# SOFTWARE METHODOLOGY

Spring 2023

Lily Chang, Rutgers CS, New Brunswick

# Overview of Software Development

Lecture Note #1


# What is software?

- **Computer programs**

AND

- **Associated documentations**

Describe the structure of the software

User documentations

AND

- **Configuration data**

To make the programs operate correctly

# Essence of Software

- Abstract and intangible
- No physical limitations
  - Good or Bad?
- Complexity
- Changeability

What do you see?

# Arbitrary Complexity

- Human Factor – involves a group of people from different disciplines / application domains with different perspectives
- Distributed – comprises many interacting autonomous components
  - *Service-oriented cloud computing*
  - *Internet of Things (IoT)*
  - *Cyber Physical Systems (CPS)*


# Changeability

## Software must evolve to remain useful

- Discovered errors must be fixed
- Software is cheap and easy to change as opposed to hardware components
- The time between technological changes is often shorter than the duration of the project
- Requirements change!!!

# Software Methodology

- Methods for developing **quality software**
- Good practices for software development

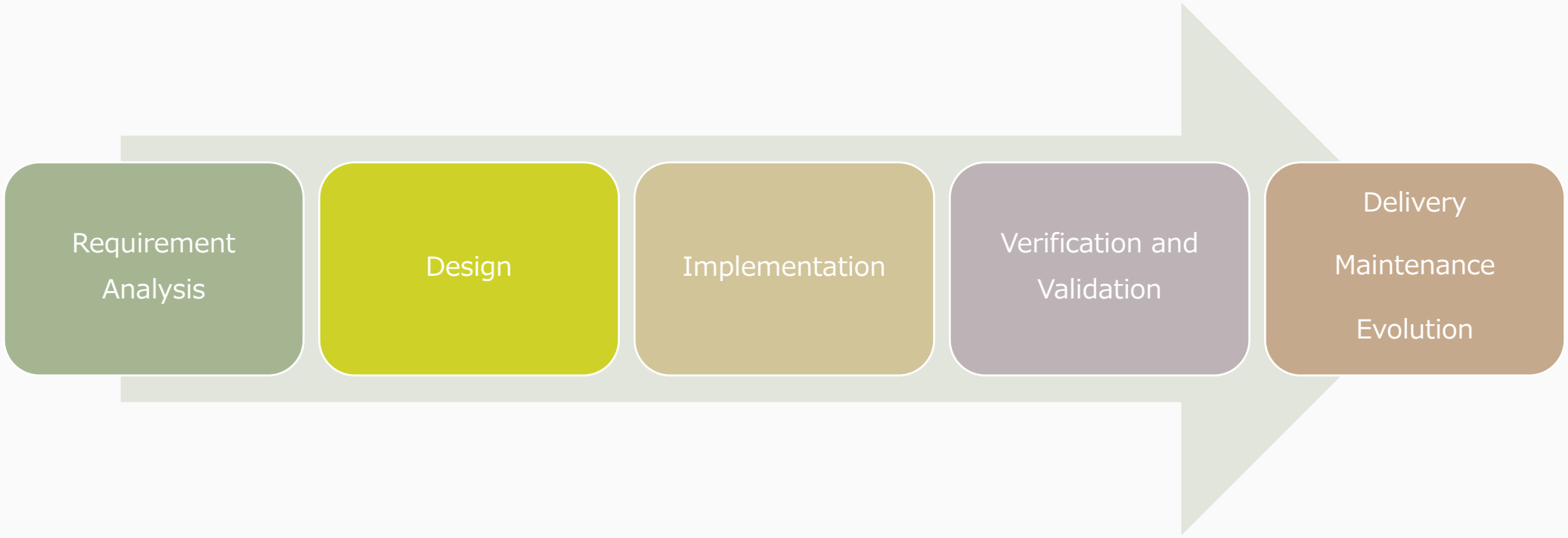

MAINTAINABILITY

DEPENDABILITY AND SECURITY

EFFICIENCY

USABILITY AND ACCEPTABILITY

# Essential Lifecycle Activities

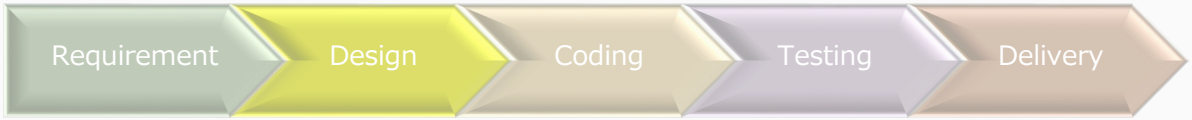


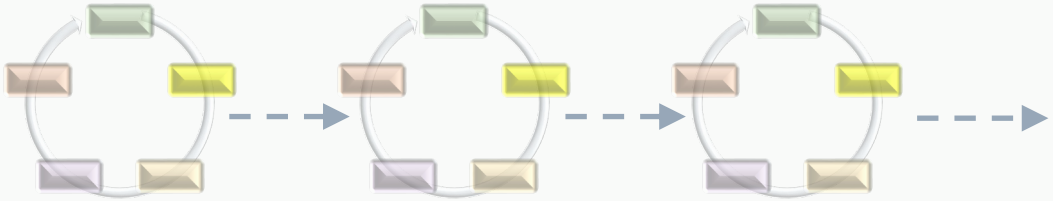
# Software Process Models

**A Software Process Model is an abstract representation of a software production process**

**Generic process models (paradigms)**

- Waterfall model
  - *Sequential process*
  - *Plan-driven*
- Incremental or evolutionary
  - *Iterative process*
  - *Feature-driven*
    - *Spiral Model*
    - *Agile*




