

# CMPUT 401

# Software Process and Product Management

---

Ildar Akhmetov

[ildar@ualberta.ca](mailto:ildar@ualberta.ca)

Department of Computing Science  
University of Alberta

# Software Development Process

---

Fall 2020

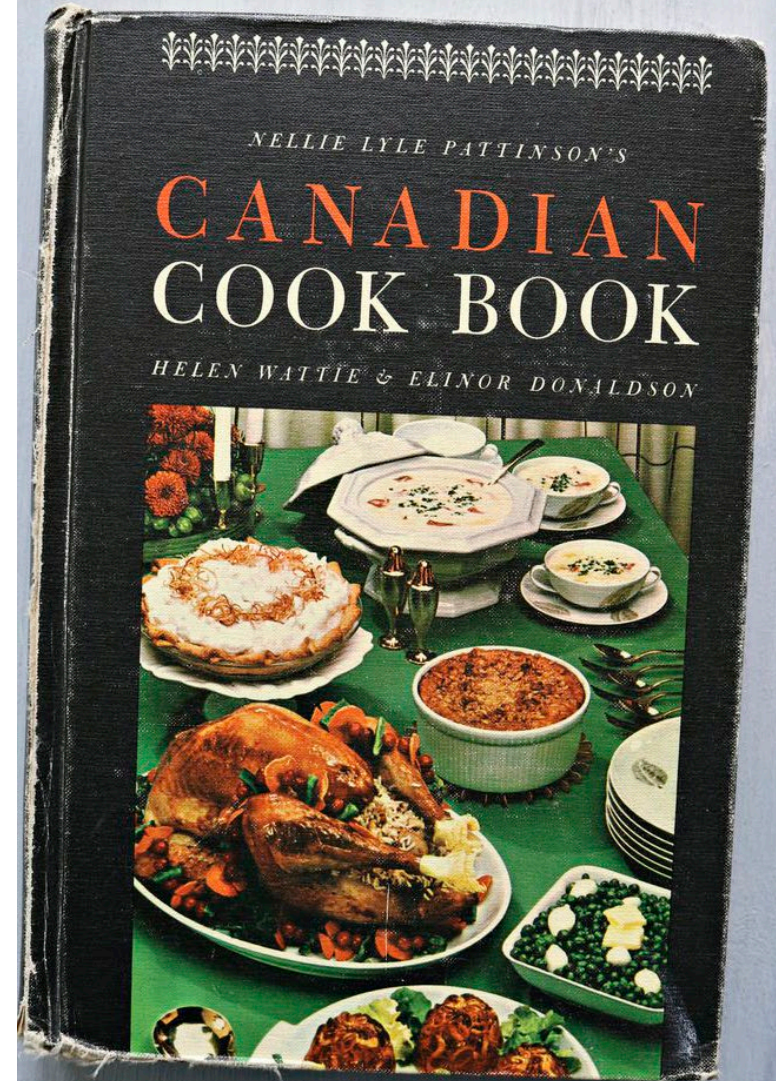
# What is a Process?

*A series of actions or operations conducing to an end.*



# Creating software...

- ...is more like authoring a cookbook than following a specific recipe
- There is no prescribed formula to writing a cookbook!
- Recipes must be collected, organized, and tested
- Recipe testing may lead to further refinements, experiments, and improvisations

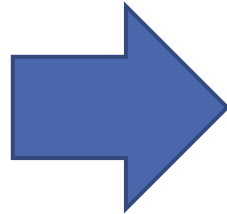


# Software Life Cycle

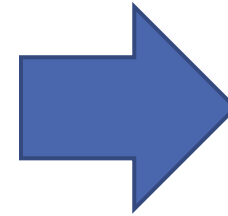
- A process that covers the entire spectrum of software development



