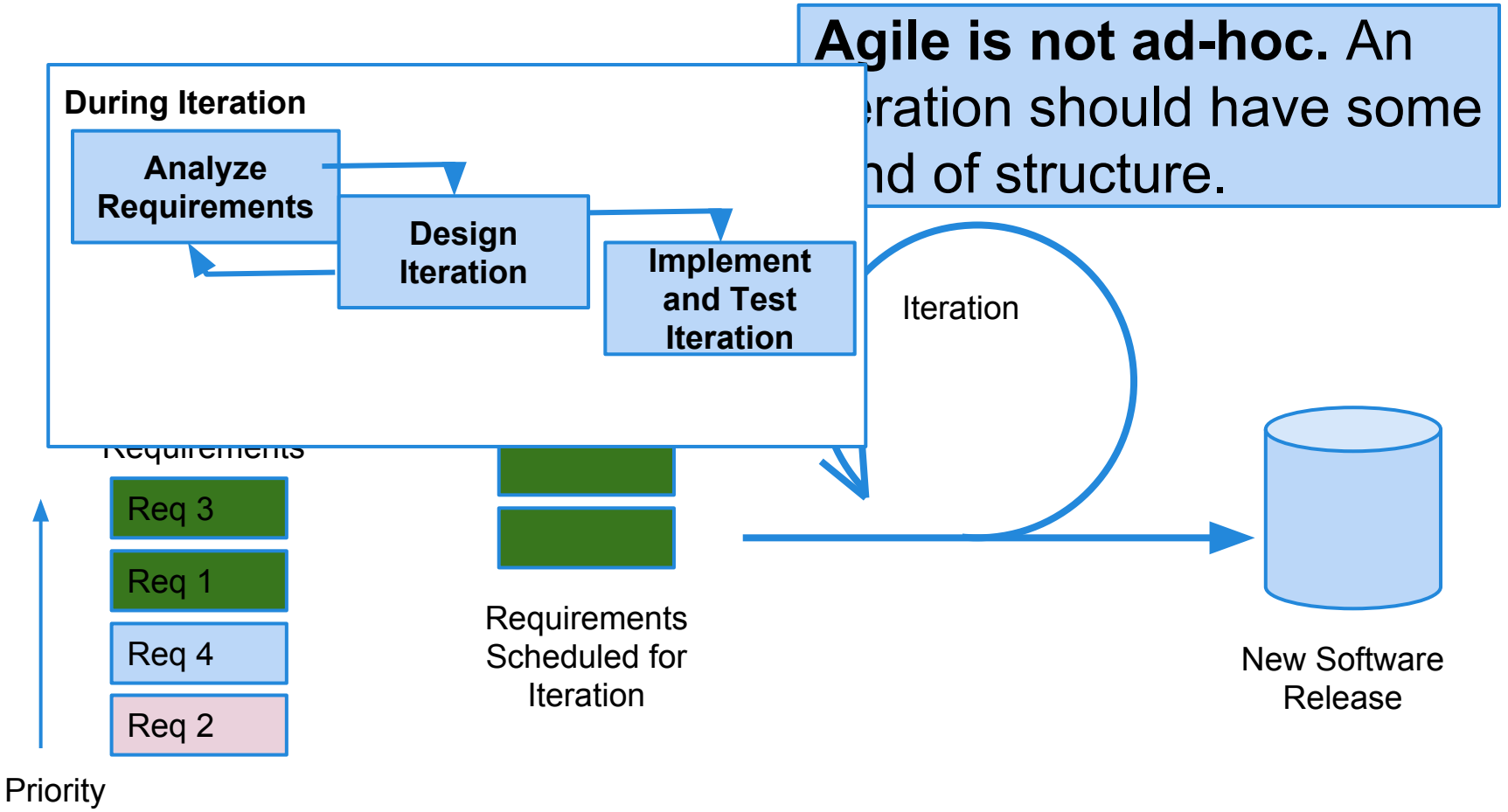


# Wait… Aren't incremental and iterative the same thing?

- **Incremental:** Add new features to build a progressively more complete system over time.
- **Iterative:** Deliver a series of progressively more complete prototypes over time.
- *Aren't these the same thing?*

Incremental is writing an essay one "perfect" sentence at a time. Iterative is writing a complete rough draft, then improving it through a complete revision.

