CS 3773
Software Engineering
Lecture 2

Dr. Mark Robinson

Office: NPB 3.350

The SE Process

- Definition: “A **series of steps** involving **activities**, **constraints**, **resources** that produce an **intended output**”
- What does this sound like?
- Note: a SE Process is AKA a software’s **lifecycle**

Process Terms

- **Activity**: a type of work performed during the process
- **Constraint**: a restriction imposed on the process (e.g., deadlines, a schedule, money, target platform)
- **Resource**: Input consumed during process activities (e.g., labor, time, money)
- **Output** (product): A result created during one or more process activities (e.g., specifications, executable software)

Process Characteristics

- Prescribes the major activities and when each occurs
- Can produce intermediate product and final product
- Can be composition of mini-processes
- Each activity has entry and exit criteria (clear when it starts and stops)
- Sequential (know which activity is next)
- Each activity has goals
- Constraints can apply to one or more activites, resources, products

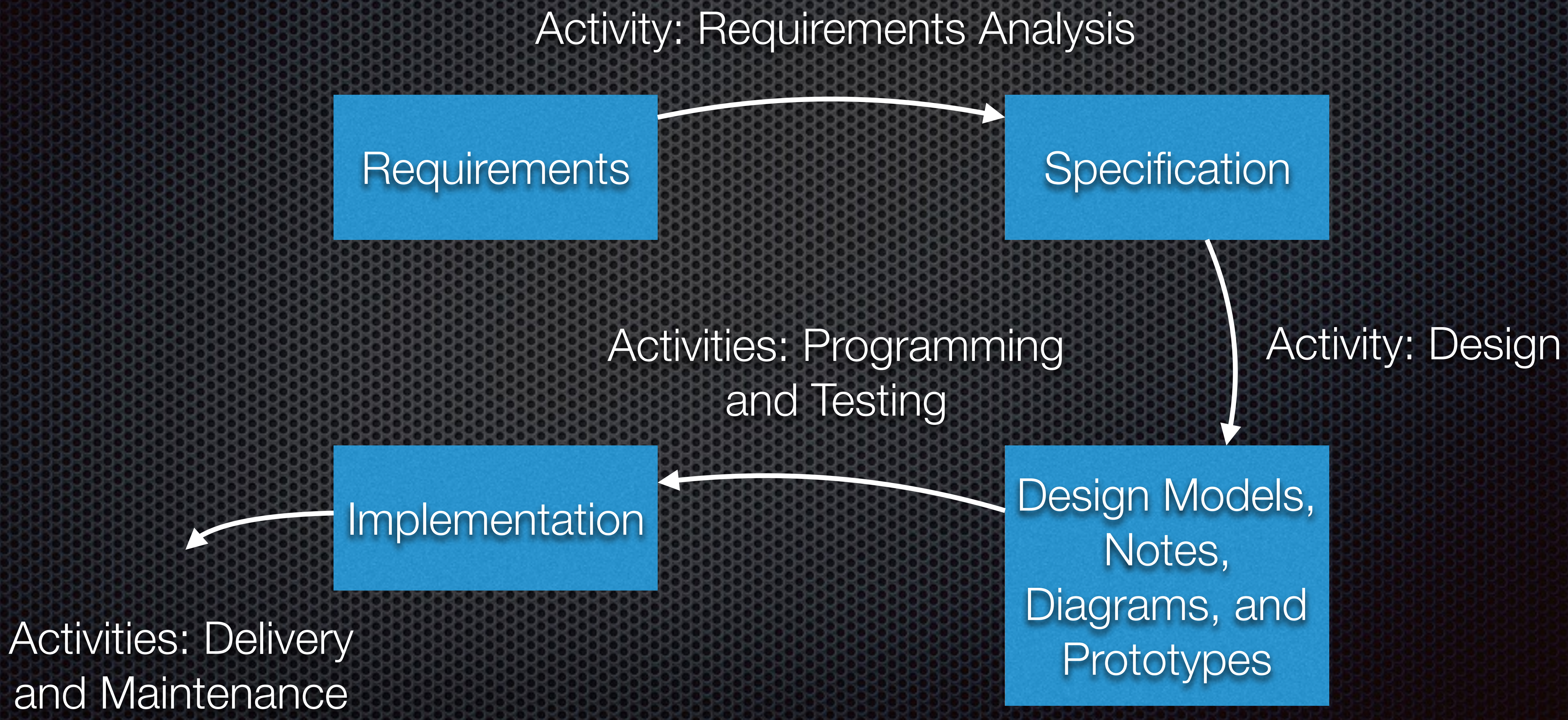
Why Use a SE Process?

- Can you build quality software by just sitting down and writing code?
- Using a Process brings:
  - **Repeatability**: consistent products of desired quality
  - **Efficient Training**: easier to train new people with a well-documented process (and again, it’s more repeatable)
  - **Room for Improvement**: easier to measure, analyze and improve parts of the process to yield better results

Process Stages/Activities

1. requirements analysis and definition
2. system design
3. program design
4. programming
5. unit/integration/system testing
6. delivery
7. maintenance

Work Products and their Transformative Activities



Activity: Requirements Analysis

Requirements

Specification

Activity: Design

Design Models, Notes, Diagrams, and Prototypes

Activities: Programming and Testing

Implementation

Activities: Delivery and Maintenance

Work Products

- **Requirements**: what the system is supposed to do in terms of behavior, data, and constraints; language: plain and anything; produced by clients, end-users, etc.
- **Specifications**: developer interpretation of what the system is supposed to do; extracted from the requirements; language: structured plain; produced by developer, RE, etc.
- **Design**: blueprints of how the system will be built; language: model notation, diagrams, prototypes; produced by developer, architect
- **Implementation**: the working software product; handwritten and/or COTS, 3rd party libraries, frameworks, databases; language: executable; produced by developer, programmer, integrator, testers

