Homework (Due on Dec 19 2017)

- Give your thoughts on whether the notion of "Software Crisis" still exists in modern day software engineering.
- Compare the limitations and strengths of various software lifecycle models.
- What role does open source play in the software lifecycle?

Software Lifecycle Models

Prof. Jonathan Lee (李允中)
CSIE Department
National Taiwan University

What’s the Problem?

- 84 % of all software projects do not finish on time and within budget (Survey conducted by Standish Group)
  - 8000 projects in US in 1994
  - More than 30% of all projects were cancelled
  - 189% over budget
- Key issues:
  - Software firms are always pressured to perform under unrealistic deadlines.
  - The clients ask for new features, just before the end of the project, and unclear requirements.
  - Software itself is awfully complex.
  - Uncertainties throughout the development project.

Real Cases

- Bank of America Master Net System
  - Trust business. 1982.
  - Spend 18 months in deep research & analysis of the target system.
    - Original budget: 20 million.
    - Original Schedule: 9 months, due on 1984/12/31.
    - Not until March-1987, and spent 60 million.
    - Lost 600 millions business
  - Eventually, gave up the software system and trust accounts transferred.
- Other cases:
  - Explosion of Ariane 5 prototype in 1996
  - Explosion of Boeing's Delta III rocket.

Software Projects Survey: Standish Group Chaos Report

- Software projects (in US): success rate means the project is completed **on time** and **within budget**.
- Projects being **challenged** means over budget and/or with less than the required features and functions.
- Projects being failed means cancelled prior to completion or delivered but never used.

	Year 2015	Year 2014	Year 2010	Year 2008	Year 2002	Year 2000	Year 1998	Year 1996	Year 1994
Successful	29%	28%	37%	32%	34%	28%	26%	27%	16%
Challenged	52%	55%	42%	44%	15%	23%	28%	40%	31%
Failed	19%	17%	21%	24%	51%	49%	46%	33%	53%

Any Improvement?

- What Software Engineering community has been doing so far to improve software project success rate.

- Engineering practice: Extreme programming, Aspect-oriented programming, Design patterns and frameworks, and etc.
- Project management: WBS, SCRUM
- Process improvement: Model-based process improvement (CMMI)
- Infrastructure: **DevOps** (continuous integration and delivery)

Software Crisis

- Coined in the NATO Software Engineering Conference in 1968.
- Refers to the development and maintenance of correct, understandable, and verifiable software.
  - Increasingly improved hardware performance
  - Productivity: build new programs from scratch
  - Maintenance: maintain existing programs
- Root causes:
  - Software Complexity, User's Expectations, and Changes over time.

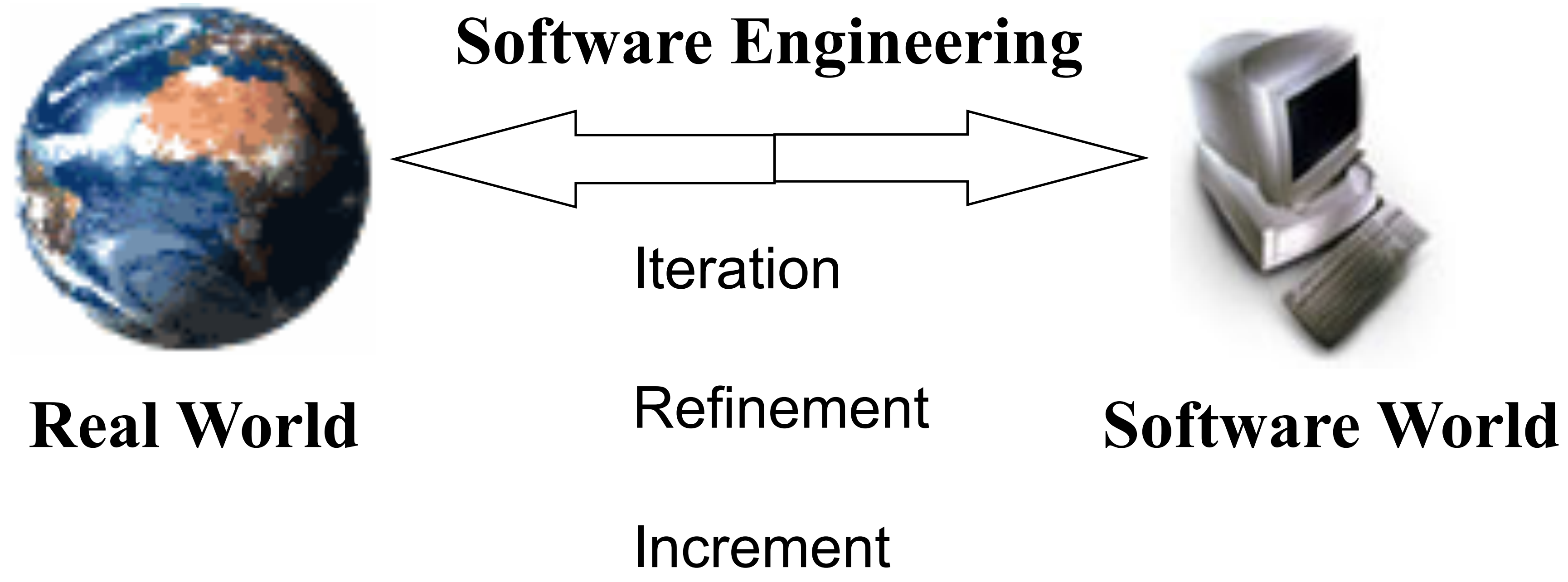
F. Brooks: No Silver Bullet

- Essence: the difficulties inherent in the nature of software
  - Complexity: development process, application domain, internals of the system (e.g., number of states, size of software, functionality, structures, and behaviors).
  - Conformity: software must adapt to interact with people according to their diverse needs; therefore, software cannot be completely represented formally.
  - Changeability: software component is usually the easiest one to modify; (application changes, software changes).
  - Invisibility: software structure is hidden, only behavior can be observed when executing the software.

F. Brooks' Accidents

- Accident: difficulties arise in representing, testing the conceptual constructs (e.g., software development process), examples of accidental difficulties:
  - accidental complexity: abstract program vs concrete machine program; solved by HLL.
  - slow turnaround: batch programming vs time-sharing (shortest response time); solved by time-sharing.
  - no standard formats: individual programs vs integrated library, standard formats; solved by unified programming environment.

What’s Software Engineering?



Software Engineering

- Software engineering is the application of engineering to software.

- The application of systematic, disciplined, and quantifiable approaches to the design, development, operation, and maintenance of software, and the study of these approaches.