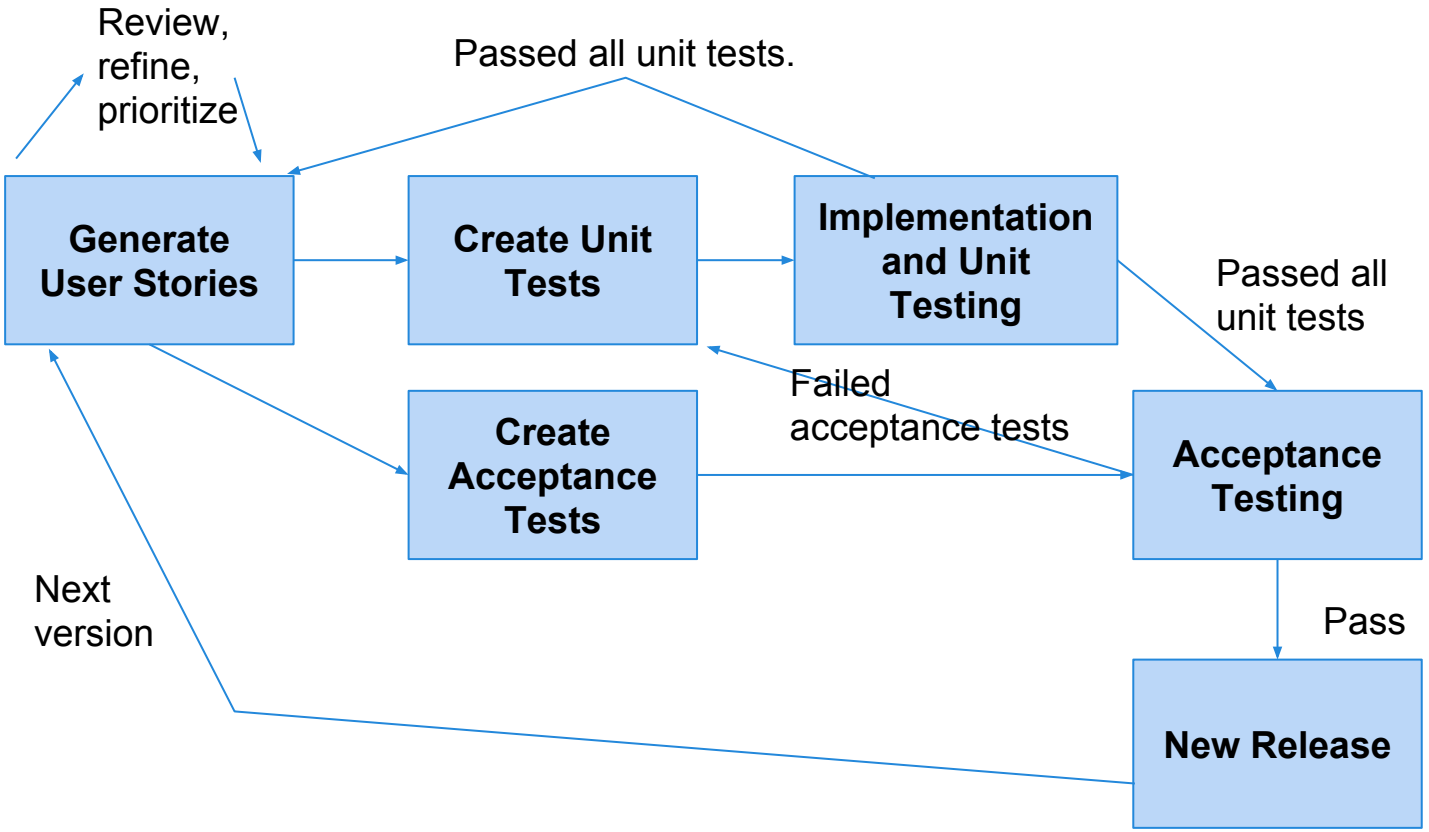
# The Agile Model



# Extreme Programming

- Extreme Programming model emphasizes:
  - Simplicity over generality.
  - Communication over structured organization.
  - Frequent changes over big releases.
  - Continuous testing over separation of roles and responsibilities.
  - Continuous feedback over traditional planning.
- Prescribes rules regarding planning, managing, designing, coding, and testing.
- Customers involved in requirement specification and acceptance testing.