Process Model

- Definition: a specific configuration of process activities that can guide real software development
  - It’s a model (i.e., abstraction) of how real development will work, from activity to activity

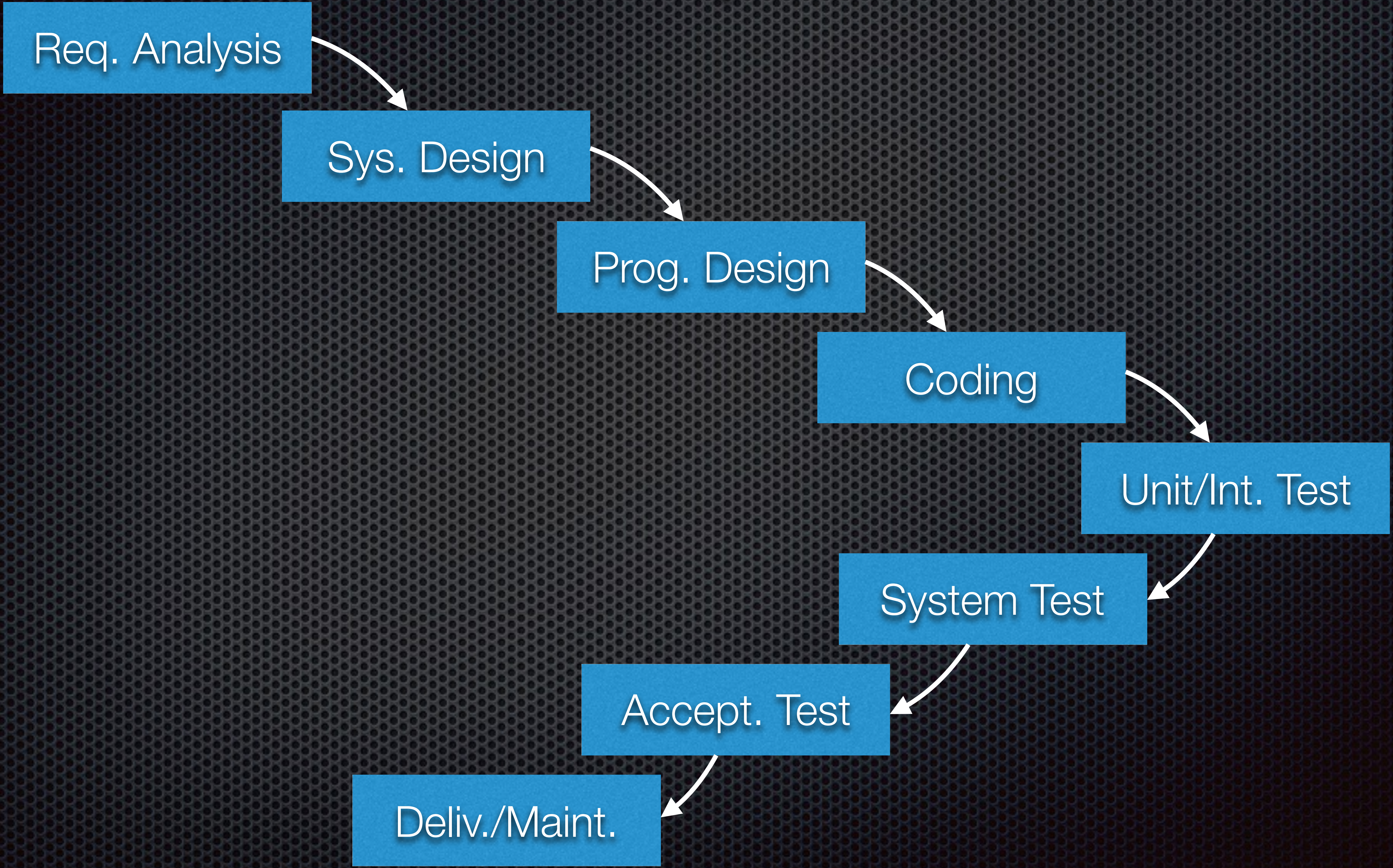
Why Model a Process?

- Provides a roadmap for development
- Documenting creates common understanding
- Helps find bugs in the process, even before dev starts
    - Model should match the special needs/ circumstances of the real project

Waterfall Model

- Linear sequence of activities
  - Proceed 1 activity at a time and no going back
- The prescriptive process for several decades
- Part of DoD standard for SE