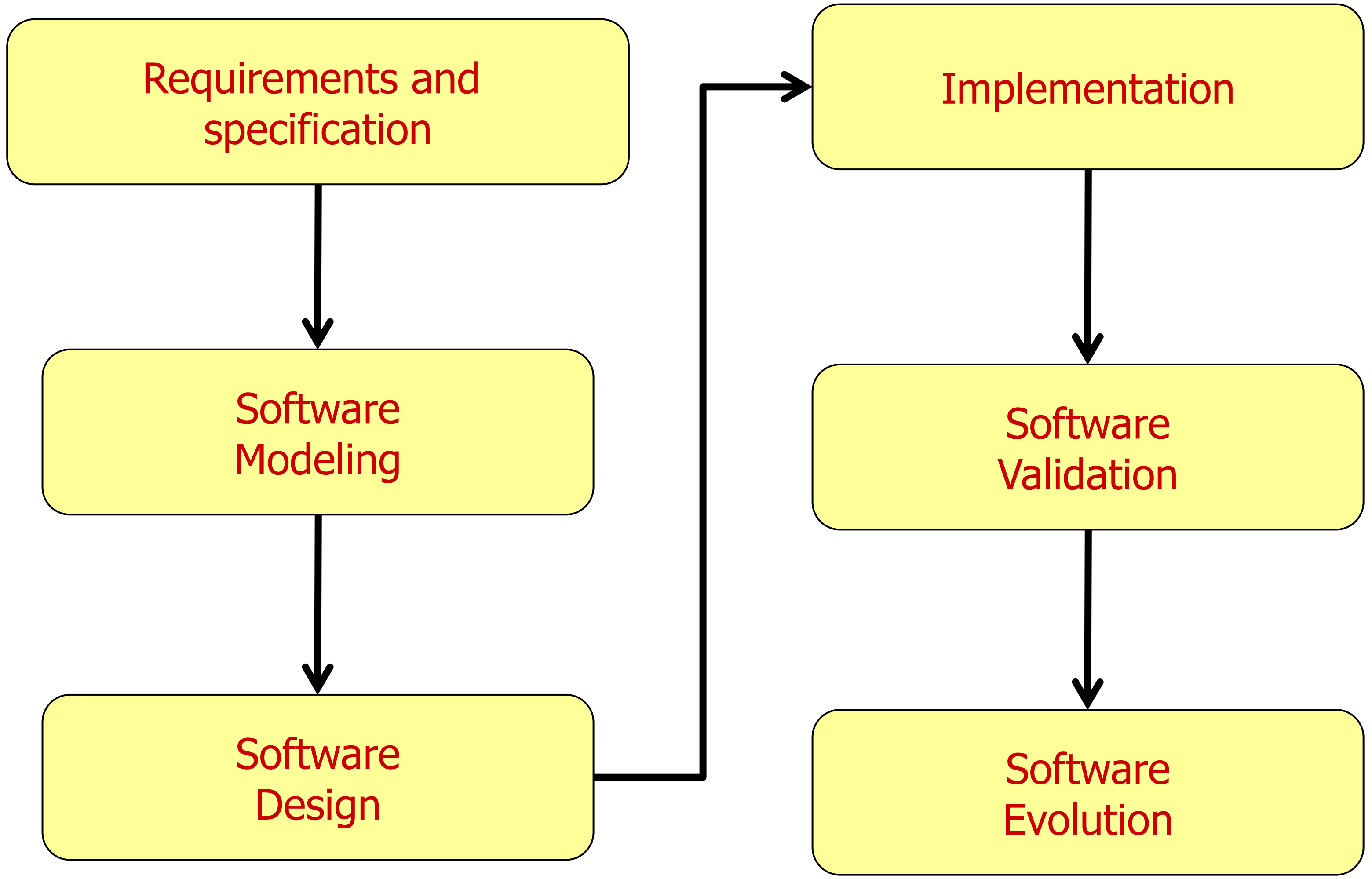
- First introduced in NATO Software Engineering Conference in 1968.

Software Process

- A *software process* is a set of activities, methods, practices and transformations that people employ to develop and maintain software and the associated products, including requirements documents, project plans, design documents, code, test cases, and etc.

- The process of building a product is called a **lifecycle** because it describes the life of that product from conception through to its implementation, delivery, use and maintenance.

Process Activities



Requirements & Specification

- The process of establishing what **services/functions** are required and the **constraints** on the system’s operation and development.

- Requirements analysis
  - **Feasibility study**
  - Requirements elicitation and analysis
  - Requirements specification
  - Requirements validation.

Software Design

- Software design: Design a software structure that **realizes the specification**, including the following: System architecture design, class diagrams design, user interface design, data structure design, and algorithm design.

- **A good design should be aligned with the requirements specifications, adopting low coupling principle (with the future changes in mind), and taking test cases into consideration.**

Implementation & Debugging

- Translating a design into a program and removing errors from that program.

- Programmers carry out some program testing to discover faults in the program and remove these faults in the debugging process.

Verification & Validation

- **Verification:** a system conforms to its specification.
- **Validation:** a system meets the requirements of the system customers.
- V&V involve checking, reviewing, and testing.
- System testing involves executing the system with test cases that are derived from the specification of the real data to be processed by the system.
- **Test cases as an important valuable and asset.**