Idea in the client's  
mind

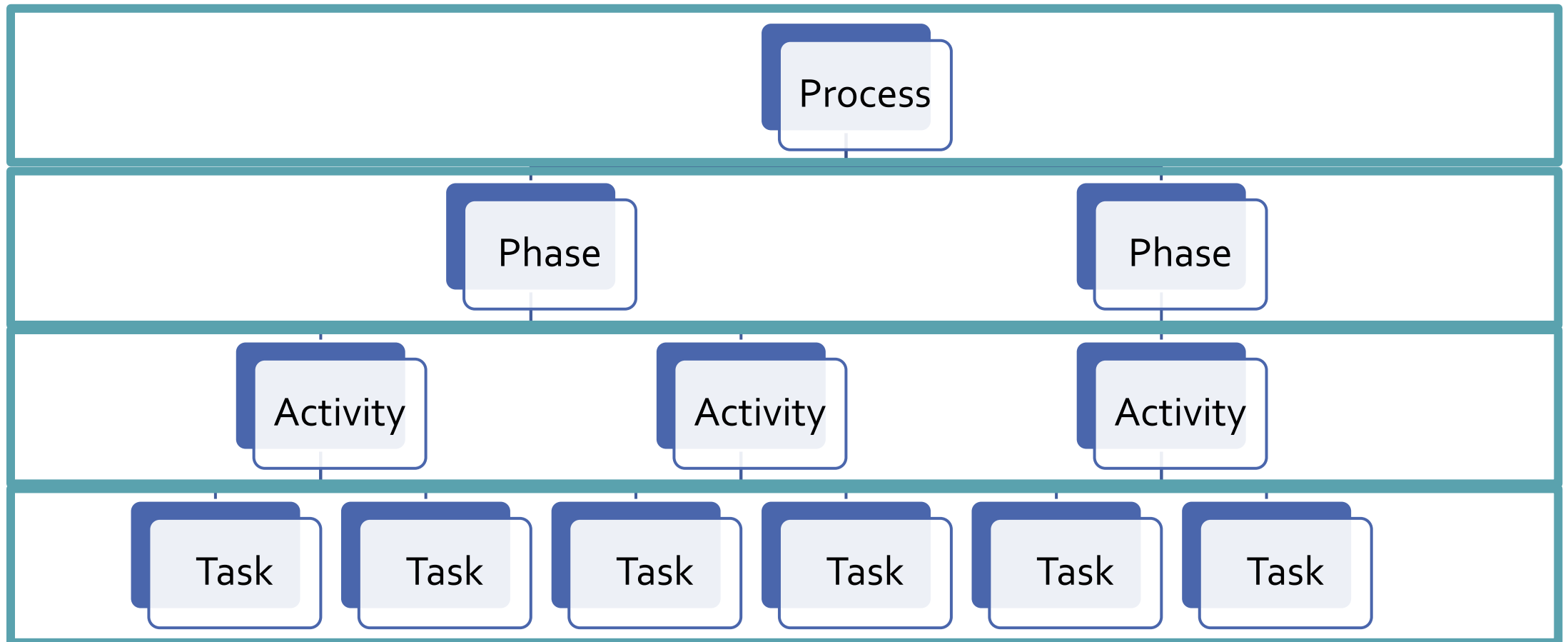


Working  
software



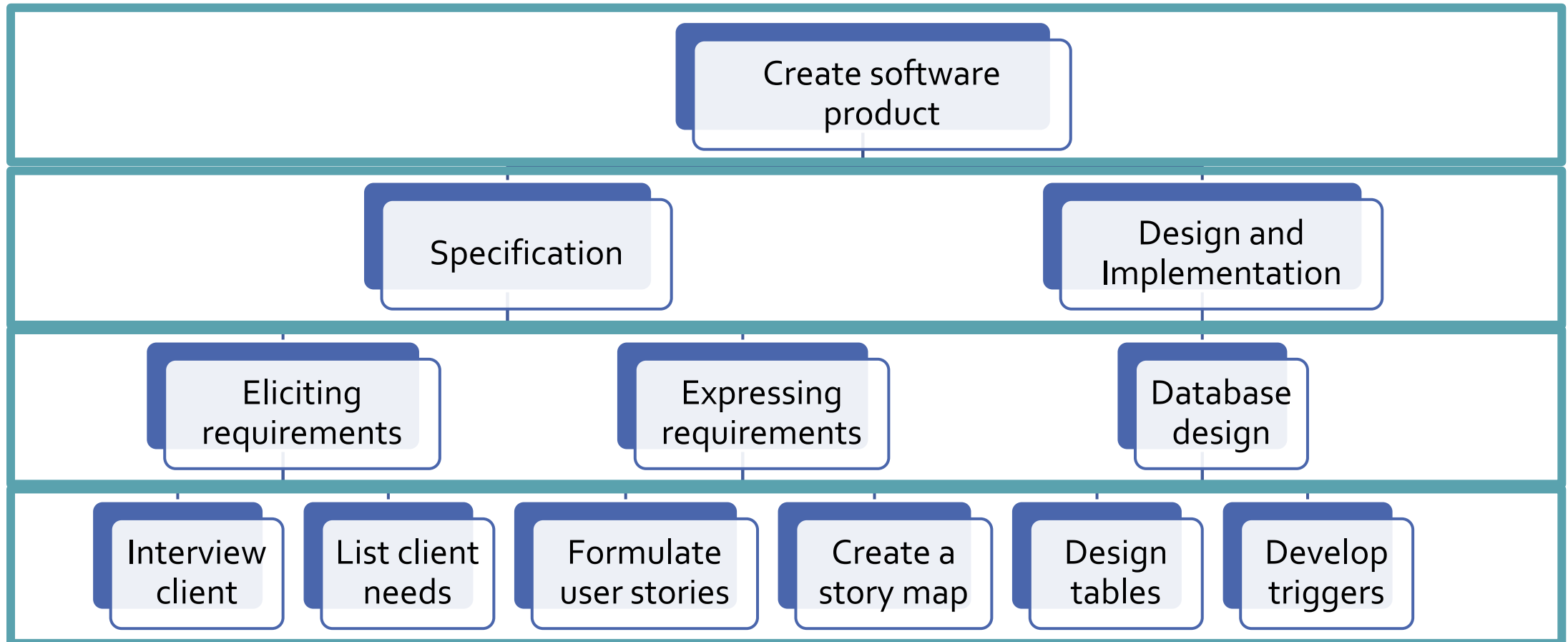
End of software  
project

Process → Phase → Activity → Task



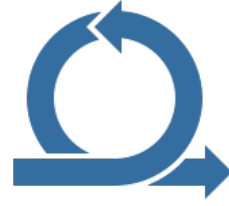
# Process → Phase → Activity → Task

*Example*

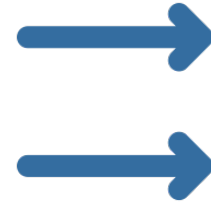




Linear  
process  
models



Iterative  
process  
models



Parallel  
process  
models

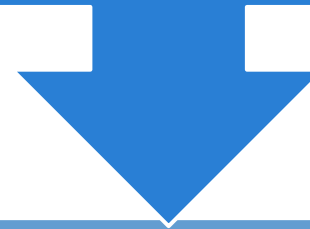
# Three Types of Process Models



*Task = small  
piece of work*