Waterfall Example



Req. Analysis

Sys. Design

Prog. Design

Coding

Unit/Int. Test

System Test

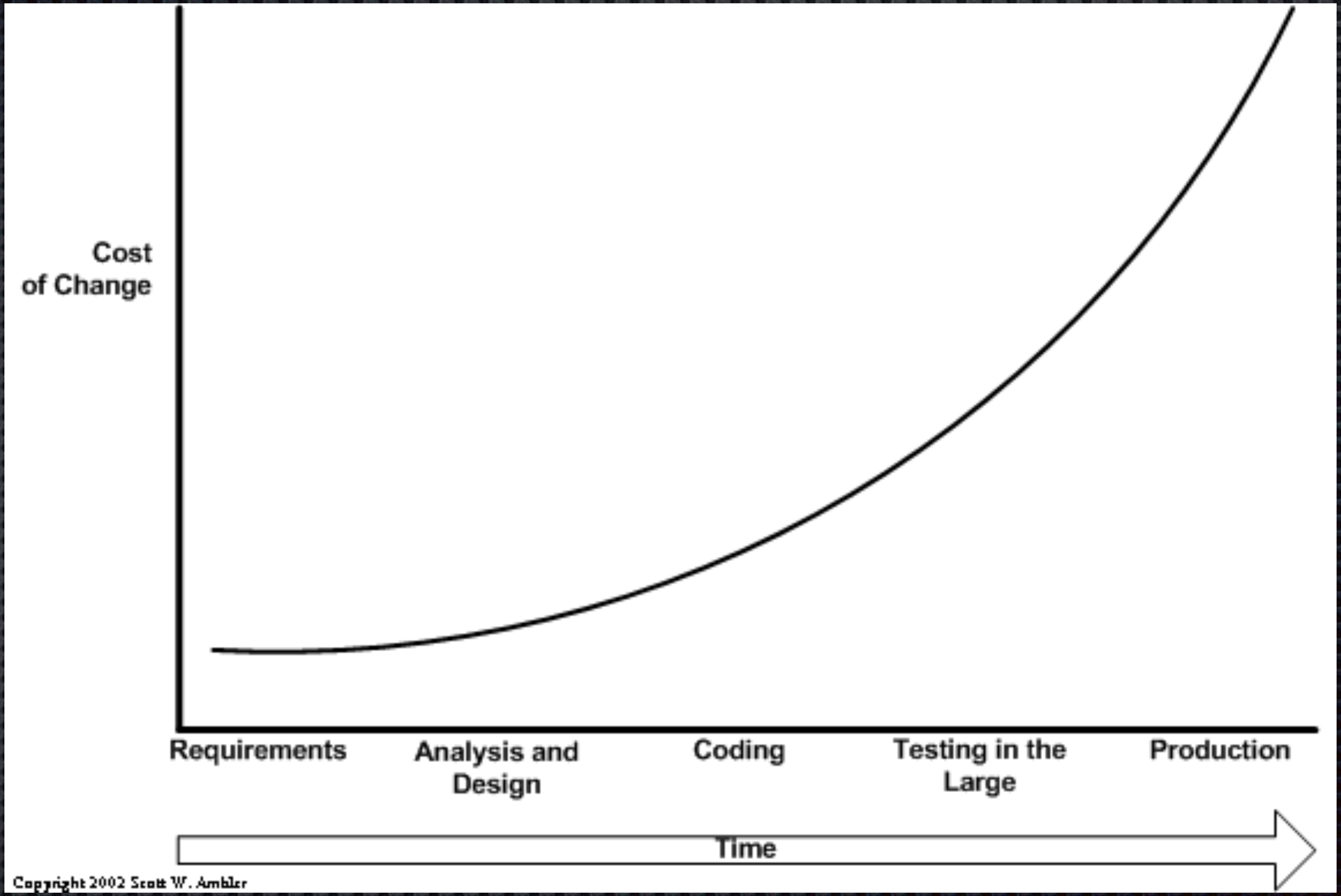
Accept. Test

Deliv./Maint.

Waterfall Notes

- Use when **all requirements are known** (not realistic)
  - Can be extremely efficient (minimal context switching)
- Cons:
  - Clients rarely know all requirements upfront
  - Doesn’t allow change during development
  - Clients must wait until software is finished

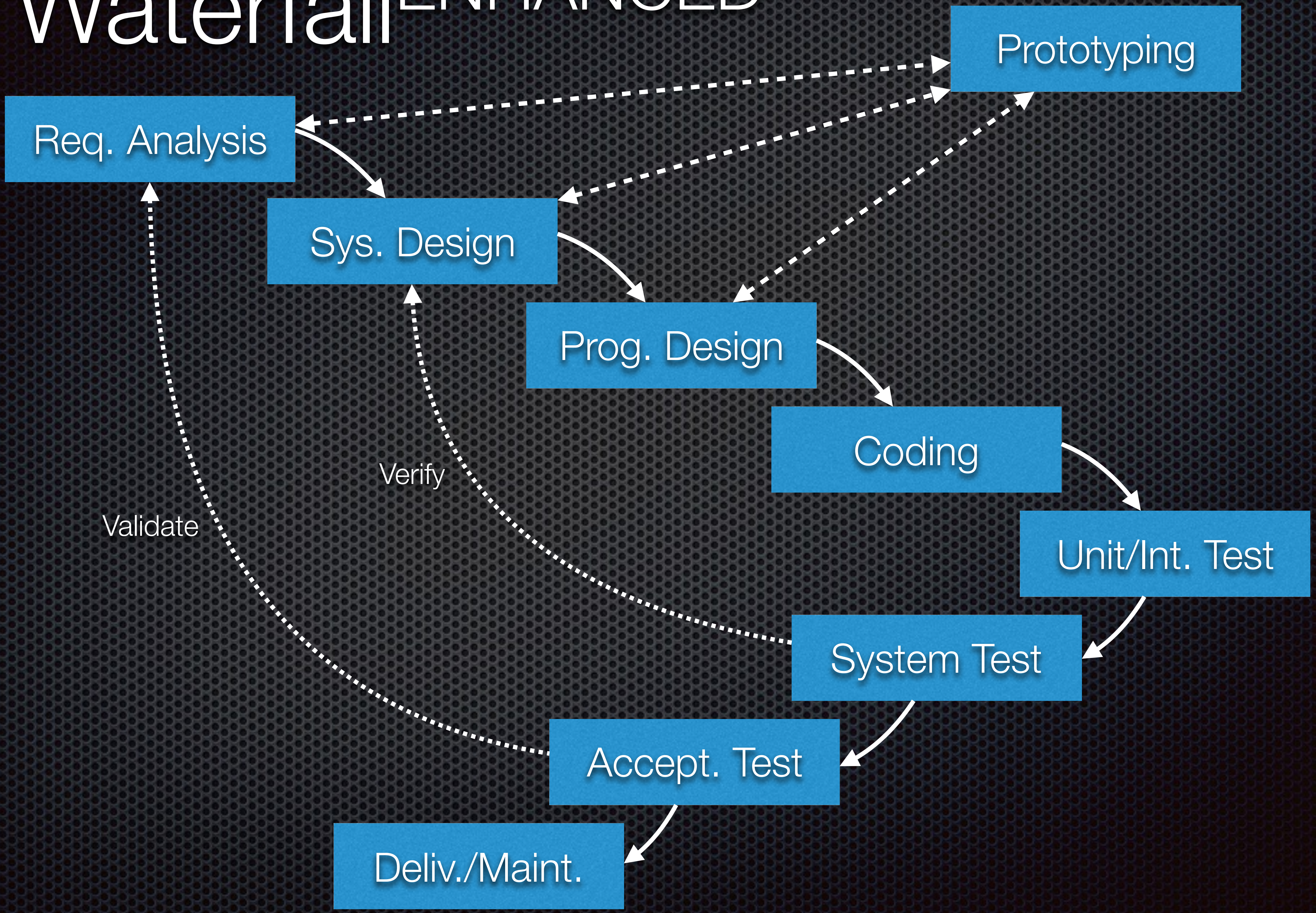
Waterfall Cost of Change/ Requirement Defects



