

WEB APPLICATION DEVELOPMENT

LECTURE 1

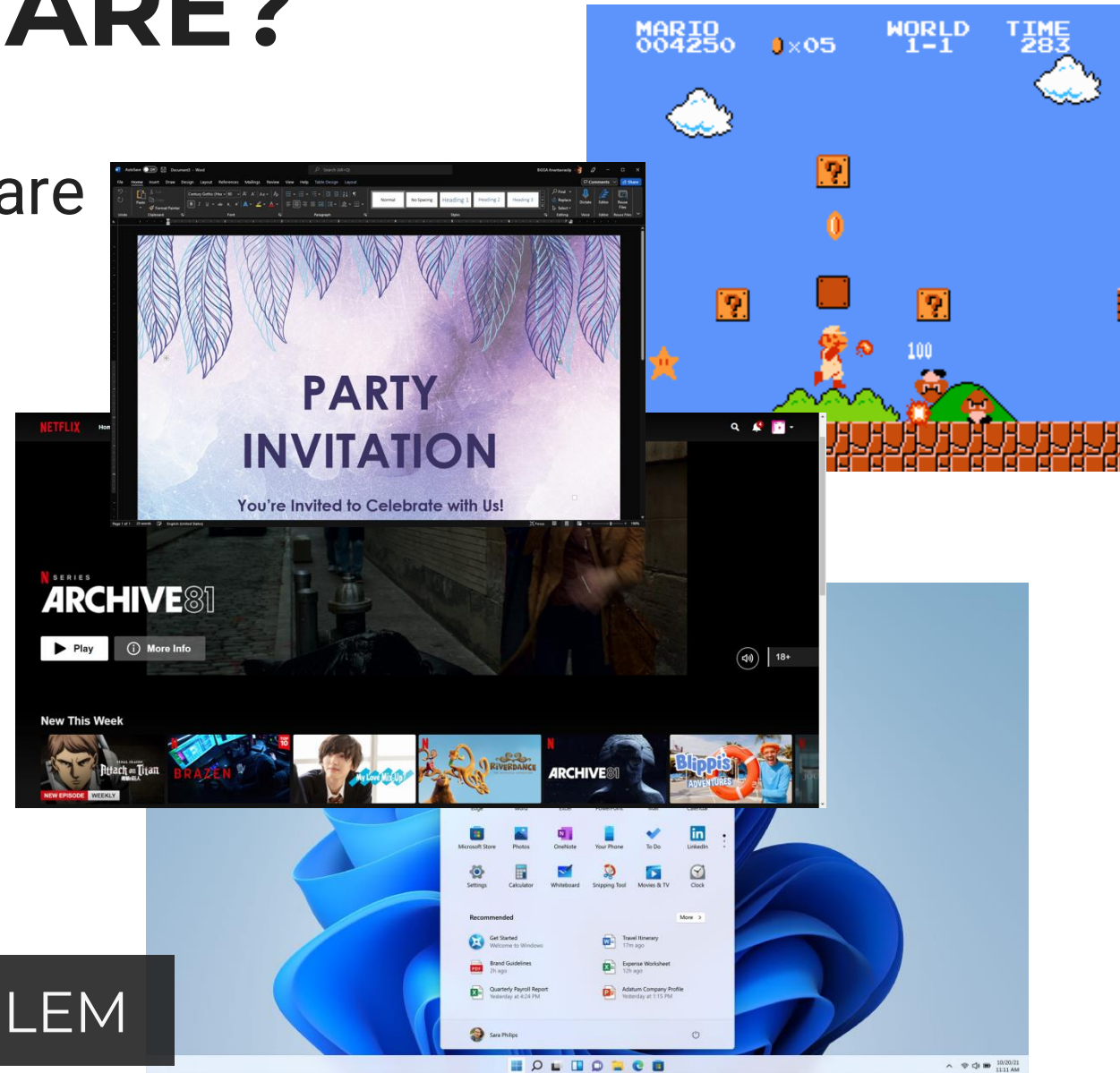
INTRODUCTION

AND A CRASH COURSE TO SOFTWARE
DEVELOPMENT PROCESS

Asst. Prof. Dr. ISARA ANANTAVRASILP

WHAT IS A SOFTWARE?