# Tasks

Task is the most basic unit of  
work that will be managed



## Examples

Write source  
code for a  
feature

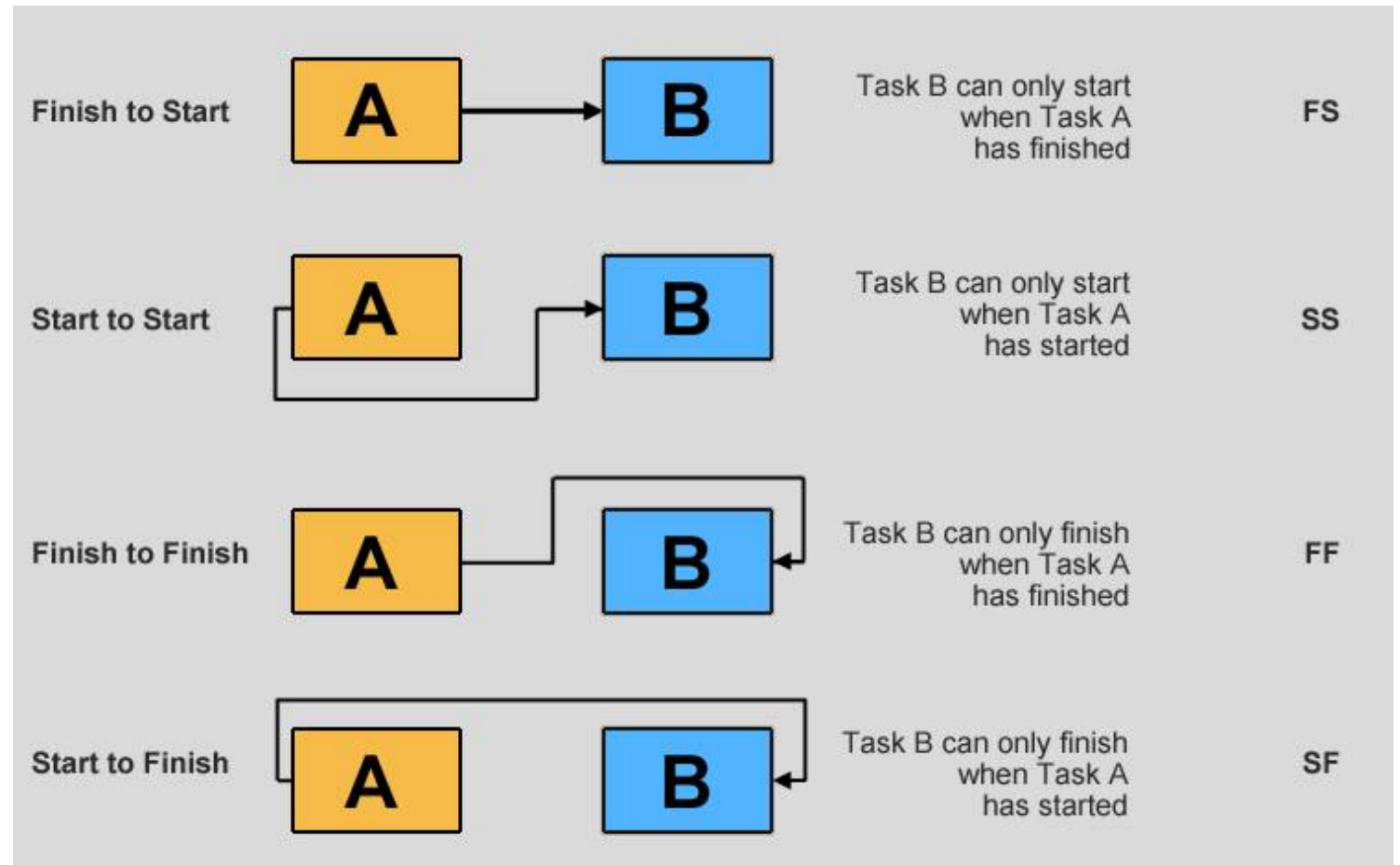
Design a  
feature

Write  
documentation

Install a  
library

Test a  
feature

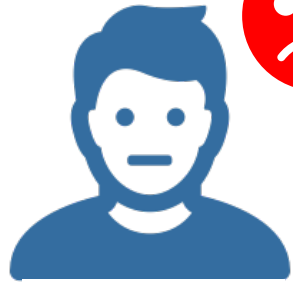
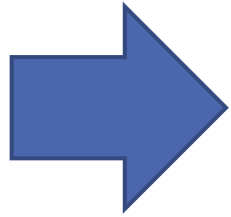
# Task Dependencies



# Roles



Task



Joe



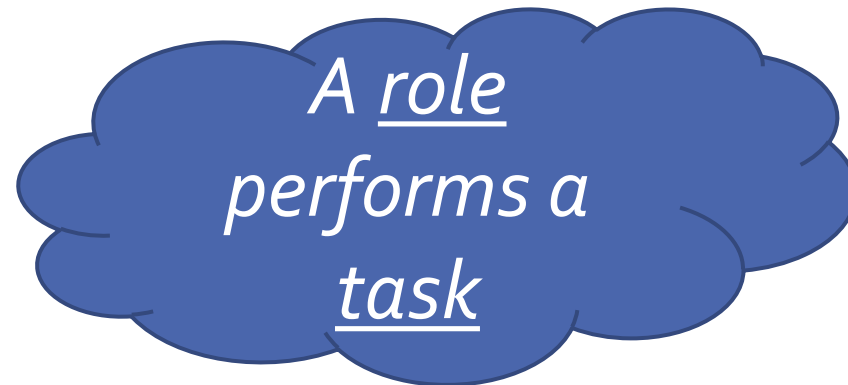
Junior Software  
Engineer



Programmer

# Roles

- Role is a job-related activity assigned to a person
- It is good practice to assign tasks to roles