Risk

- Waterfall requires an activity to be 100% correct/ finished before moving to next activity
  - How do you know if an activity is done? if it is right?
- **Risk**: the chance that something bad will happen (e.g., an activity's work product is not 100% correct)
- How can we **reduce risk** in the Waterfall process to better trust that when activity is done, it is 100% right?

Waterfall$^{\text{ENHANCED}}$



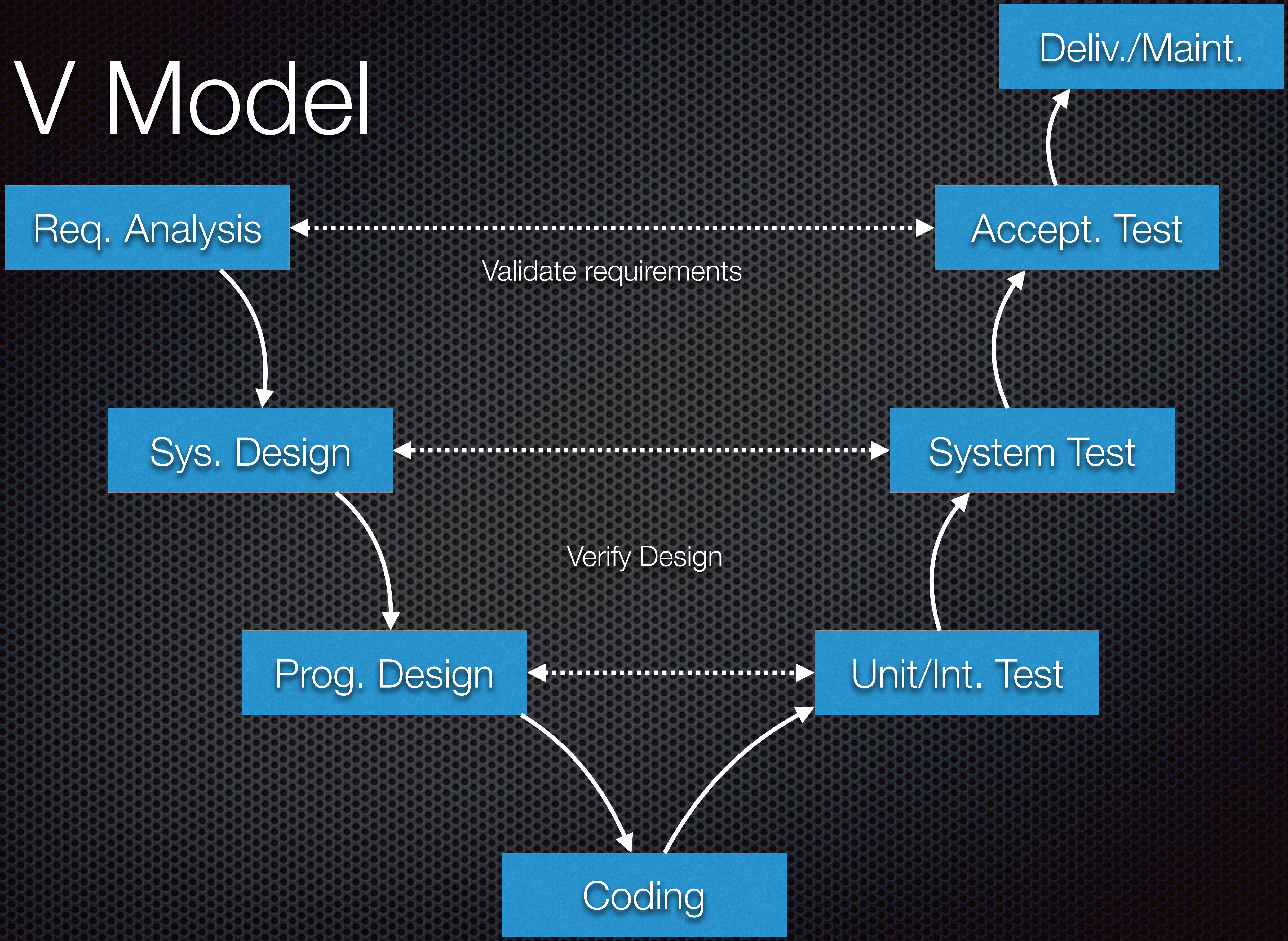
Some Quality Terms

- **Validation**: check if all requirements accounted for; is this what the customer wants?
- **Verification**: everything works correctly (i.e., according to the specifications)
- Not the same thing: if you build 100% bug-free software (verified) but it doesn’t do what customer wants (not validated), software will not be used (not a success)

V Model

- Variation of Waterfall
- Close couples different levels of testing with analysis and design
- Divided into levels and sides: left side is production, right side is testing
- Iterates each level between production and testing UNTIL production is correct

V Model



Req. Analysis

Accept. Test

Validate requirements

Deliv./Maint.