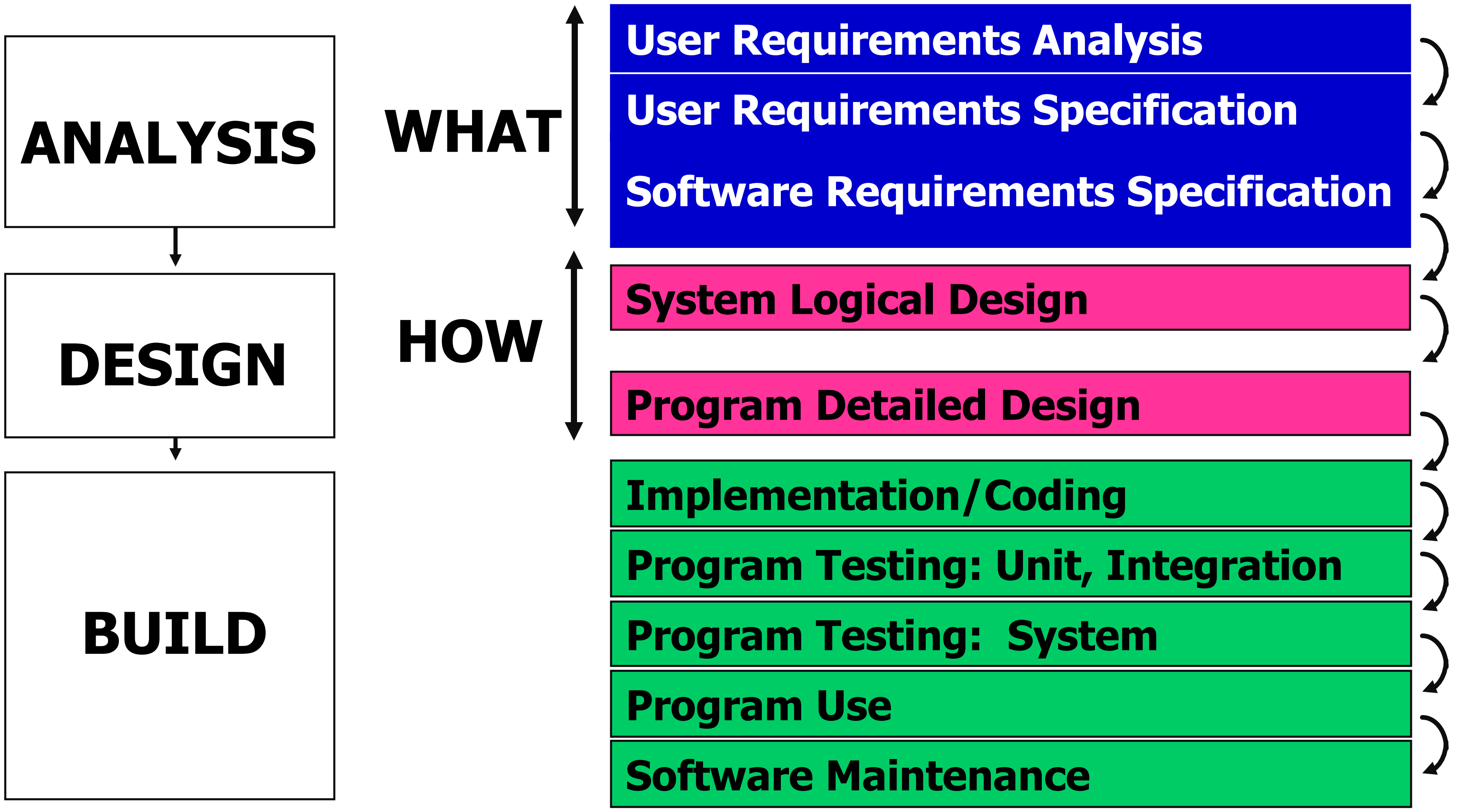
Software Evolution

- Software is inherently flexible and gets **changed all the time**.
- As requirements change through changing business circumstances, the software that supports the business must also evolve and change.

Software lifecycle Models

- ❑ Waterfall Model
- ❑ V Model
- ❑ Prototyping Model
- ❑ Component-based Software Process Model
- ❑ Spiral Model
- ❑ Automatic Synthesis Model
- ❑ Object-oriented Software Process Model
- ❑ Unified Process Model

Waterfall Model



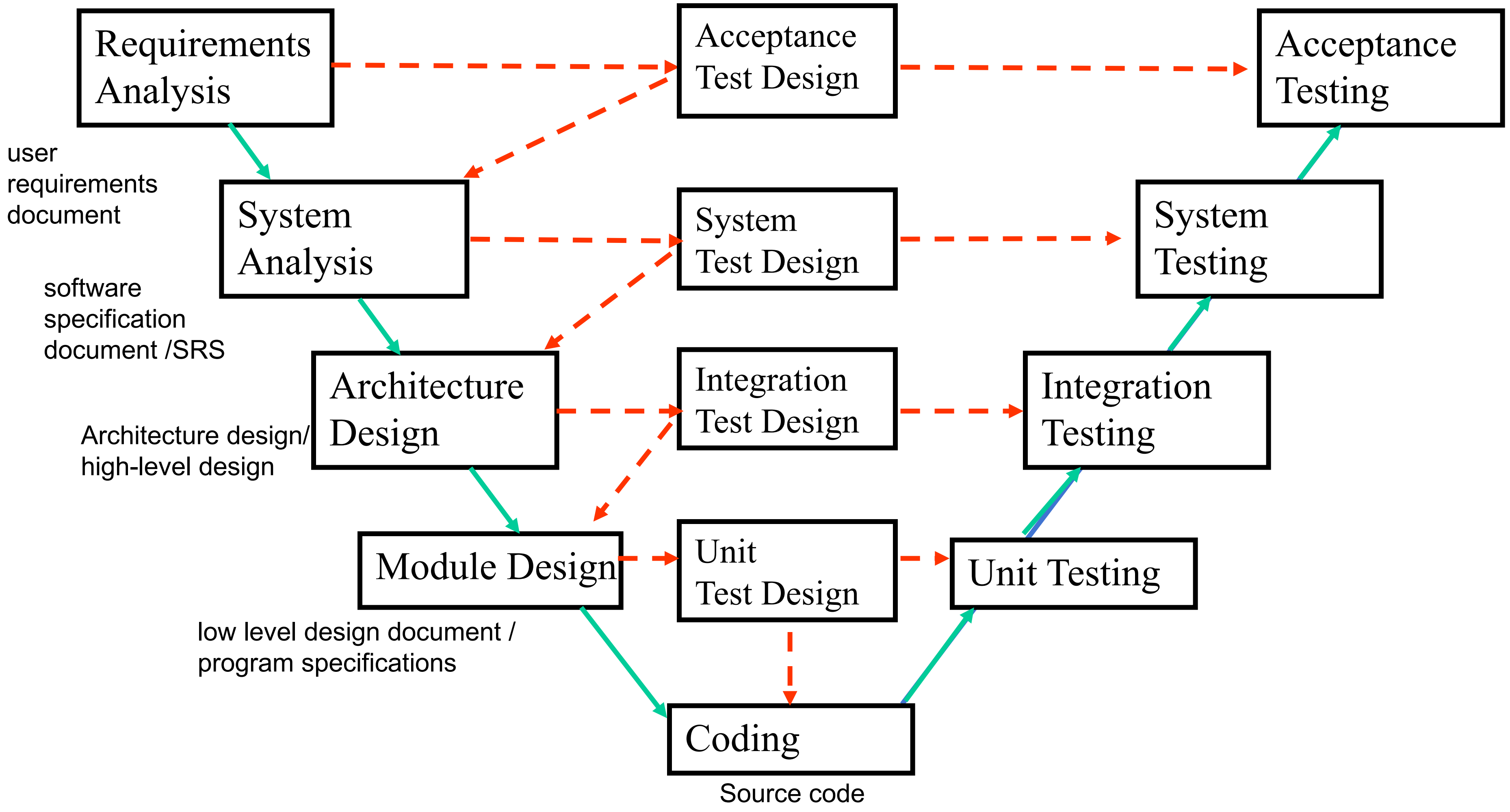
Pros & Cons

- Benefits of the waterfall model
  - Being a **linear model**, it is easy to implement.
  - Output (documentation) is generated after each stage.
- The main drawbacks of the waterfall model
  - Assume the requirements are **well-understood and complete**.
  - Hard to accommodate **changes** after the process is underway.
  - One phase has to be complete before moving onto the next one.
  - **Commitments** must be made at an early stage in the process.