Contains



Consumes



Performs

Produces

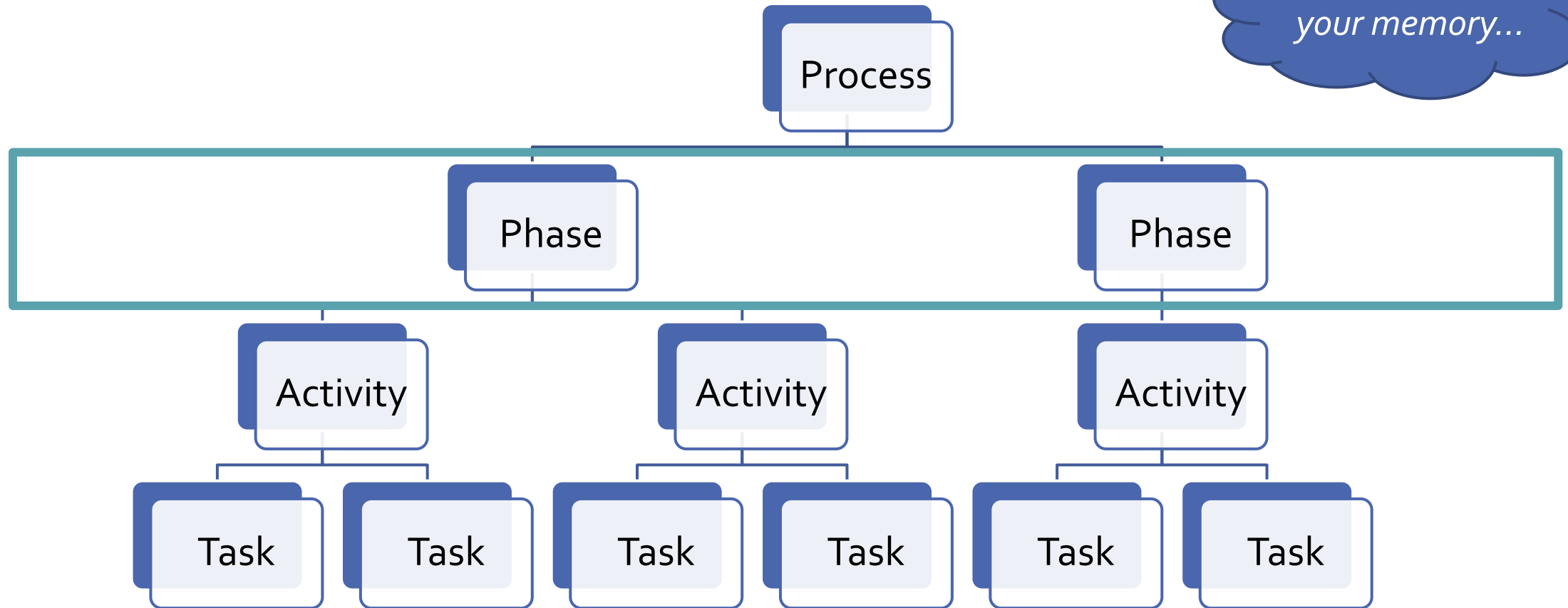
Uses



Product

# Process → Phase → Activity → Task

*Let's refresh  
your memory...*



Project Management Phase

Specification Phase

Design and Implementation Phase

Verification and Validation Phase

# Important Software Engineering Phases

# Specification Activities



- Identifying ideas or needs
- Eliciting requirements
- Expressing requirements
- Prioritizing requirements
- Analyzing requirements
- Managing requirements
- Formulating potential approaches



# Design and Implementation Activities

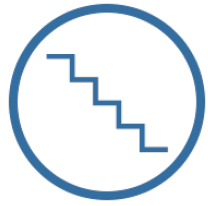
- Designing the architecture
- Designing the database
- Designing interfaces
- Creating executable code
- Integrating functionality
- Documenting

# Verification and Validation Activities

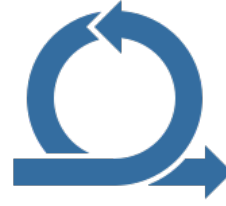
- Developing test procedures
- Creating tests
- Executing tests
- Reporting evaluation results
- Conducting reviews and audits
- Demonstrating to clients
- Conducting retrospectives

# Project Management Activities

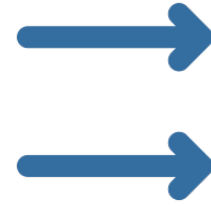
- Creating a process
- Software Processes and Agile Practices | 12
- Setting standards
- Managing risks
- Performing estimations
- Allocating resources
- Making measurements
- Improving the process