Sys. Design

System Test

Verify Design

Prog. Design

Unit/Int. Test

Coding

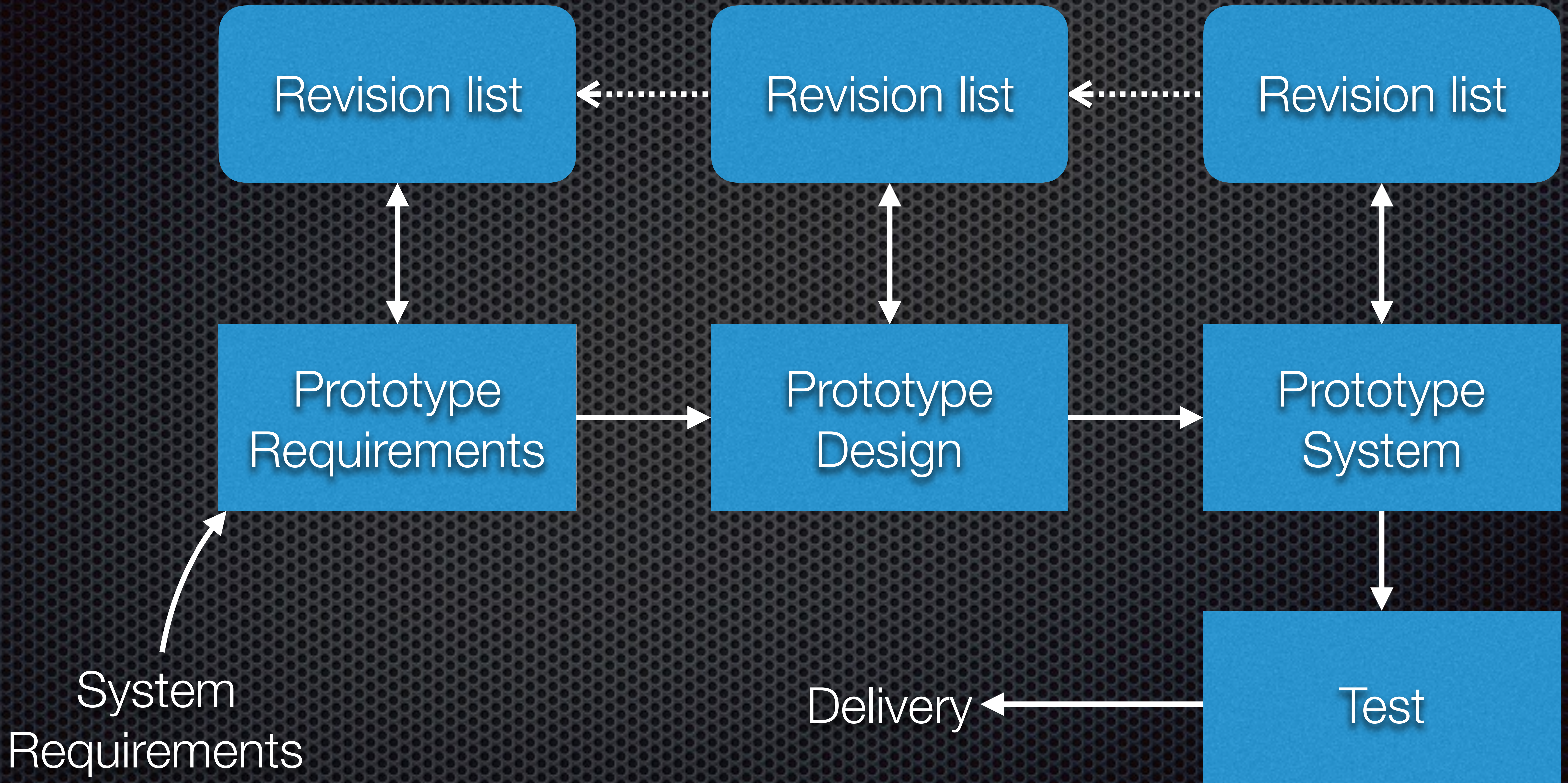
V Model Notes

- Requirement changes are not be detected until just before delivery (where changes are very expensive)
- Customer must wait until software is finished

Prototype Model

- **Prototype**: a throw-away demonstration
- Build and check prototypes at each production activity to ensure project matches customer requirements
- Validation at every step reduces overall risk of having to make late changes
- Last system prototype is the software to deliver