- Think about your favorite software
 - Applications / Utilities
 - Games
 - Mobile apps
 - Operating Systems
 - Web servers
- They are built for some specific purposes, i.e., to solve problems



SOFTWARE IS **SOLUTION** TO A PROBLEM

SOFTWARE DEVELOPMENT

- **It is not just coding!**
- It is problem solving
 - Understanding a problem
 - Proposing a solution and plan
 - Engineering a system based on the proposed solution using a **good** plan
- It is about dealing with complexity
 - Creating abstractions and models
 - Notations for abstractions
- It is knowledge management
 - Elicitation, analysis, design, validation of the system and the solution process
- It is tools making
 - Implement the solution according to the plan
 - Maintain and improve the tools

IN SHORT

***BUILD THE RIGHT THING
AND BUILD THE THING RIGHT.***

BUILD THE **RIGHT THING**

- Your **users** (not just the customers) get what they want
 - **Users**: The ones that use the product
 - **Customers**: The ones that pays for the product
- That is, your software is **solving their problems**
- How can we even know what is right?
 - Usually, they all come down to understand your users
 - Who are they?
 - What do they want?
 - Why do they want what they want?

BUILD THE **THING** RIGHT

- How can we deliver the product **right**?
- How can we do it in time and in budget?
- How to determine what is right?
 - Get to know your user and what they want
- Communicate what you know/learned to other developers (and users)
 - We will get to know how to do so as well

SOFTWARE AS A **SOLUTION**

Requirement
Analysis

What is the problem?

System Design

What is the solution?

Application Domain

Detailed Design

What are the best mechanism
to implement the solution?

Implementation

How is the solution
constructed?

Testing

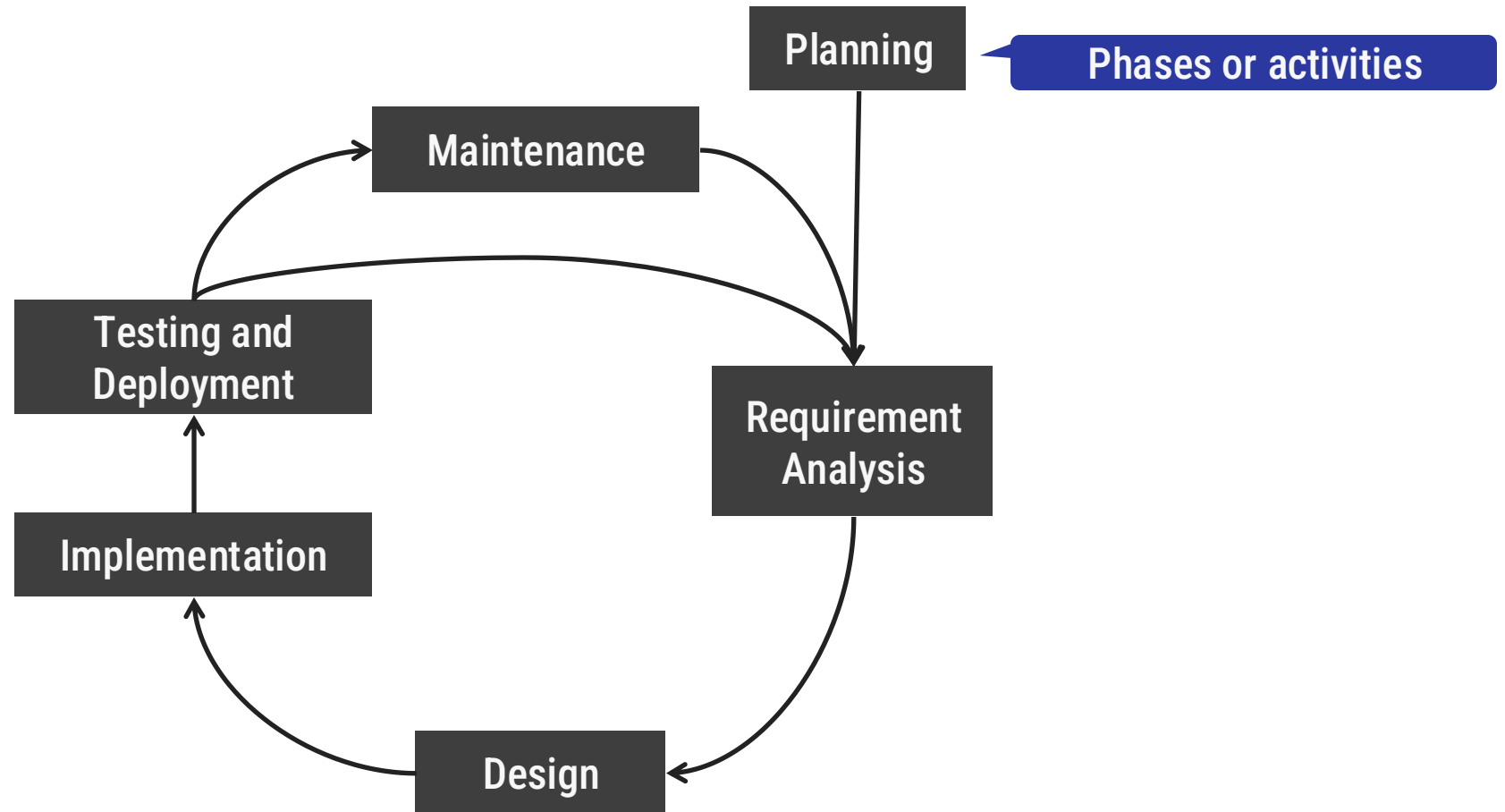
Is the problem solved?
Can the customer use the solution?