Linear  
process  
models



Iterative  
process  
models



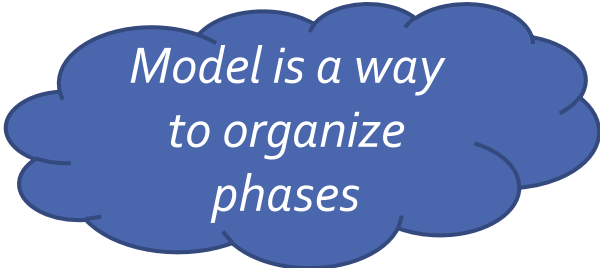
Parallel  
process  
models

*Model is a way  
to organize  
phases*

# Three Types of Process Models

# Linear Models

- Waterfall
- V-Model
- Sawtooth Model



*Model is a way  
to organize  
phases*

# Waterfall Model

*Documentation  
is critical:  
6 documents*

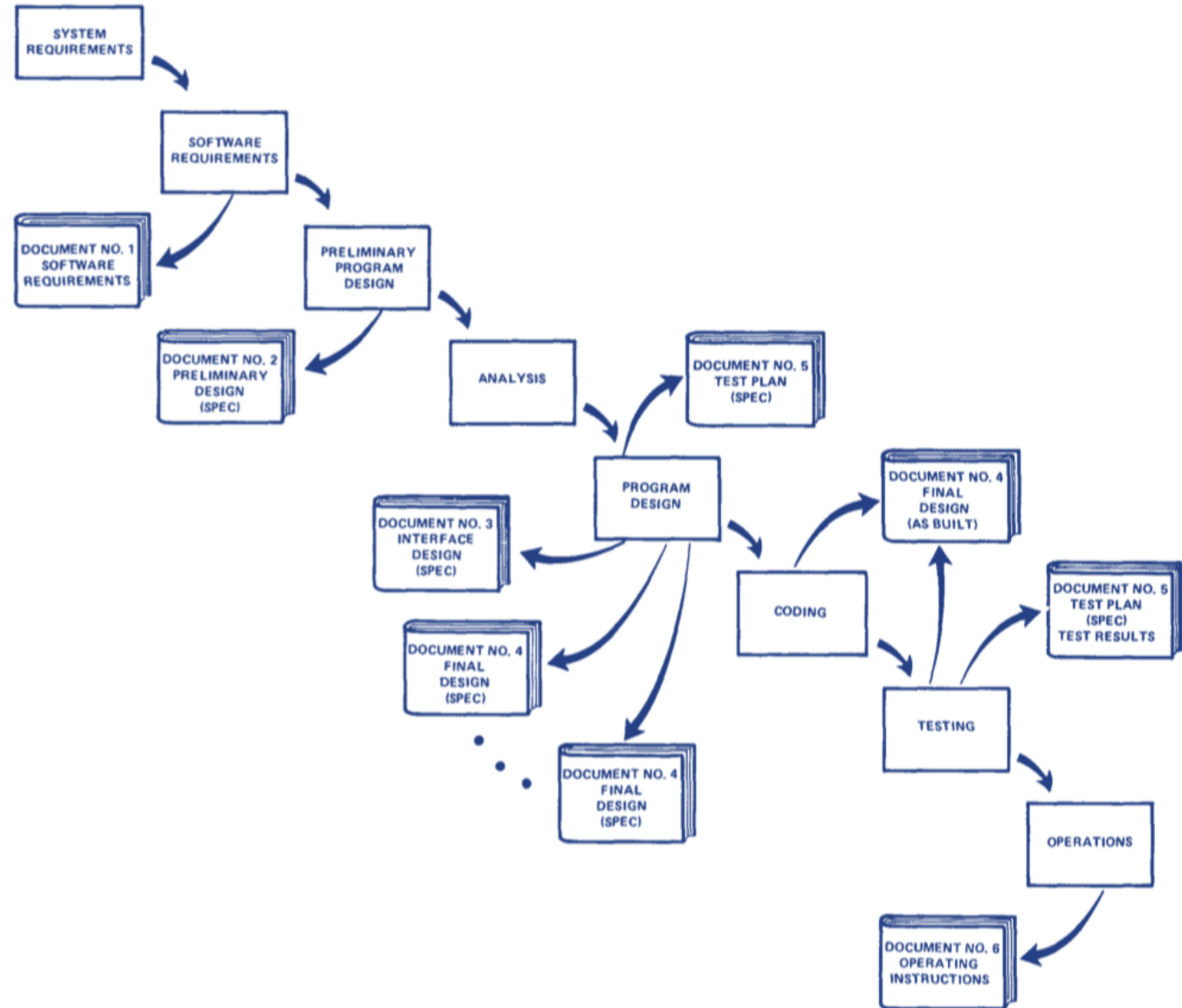


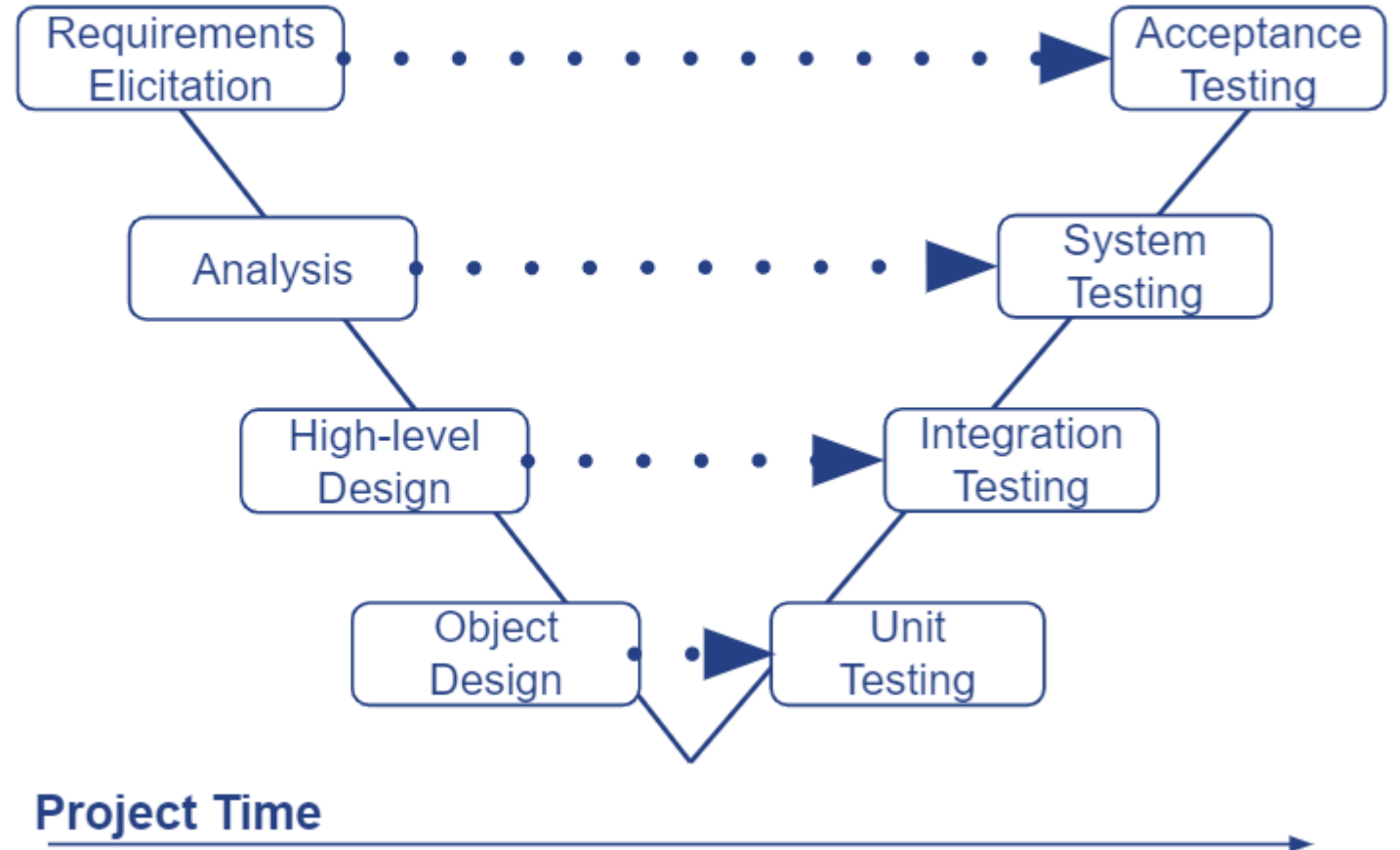
Figure 6, Step 2: Insure that documentation is current and complete — at least six uniquely different documents are required.