Prototype Model



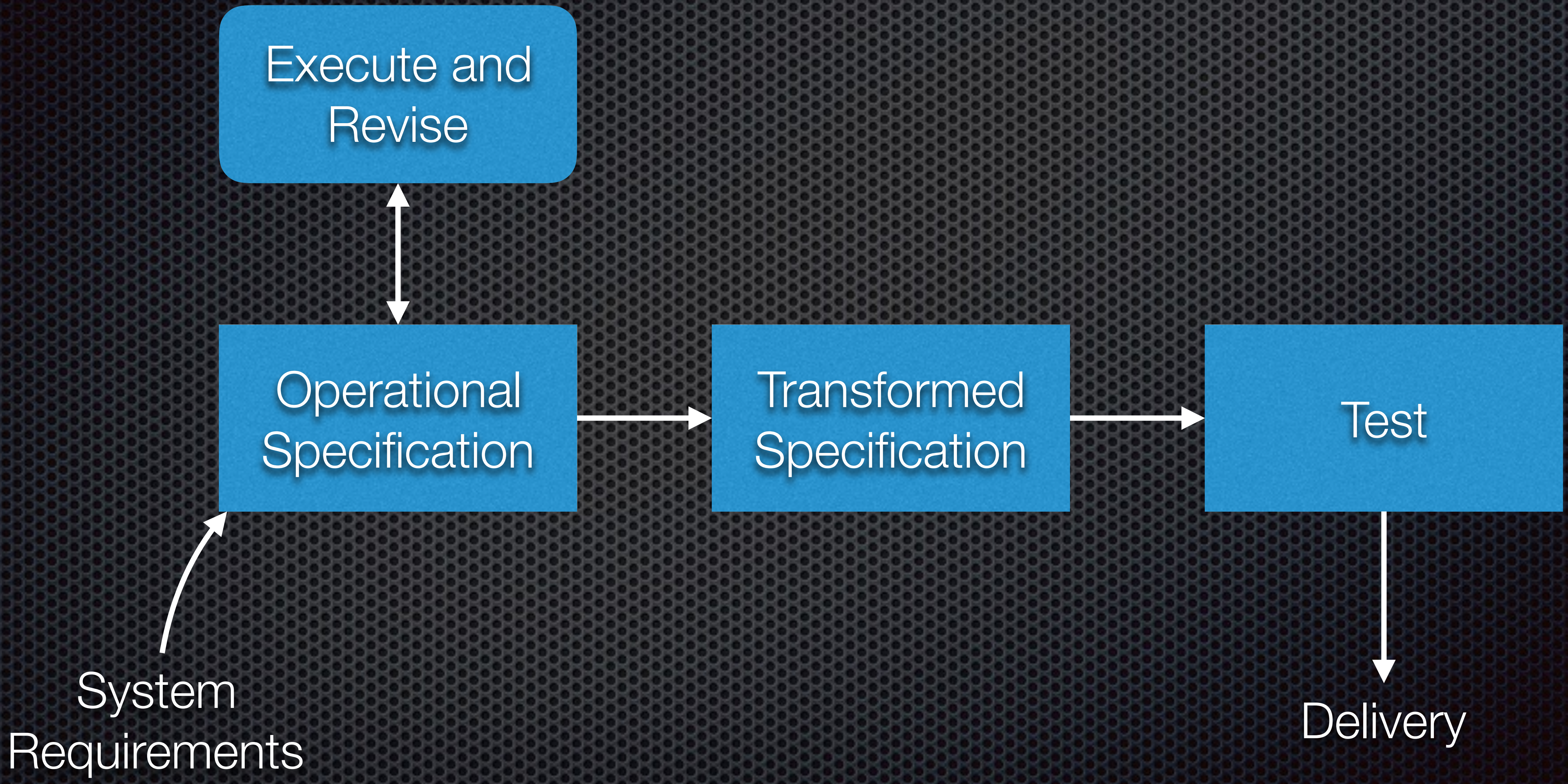
Prototype Notes

- Lots and lots of prototyping
- Prototypes should be throw-away
    - Must resist temptation to keep product that should be discarded

Operational Specifications Model

- Specify/Design/Build/Test/Validate subsets of requirements
- Integrate validated parts into a deliverable system