# Testing in Extreme Programming



# Monitoring and Improving the Process

# Monitoring the Process

- Must monitor progress of all quality activities.
- Identify deviations from the plan as early as possible and take corrective action.
- Relies on a plan that is realistic, organized, and detailed.
  - Clear, unambiguous milestones.
- Process must be **visible** - able to be monitored and assessed.

# Process Visibility

- Activity completion must be distinguished from activity termination.
  - Must include metrics of the thoroughness or completeness of an activity.
- Must make decisions based on overall picture of project progression.
  - Monitoring must collect aggregate measures about activity results.
  - One measure - **number of faults revealed and removed**, tracked against time.
    - Can be compared to past projects.

# Detecting Anomalies

- Unexpected pattern in fault detection implies problems in process.
  - Early decline in growth of fault detection usually implies ineffective quality assurance efforts.
  - Growth rate that remains high implies that quality goals may be met late or not at all.
    - May indicate weaknesses in fault removal, lack of discipline in development.
- If faults remain open longer than expected, there are process issues.
  - Confirmed when number of open faults does not stabilize at acceptable level.

# Improving the Process

- Many faults are rooted in process flaws.
- Such faults can be prevented by improving the process.
- Root cause analysis (RCA) is a technique for identifying and removing process faults.
  - Selects significant classes of faults and traces them to their original causes.
  - Four steps: What, When, Why, and How

# What are the Faults?

- Identify a class of important faults.
- Faults classified by severity and type.
  - Severity: cosmetic, moderate, severe, critical
  - Type does not use a predefined set of categories.
    - Categorization based on the project type and expected sources of faults.
    - Granularity based on focus of development.
      - If interface issues are the focus, apply finer classifications to interface faults, and coarser to other types.
    - Classification scheme altered after identifying and removing the cause of a fault type.

# When?

- When did faults occur?
  - Can we determine when a fault was introduced?
  - During coding, design, specification, etc.
- When were the faults found?
  - Can we determine when the fault was found by a quality process?
    - Integration testing, design inspection, etc.

# Why did the Fault Occur?

- Trace representative faults back to causes.
  - Identify the “root” cause associated with most faults in this class.
- Analysis attempts to explain the error that led to the fault, then the cause of the error, the cause of the cause, and so on.
  - Tracing is a manual process, requires experience and judgement.
  - Each step requires information about the class of fault and the process.
    - Acquired through inspection of documentation and interviewing developers.

# Example

- Memory leaks are the most significant class of faults.
  - Moderate frequency, severe impact, high cost to diagnose and repair.
  - Result of forgetting to release memory in exception handlers.
  - Result of being unable to determine what needs to be cleaned up in exception handlers.
  - Result of the resource management scheme assuming normal flow of control.
  - **Root problem:** exceptional conditions were an afterthought dealt with late in design.

# How Could Faults be Prevented?

- Improve the process by removing root causes or making early detection likely.
- Can involve minor tweaks to process…
  - Adding consideration of exceptional conditions to design checklists.
- Or major overhaul…
  - Making explicit consideration of exceptional conditions a part of all analysis and design steps.
- Requires judgement, should be followed-up on in future projects.

# We Have Learned

- Quality must be a part of all development stages and must be planned for.
- Verification activities can be integrated into all development processes, and planned as part of each phase.
- The quality process must be monitored as it executes to detect issues and correct them.
- RCA can be used to diagnose process issues, and prevent them in future projects.

# Next Time

- Presentations Begin
  - April 19, 21, **26**, May 3
  - 12 minute talk.
  - Not much room for problems, so be prepared.
    - Bring slides on a thumb drive or e-mail them to me.
  - Attendance will be taken on all presentation dates.

- Homework:
  - Assignment 5 - April 25.