Maintenance

Are enhancements needed?

Solution Domain

SOFTWARE DEVELOPMENT **PROCESS**



AKA: **Software Development Life Cycle (SDLC)**

SOFTWARE DEVELOPMENT **PROCESS**

- **Planning**: Define initial idea/concept of the software product and determine rough schedule, resources and costs.
- **Requirement Analysis**: Specify what the application must do; answers “**what?**”
- **Design**: Specify the components (subsystems) and how they fit; answers “**how?**”
- **Implementation**: Write the code
- **Testing**: Execute the application with test data
- **Maintenance**: Repair defects and add capability
- Virtually all software involve these phases
- The question is how to execute those phases effectively

PLANNING

- **Inception**: Formulate the product idea
 - “What are we going to do?”
 - Very high-level
 - E.g., chatting app, photography
- **Project planning**: After the high-level idea is conceived, a work plan is developed
 - Identify high-level activities, work items, schedule, available resources and cost
 - “What do we have to do and what do we have?”
 - **Result**: a Software Project Management Plan (SPMP)

REQUIREMENT ANALYSIS

- Obtain detailed product information
 - Customer's wants and needs
 - The problems that the software is intended to solve
- Specific product features and functionalities and also performance, reliability and usability are determined
- “**What**” the software is supposed to do
- **Result:** Software Requirement Specification (SRS) or Requirement Analysis Document (RAD)

SYSTEM DESIGN

- Determine “**how**” to construct the software
- Categorized into two levels:
 - Architecture design
 - Overall, high-level design
 - How the software are divided into subsystems
 - How the subsystems relate to each other
 - Detailed design
 - How each subsystem works
 - How do they communicate with each other
 - Specific algorithms, data structure, interfaces, etc.
 - User interface and database design
- **Result:** Software Design Document (SDD)

IMPLEMENTATION

- **Coding:** Translate the software design to a programming language
 - Subsystem implementation
 - Subsystems integration
- **Result:** Source code and the object code that is ready to be tested

TESTING

- Test the implemented code for correctness
- Testing can be divided into three levels:
 - **Unit test**: Conducted by developers
 - **Integration test**: Subsystems are integrated and tested together to see if they interface properly
 - **System test**: All subsystems are integrated and the entire system is tested to ensure that it meets the user requirements
- System testing typically follows by **beta testing** and **acceptance testing**
- Acceptance testing is conducted by the customer on the **final release** of the software