# V-Model

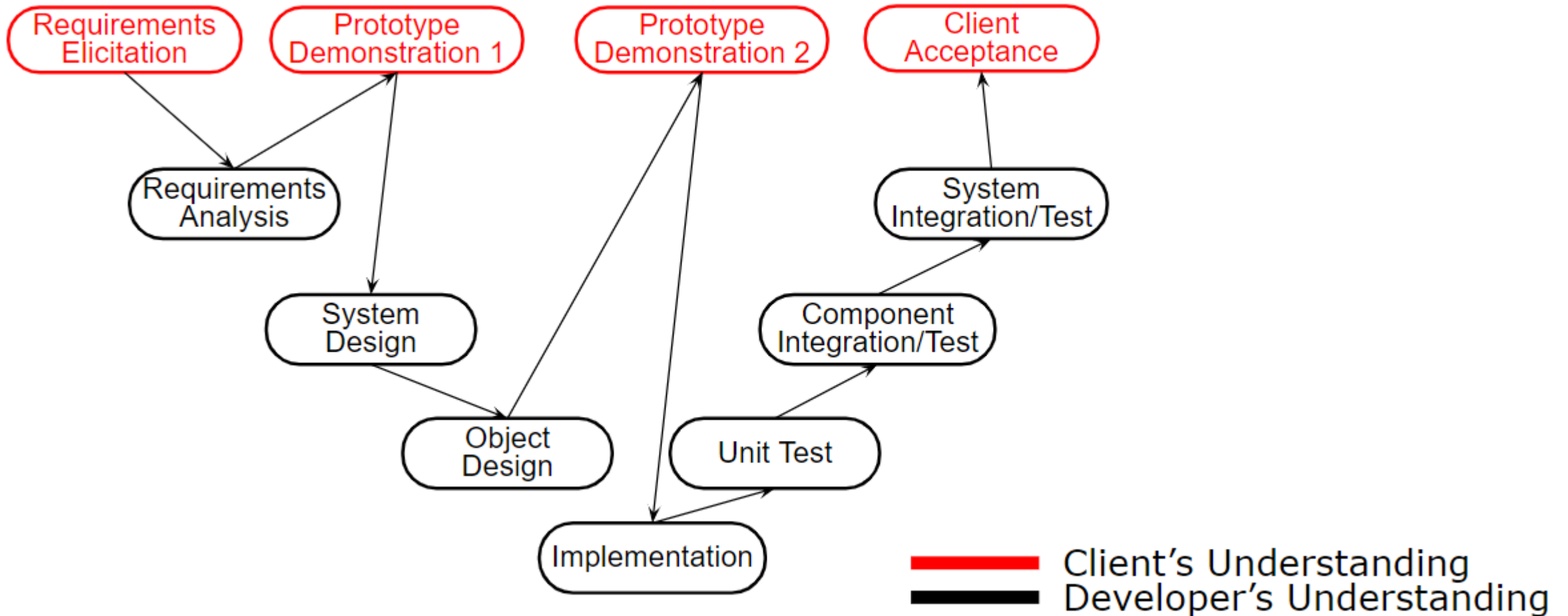
Level of Detail

Low

High

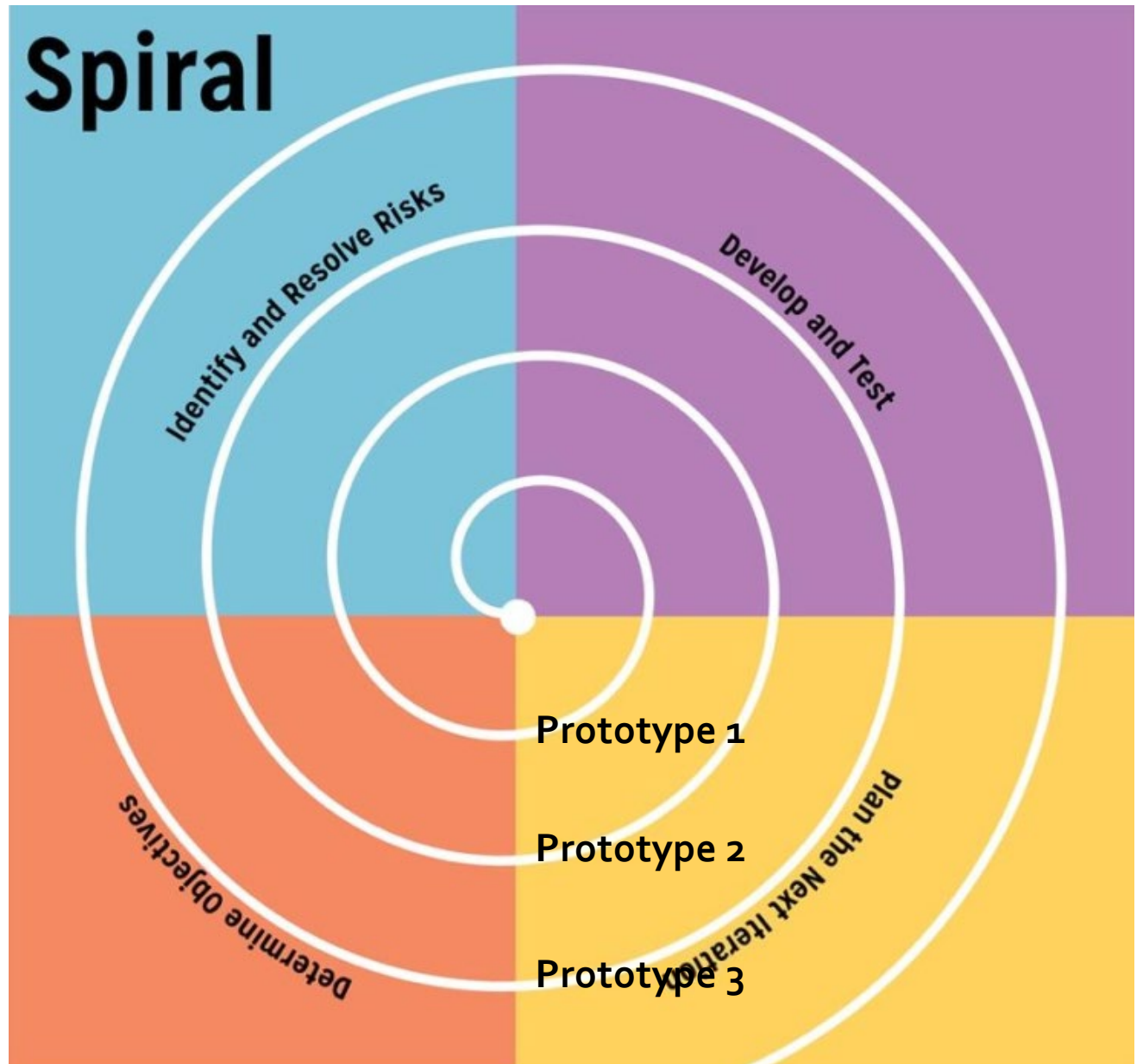


# Sawtooth Model

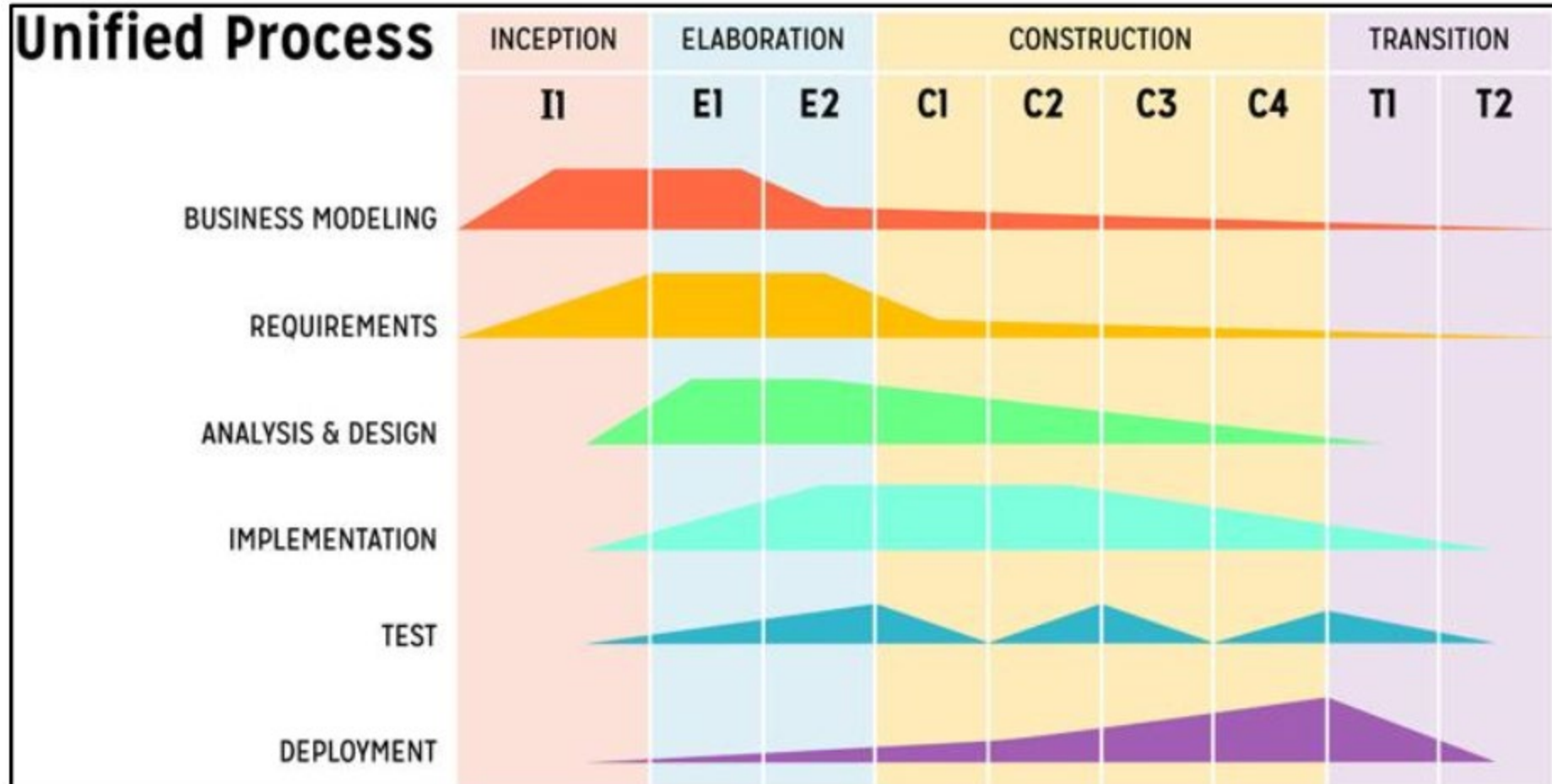




# Iterative Models: Spiral Model



# Parallel Models: Unified Process Model



# Unified Process Model

- Similar to the Spiral Model
  - series of cycles ending with a release
- Each cycle has four phases
  - **inception** - e.g., requirements, planning
  - **elaboration** - e.g., architecture, unit tests
  - **construction** - e.g., design, implementation, testing
  - **transition** - e.g., beta release, postmortem
- Each phase may involve one or more iterations

# Prototypes

<b>Illustrative</b>	User-interface wireframes, storyboards
<b>Exploratory</b>	Functional, working code (to explore feasibility and requirements)
<b>Throwaway</b>	To identify and learn from mistakes
<b>Incremental</b>	Deliver requirements in terms of their priorities
<b>Evolutionary</b>	Deliver all requirements in instalments

## Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Kent Beck	James Grenning	Robert C. Martin
Mike Beedle	Jim Highsmith	Steve Mellor
Arie van Bennekum	Andrew Hunt	Ken Schwaber
Alistair Cockburn	Ron Jeffries	Jeff Sutherland
Ward Cunningham	Jon Kern	Dave Thomas
Martin Fowler	Brian Marick	

© 2001, the above authors

This declaration may be freely copied in any form,  
but only in its entirety through this notice.

els

## Principles behind the Agile Manifesto

### *We follow these principles:*

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Business people and developers must work together daily throughout the project.

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Working software is the primary measure of progress.

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

Continuous attention to technical excellence and good design enhances agility.

Simplicity, the art of maximizing the amount of work not done, is essential.

The best architectures, requirements, and designs emerge from self-organizing teams.

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

# Agile Methods Principles

- **Individuals and Interactions**

- trust motivated individuals
- face-to-face conversation
- self-organizing teams work best
- team adjusts their behavior
- promote sustainable pace

- **Working Software**

- working software as the main measure of progress
- continuous, frequent delivery of value

- **Customer Collaboration**

- satisfy customer early
- customers and developers work together

- **Responding to Change**

- welcome changing requirements, even late
- technical excellence and good design
- simplicity—art of maximizing work not done

# XP Practices

- **Fine scale feedback**
  - Pair Programming
  - Planning Game
  - Test Driven Development
  - Whole Team
- **Continuous process**
  - Continuous Integration
  - Design Improvement
  - Small Releases
- **Shared understanding**
  - Coding Standards
  - Collective Code Ownership
  - Simple Design
  - System Metaphor
- **Programmer welfare**
  - Sustainable Pace

# Quiz

- Available on eClass
- Submit until the end of this week (Sunday 11:59 pm)