V Model

- Emerged in reaction to some waterfall models that show testing as a single phase following the traditional development phases of requirements analysis, high-level design, detailed design and coding.
- The V model portrays several distinct testing levels and illustrates how each level addresses a different stage of the software lifecycle.
- The V shows the typical sequence of development activities on the left-hand (downhill) side and the corresponding sequence of test execution activities on the right-hand (uphill) side.

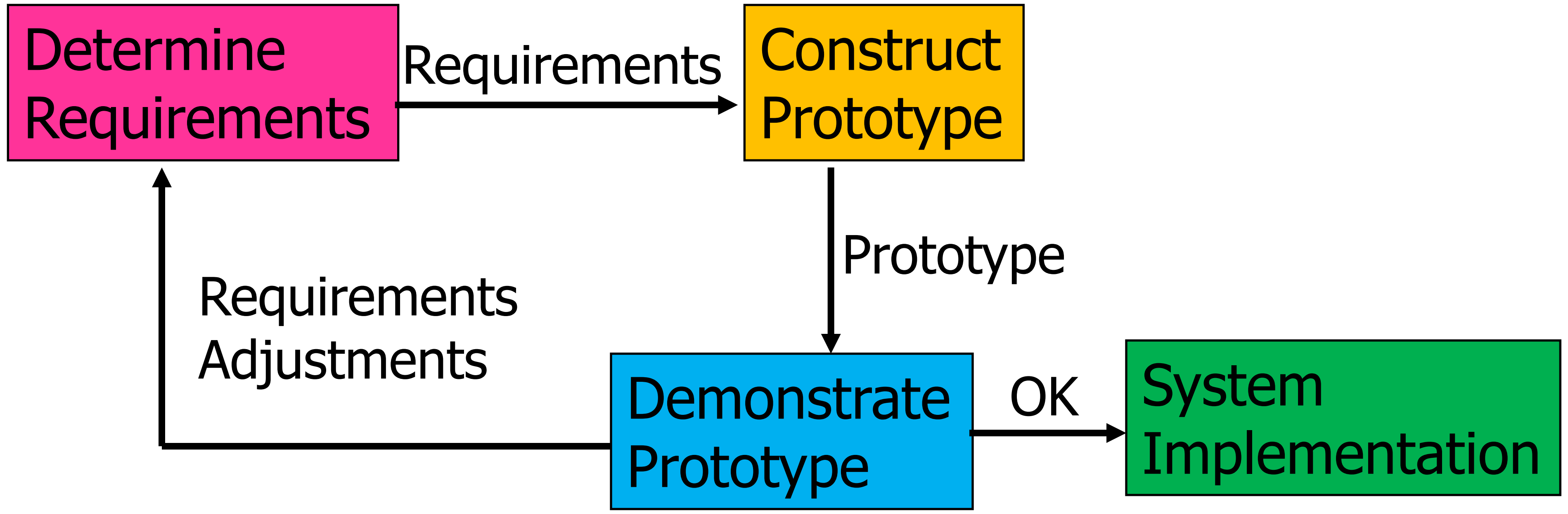
The V Model


Requirements Analysis
Acceptance Test Design
Acceptance Testing
user requirements document
System Analysis
System Test Design
System Testing
software specification document /SRS
Architecture Design
Integration Test Design
Integration Testing
Architecture design/ high-level design
Module Design
Unit Test Design
Unit Testing
low level design document / program specifications
Coding
Source code

Prototyping Model

- Specifications can be developed **incrementally**.
  - As users develop a better understanding of their problem, this can be reflected in the software system.
- Two types of prototyping:
  - Throwaway: implement only aspects poorly understood.
  - Evolutionary: implement aspects best understood.