# Good Practices for Software Development

❖ **Not just coding, but ”building” quality software**

❖ **Clean code**

- Clean code can be read and enhanced by a developer other than its original author
- It has well-designed tests
- It has meaningful names
- It provides one way rather than many ways for doing one thing
- It has minimal dependencies, which are explicitly defined, and provides a clear and minimal API
- The logic should be straightforward to make it hard for bugs to hide, the dependencies minimal to ease maintenance

# Coding Standard for Projects

- Class comments and method comments
- Descriptive names for classes, methods, constants and variables
- Code indentations and lineup (readability)
- No MAGIC numbers!

*A magic number is a numeric value that remains unchanged during program execution, and it is used directly in code without a name*

- Modularity

*Do only ONE THING in a method*

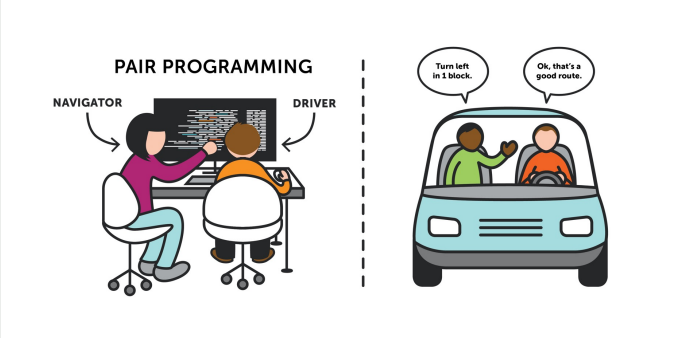
*A long method is an indication that the method is doing too much!*

# Good Practices for Software Development

**Agile manifesto**

- Adapt software changes better
- Frequent delivery (continuous integration and delivery)
- Involve customers in the development process
- Popular agile methods
  - XP (eXtreme Programming)
  - Scrum framework for software production management


https://www.agilealliance.org/agile101/subway-map-to-agile-practices/




# Pair Programming in eXtreme Programming (XP)

- Two developers shares a single workstation (one screen, keyboard and mouse among the pair)
- The developer at the keyboard is the “driver”, the other one actively involved in the programming task but focuses more on overall direction, is the “navigator”
- It is expected that the developers swap roles every few minutes or so.

# Pair Programming – Expected Benefits

Increased code quality: “programming out loud” leads to clearer articulation of the complexities and hidden details in coding tasks, reducing the risk of error or going down blind alleys

Better diffusion of knowledge among the team; in particular, when a developer unfamiliar with a component is pairing with one who knows it much better

Better transfer of skills, as junior developers pick up micro-techniques or broader skills from more experienced team members

Large reduction in coordination efforts, since there are N/2 pairs to coordinate instead of N individual developers

Improved resiliency of a pair to interruptions, compared to an individual developer: when one member of the pair must attend to an external prompt, the other can remains focused on the task and can assist in regaining focus afterward

# Pair Programming – Common Pitfalls

Both programmers must be actively engaging with the task throughout a paired session, otherwise no benefit can be expected

A simplistic but often raised objection is that pairing "doubles costs"; this is the worst-case outcome of poorly applied pairing

At least the driver, and possibly both programmers, are expected to keep up a running commentary; pair programming is also "programming out loud" – if the driver is silent, the navigator should intervene

Pair programming cannot be fruitfully forced upon people, especially if relationship issues, including the most mundane (such as personal hygiene), are getting in the way; solve these first!

# Object-Oriented Software Development

- Object-oriented analysis, design and programming are related but distinct
- OOA is concerned with developing an object model of the application/problem domain (What)
- OOD is concerned with developing an object-oriented system model to implement requirements (solutions domain) (How)
- OOP is concerned with realizing an OOD using an OO programming language such as Java or C++

# Java language

- **A popular language - https://www.tiobe.com/tiobe-index/**
- **The prerequisite is CS 112, Data Structures**
- **Java is a pure OO programming language**

Includes the features to facilitate the development of quality software

Design patterns with Java language

- **Many open-source Java library classes (APIs) available**

Software reuse reduces the efforts in software development

- **IDE tools – IntelliJ community edition** and **Android Studio**

Download install the free version to your computer, OR

Use the iLab machines: https://weblogin.cs.rutgers.edu/, use your NetID and CS password, must activate your CS account here: https://services.cs.rutgers.edu/accounts/

# Software Design

- **Architectural Design**

Decomposition at system-level

– *Software Architecture*

– *Presentation logic, business logic, data access logic, data storage*

Decomposition at component-level

– *Design patterns*

– *UML - Class Diagram*

- **Frontend – user interactions**

User Experience Design

User Interface Design

- **Backend – processes and data management**