CPSC 3720: Lecture 7: Introduction to Design

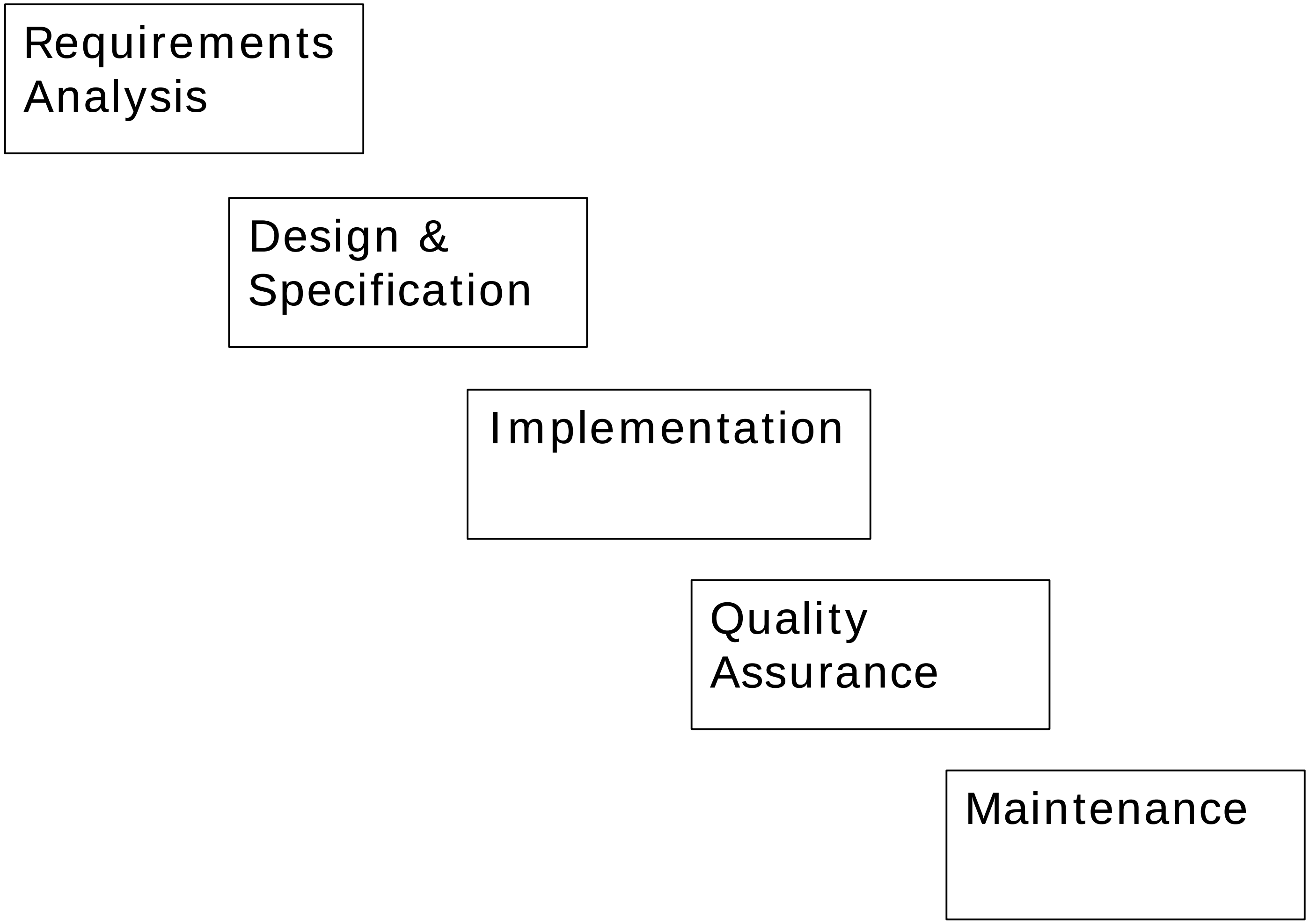
Eileen T. Kraemer and Murali Sitaraman, Clemson
E-mails: etkraem@clemson.edu and murali@Clemson.edu

Fall 2018 Version

Some of the lecture slides have benefitted from the findings of grants from the US National Science Foundation.

These slides used by Dr. Holly with permission of the authors

Software Lifecycle Phases



Discussion: Purpose of design

Design & Specification Intro

DESIGN & Specification

- Objectives: HOW?
- Inputs
- Outputs
- Approaches
- Verification

Implementation

Quality Ass

Maintenance

Design Intro

- Objectives: HOW should this system be built?
  - Key points: Modularization, reuse
- Inputs: Requirements Definition Document (RDD)
- Outputs: Design Documents
  - Design diagrams and specifications of modules
- Example Approaches:
  - Structured design
  - Object-oriented design
  - Component-based design

Design Principles

- Coupling
  - How inter-related are the modules in a system?
- Cohesion
  - How “singled-minded” is each module?

Coupling

- Coupling is communication
- Design goal: Minimize coupling, so there is less need for communication among modules
- All coupling cannot be avoided.
- Design goal: Where coupling is unavoidable:
  - make it a “desirable form” of coupling
  - Make the communication that is needed precise

Undesirable Forms of Coupling

- They are “non-modular”; Make it difficult to reason about software
- Global (or Common) coupling
  - E.g., Use global variables to share information among modules
- Content coupling
  - E.g., Implementation inheritance, whereby changes to the content of a class implementation affects that of another
- Control coupling
  - An external flag controls flow in a module

Desirable Forms of Coupling

- Parametric coupling
  - E.g., use of parameters to communicate information among modules
- Coupling strictly through interfaces
  - Reuse of components strictly based on their interface specifications
  - Specification inheritance, whereby existing specification is extended

Cohesion

- Cohesion is a module being coherent
- Design goal: Maximize cohesion in each module
- Design goal: Make the cohesion to be a “desirable form” of cohesion

Undesirable Forms of Cohesion

- Grouping of “unrelated” elements
- Coincidental cohesion
  - E.g., elements of a module come together by accident
- Logical cohesion
  - Sounds better than it actually is.
  - E.g., All outputs are grouped in a module
- Temporal cohesion
  - Elements that happen in close proximity in time are grouped together
  - E.g., “A start up” module

Desirable Forms of Cohesion

- Functional cohesion
  - Elements in a module perform related functionality
  - E.g., Interface specification and implementations that capture a well-designed “abstract data type” (e.g., stacks, queues, lists, or maps)