Operational Specifications



Execute and Revise

Operational Specification

Transformed Specification

Test

System Requirements

Delivery

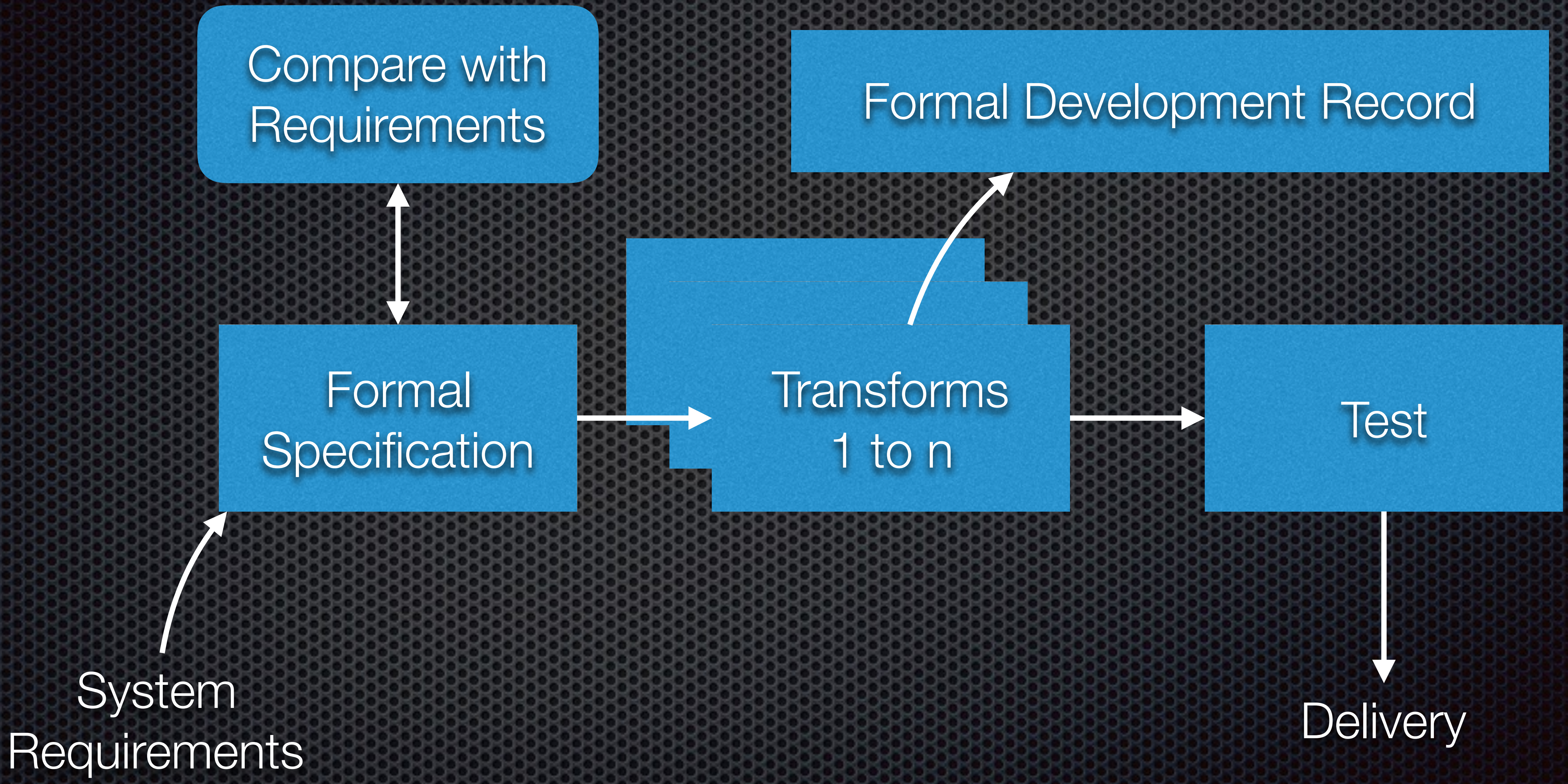
Operational Specifications Notes

- Validated parts may perform differently when integrated with other validated parts
  - 100 concurrent users put a different load on system than 100 concurrent users all fetching 10,000 records
- Late requirement change may invalidate lots of validated parts

Transformational Model

- Given a formal specification (via proof, logical notation or model)
    - Automatically generate the implementation/delivered system
- Sequence of transformations controlled in a **formal development record**

Transformation Model



Compare with Requirements

Formal Development Record

Formal Specification

Transforms 1 to n

Test

System Requirements

Delivery

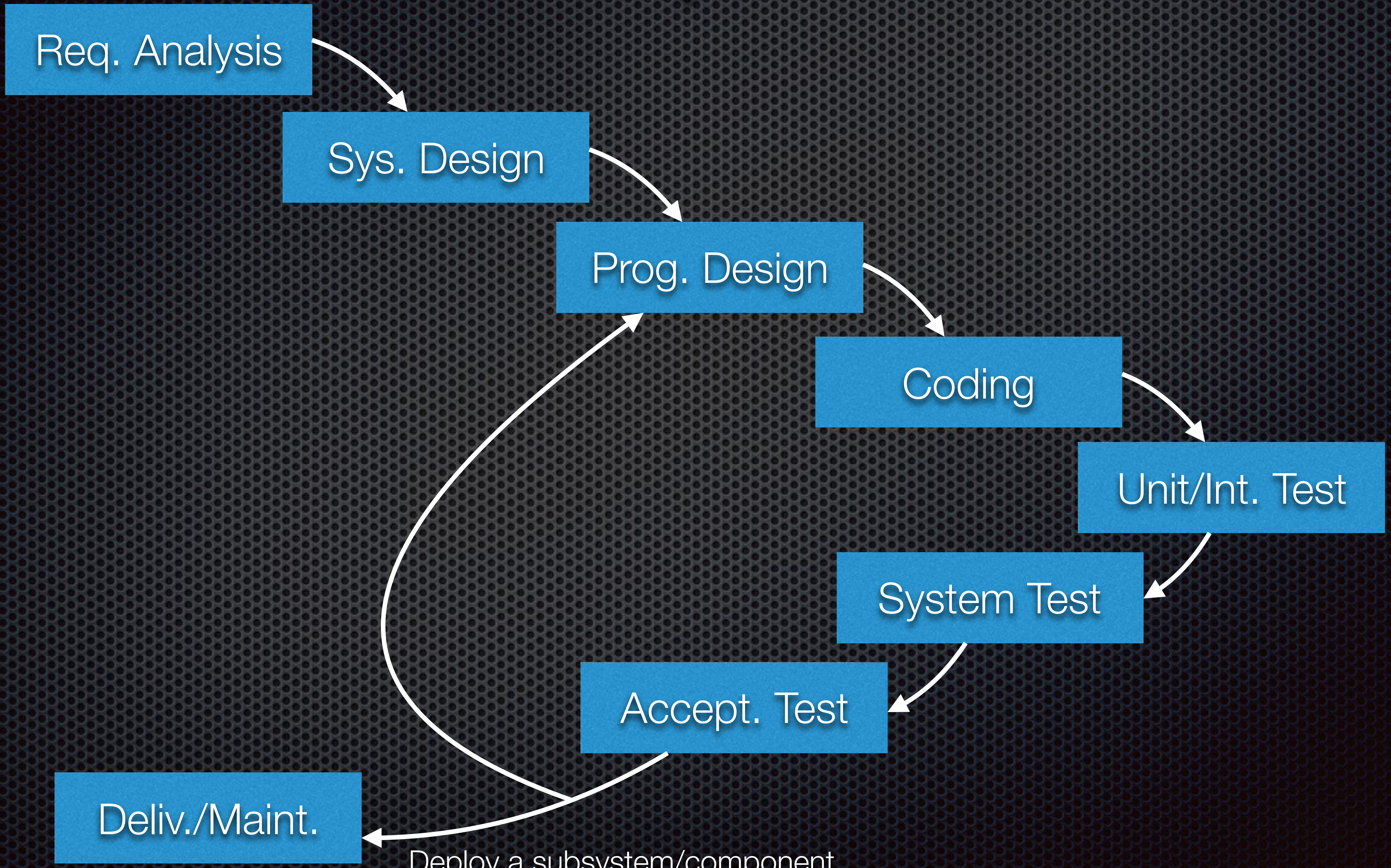
Transformational Notes

- Formal specifications are **extremely** difficult and costly to produce
  - In many situations, can only be produced for small, critical components
- Automatic transformation/implementation is *probably* infeasible for complex software