Prototyping Model



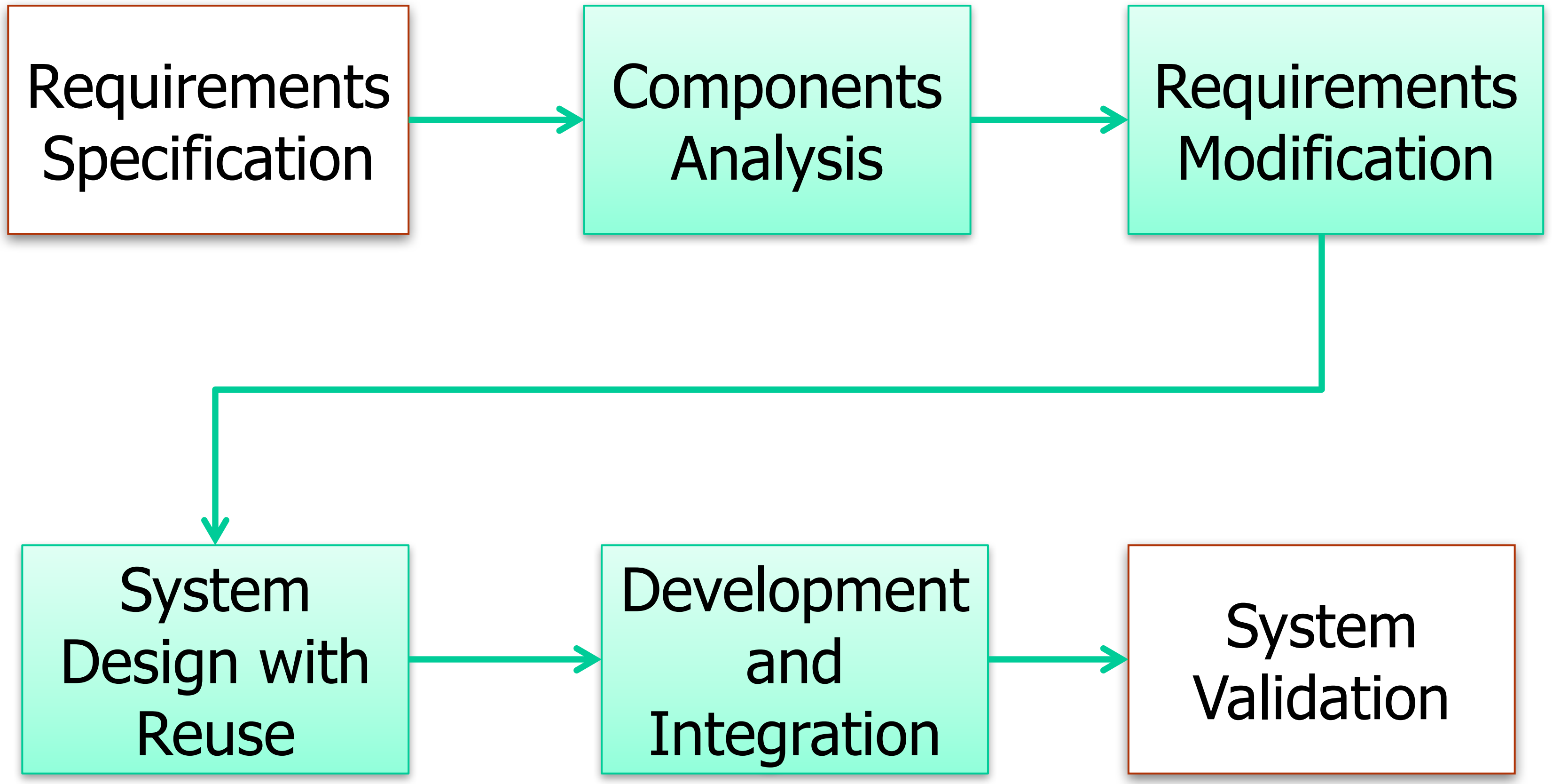
Pros & Cons

- Benefits of prototyping:
  - Improve communication
  - Reduce risk
  - Validate specification
  - For maintenance as well
- Problems of prototyping
  - Systems are often poorly structured

Component-based Software Process Model

- Based on systematic reuse where systems are integrated from existing components or **COTS (Commercial-off-the-shelf)** systems.
- **Off-the-shelf**: not designed or made to order, but taken from existing stock or supplies.
- **Can we treat open source as a kind of COTS?**
- Process stages
  - Component analysis
  - Requirements modification
  - System design with reuse
  - Development and integration.

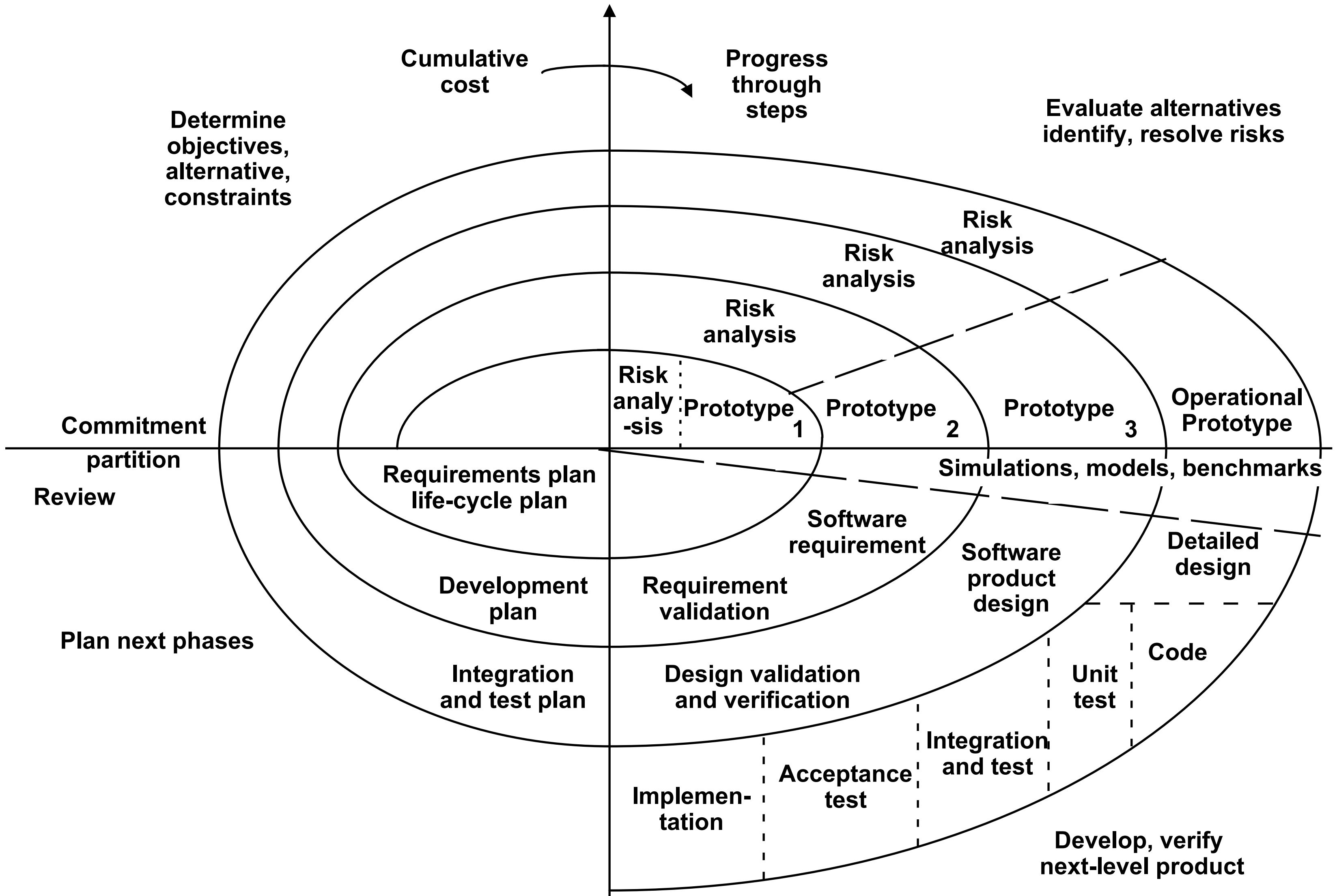
Component-based Software Process Model


Requirements Specification
Components Analysis
Requirements Modification
System Design with Reuse
Development and Integration
System Validation

Pros and Cons

- Advantages
  - Reducing the amount of software to be developed and so **reducing cost and risks**.
  - It usually also leads to **faster delivery** of the software at the initial stage.
- Problems
  - Requirements compromises are inevitable and this may lead to a system that does not meet the real needs of users.
  - At the expense of an increase of software components integration.

Spiral Model: Risk-Driven



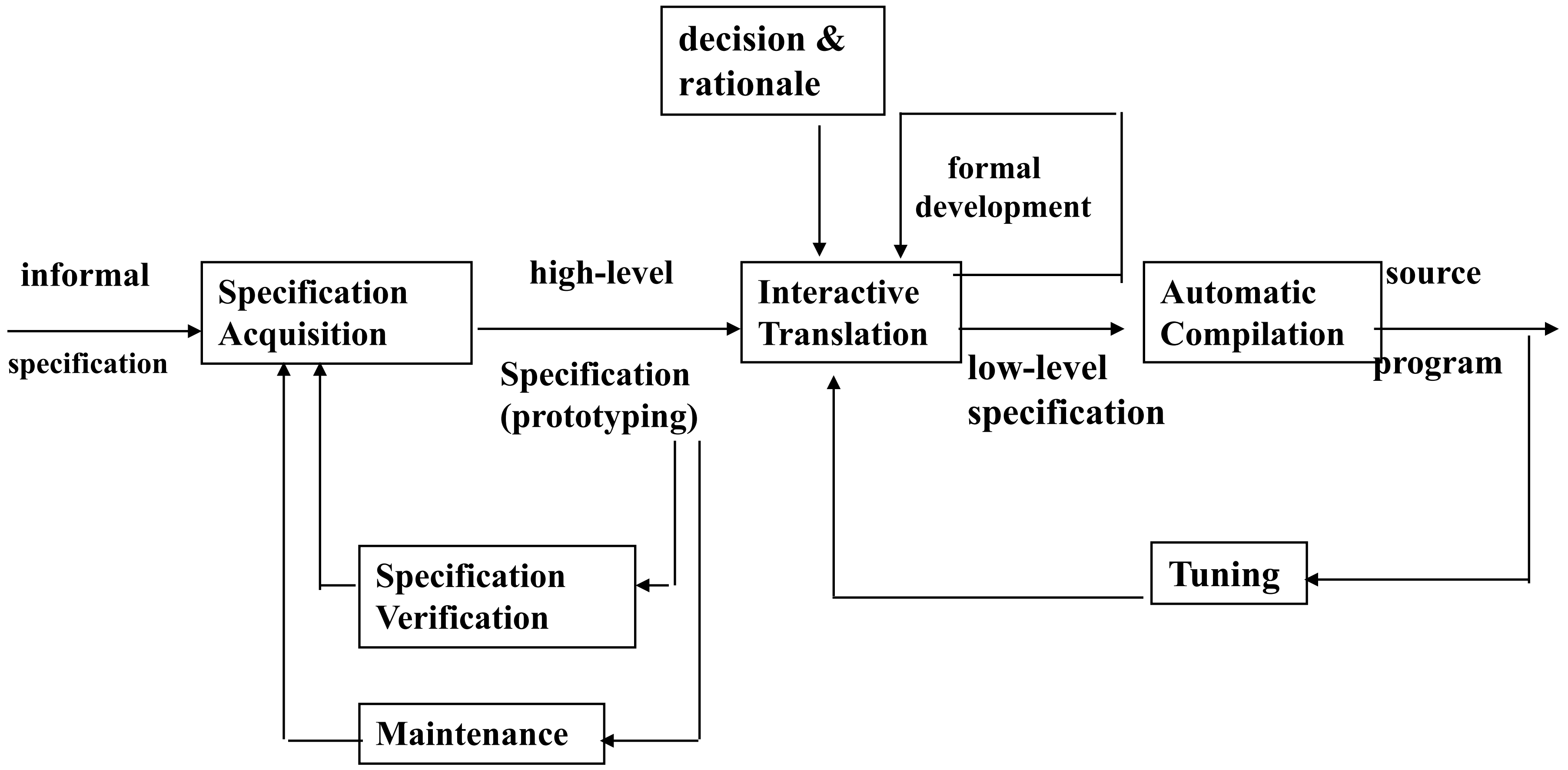
Risk Identification

- ❑ Technology risk: unproved technology, quality of the technical solution, performance issue, complex technology.
- ❑ Project management: project schedule, project scope, resource planning, cost estimation, budget.
- ❑ Organization: unstable organization, unrealistic project purpose, insufficient funding, insufficient talent pool.
- ❑ External factor: climate, labor force, natural disaster.

Pros & Cons

- Spiral Model is risk-driven rather than document-driven.
- Advantages
  - Introduce risk management
  - Prototyping controls costs
  - Evolutionary development
- Disadvantages
  - Very costly
  - Depends heavily on experts to correctly assess risks