Incremental/Iterative Models

- Has loops between some of the activities
  - Each iteration develops either:
    - A completed component or subsystem (incremental)
    - Several incomplete components (iterative)
- Working product delivered at each iteration
  - Functionality delivered based on importance to client or development process

Iterative Example



Req. Analysis

Sys. Design

Prog. Design

Coding

Unit/Int. Test

System Test

Accept. Test

Deliv./Maint.

Deploy a subsystem/component

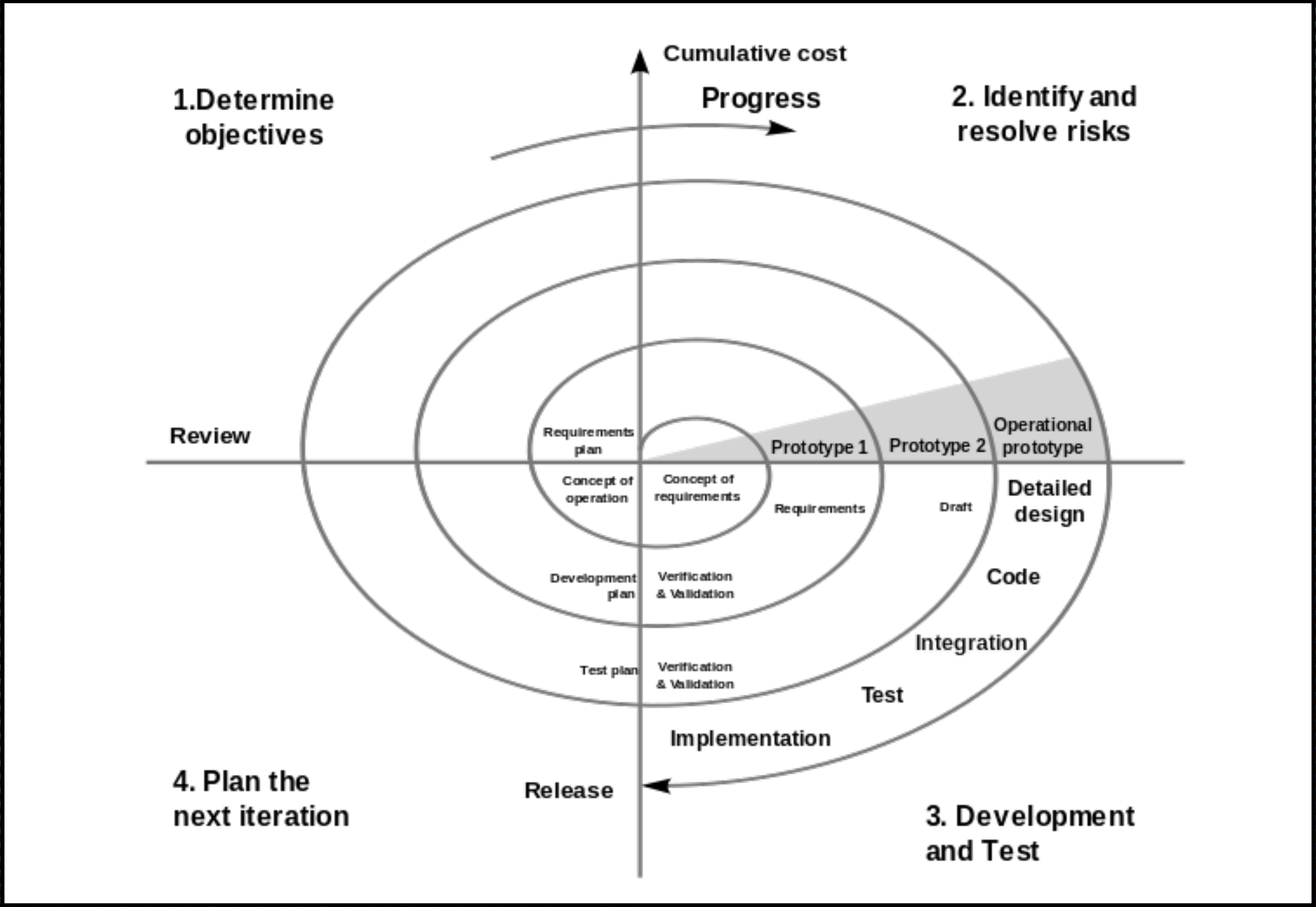
Incremental/Iterative Notes

- Good process for when requirements unclear
    - Or deadline is too aggressive for finished product
    - Or resources not all available
- Cons:
    - Total time to deliver might be longer than Waterfall (more context switching)
    - Possible work/artifact redundancy or waste

Spiral Model

- Each iteration decreases product’s risk and increases product’s definition
    - 1st iteration may result in formal specs or prototype
- Risk considered and managed each iteration
- Use when: risk analysis and feedback are important
- Cons: needs risk expertise and may be slower

Spiral Model



(Rational) Unified Process

- Object-oriented, iterative process (similar to incremental/evolutionary)
  - Different terms for activities and overlaps some of the activities in previous process models
- **Focuses on requirements analysis and design** (uses modeling to visualize, analyze, and reason)
- Based on the UML (Unified Modeling Language)
  - Essentially “how to” use the UML
- RUP has some heavyweight tool support (e.g., Rational Rose)