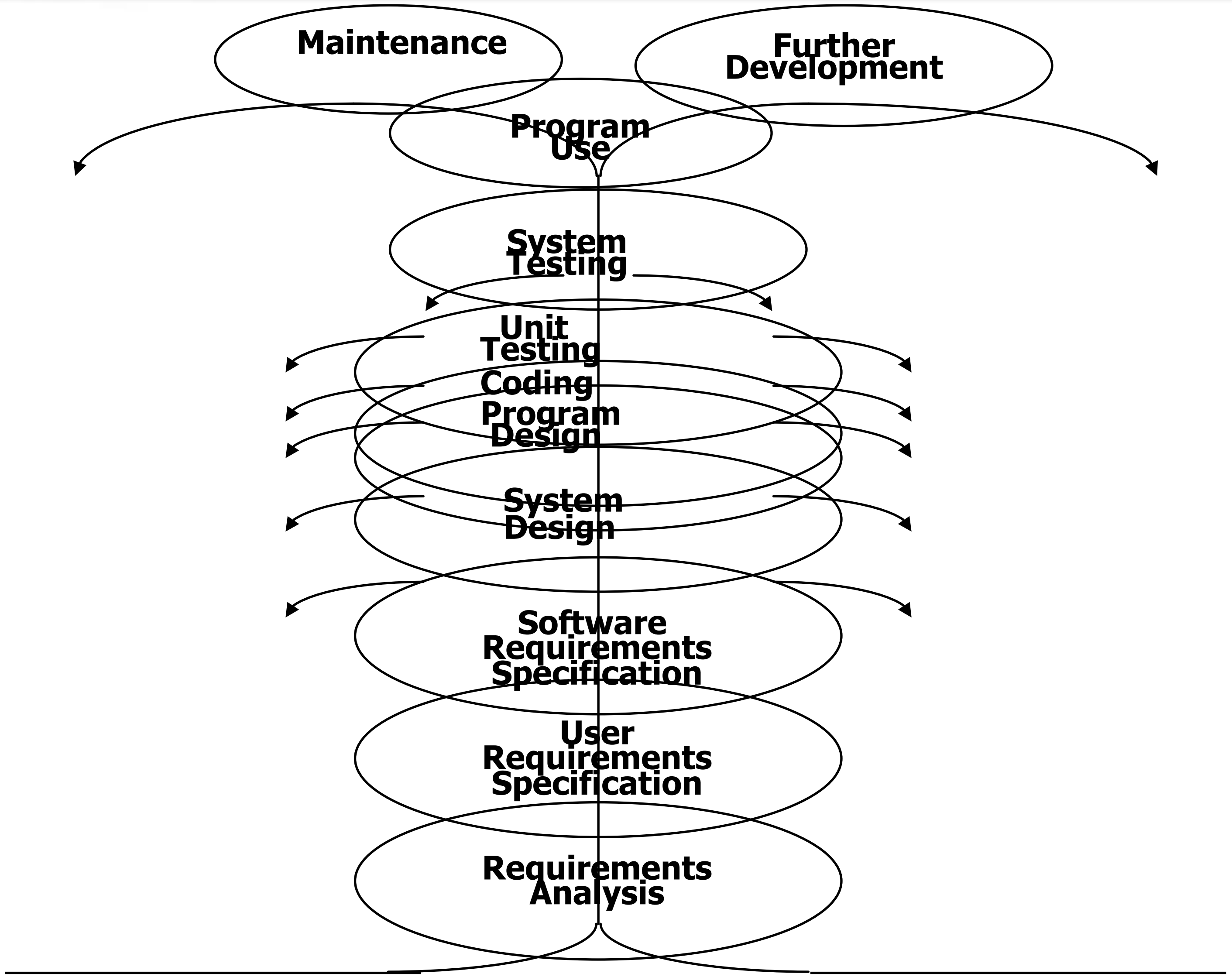
Automatic Synthesis Model



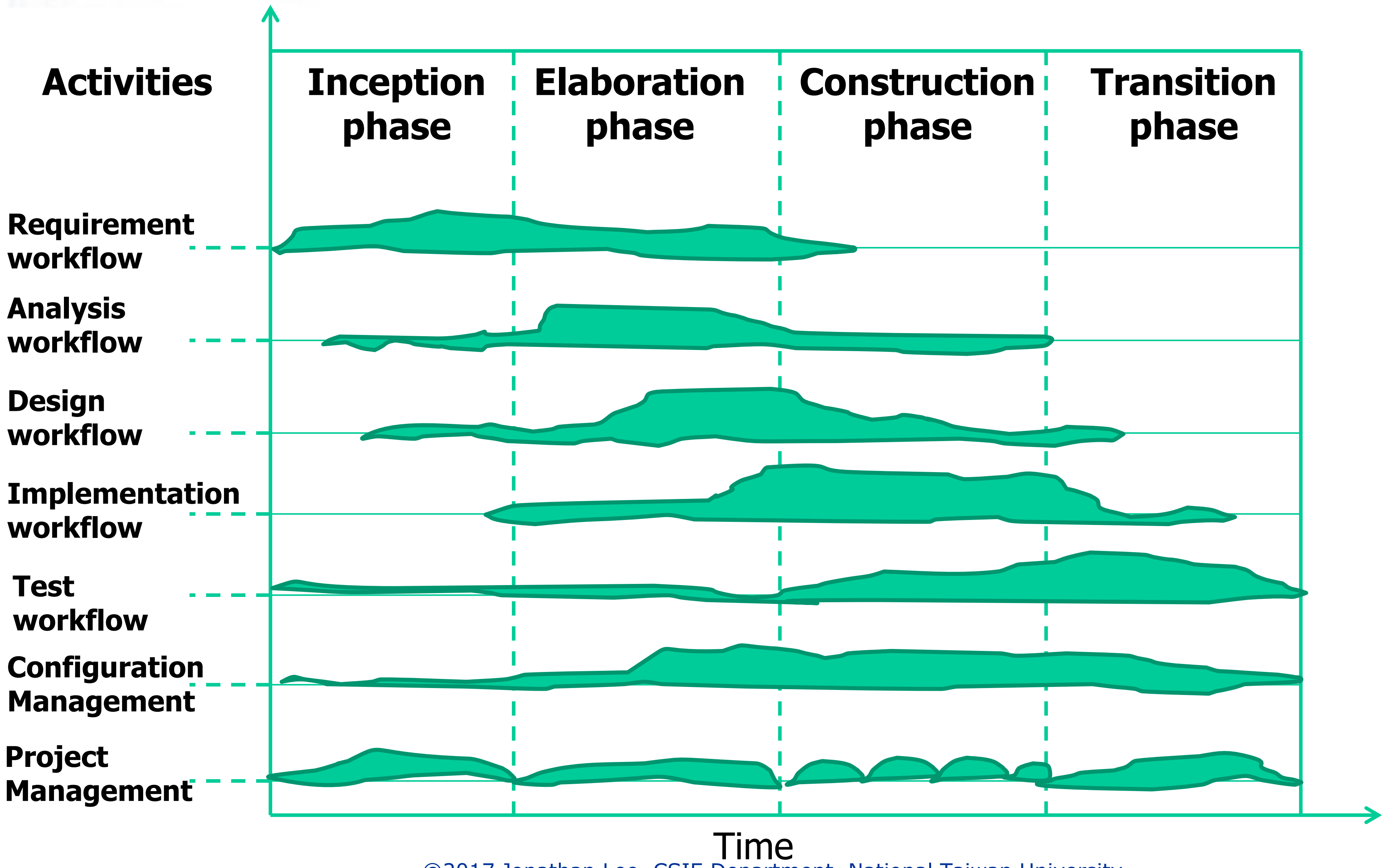
Object-Oriented Approach

- Object-Oriented Analysis (OOA) emphasizes on finding and describing the objects of a problem domain.
- Object-Oriented Design (OOD) emphasizes on defining logical software object that will ultimately be implemented in an objec-oriented programming language.
- Object-Oriented Programming (OOP) implements the designed components in an object-oriented programming language (e.g. Java)

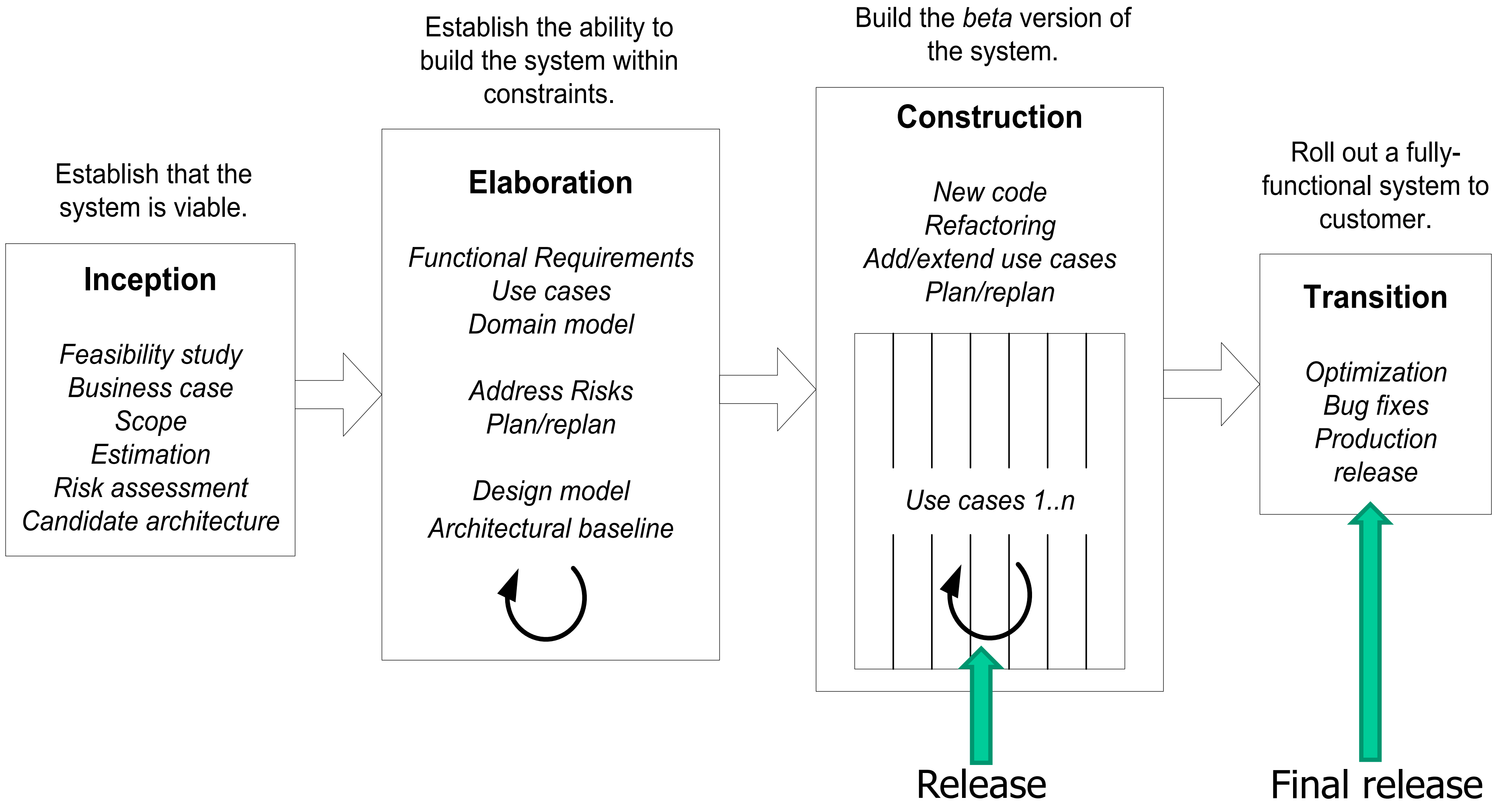
Object-Oriented Model



Unified Process



Unified Process Phases



Unified Process Phases

- Inception
  - Establish the **business case** for the system. A business case captures the reasoning for initiating a project.
  - Identify all external entities (people or systems)
- Elaboration
  - Develop an understanding of the problem domain and the system architecture (**architectural description**)
  - Develop the project plan and identify key project risks (**development plan**)
  - Requirements model for the system (**use cases model**)

Unified Process Phases

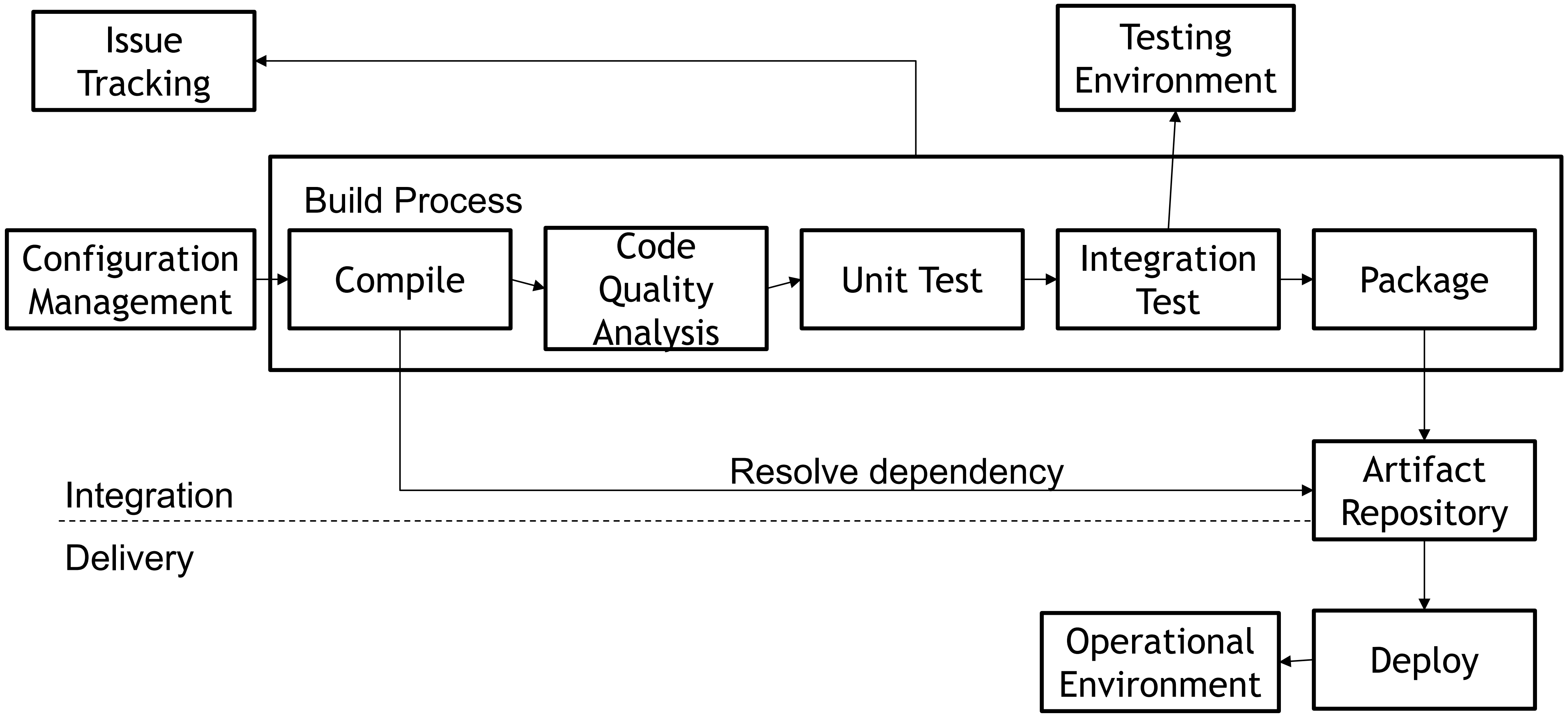
- Construction
  - System **design, programming and testing** (A working software system and associated documentation for delivery)

- Transition
  - **Deploy** the system in its **operating environment** (A documented software system that is working correctly in its operational environment)

Software Lifecycle

- Definition: What
- Development: How
- Maintenance: Change

DevOps Process



References

- Frederick P. Brooks, "No Silver Bullet: Essence and Accidents of Software Engineering," IEEE Computer, Vol. 20, No. 4 (April 1987) pp. 10-19.
- The NATO Software Engineering Conference, 1968.
- 李允中, 軟體工程,台灣軟體工程學會, 2013.