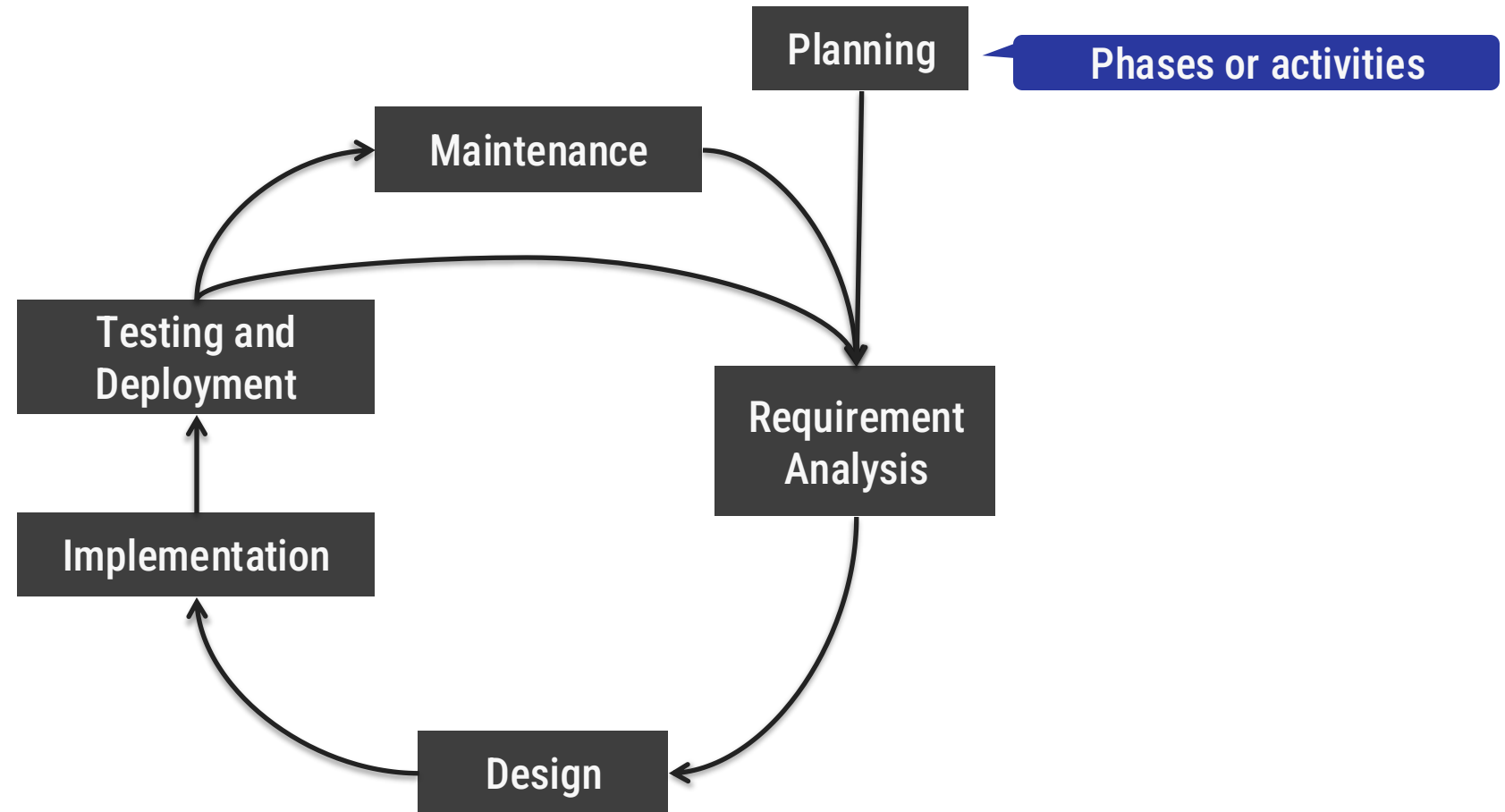
TESTING

- Testing is carried out to ensure that the software product does what it is intended to
- The more closely software product meets its requirements, and the requirements meet customer needs, the higher the **software quality**
- A deviation from what the software is required to do is called a **defect**

MAINTENANCE

- Maintenance phase takes place after the final release
- Maintenance involves:
 - Repair software defects
 - Additional features and functionalities
 - Improve attributes of the system such as performance or reliability

TYPICAL SDP

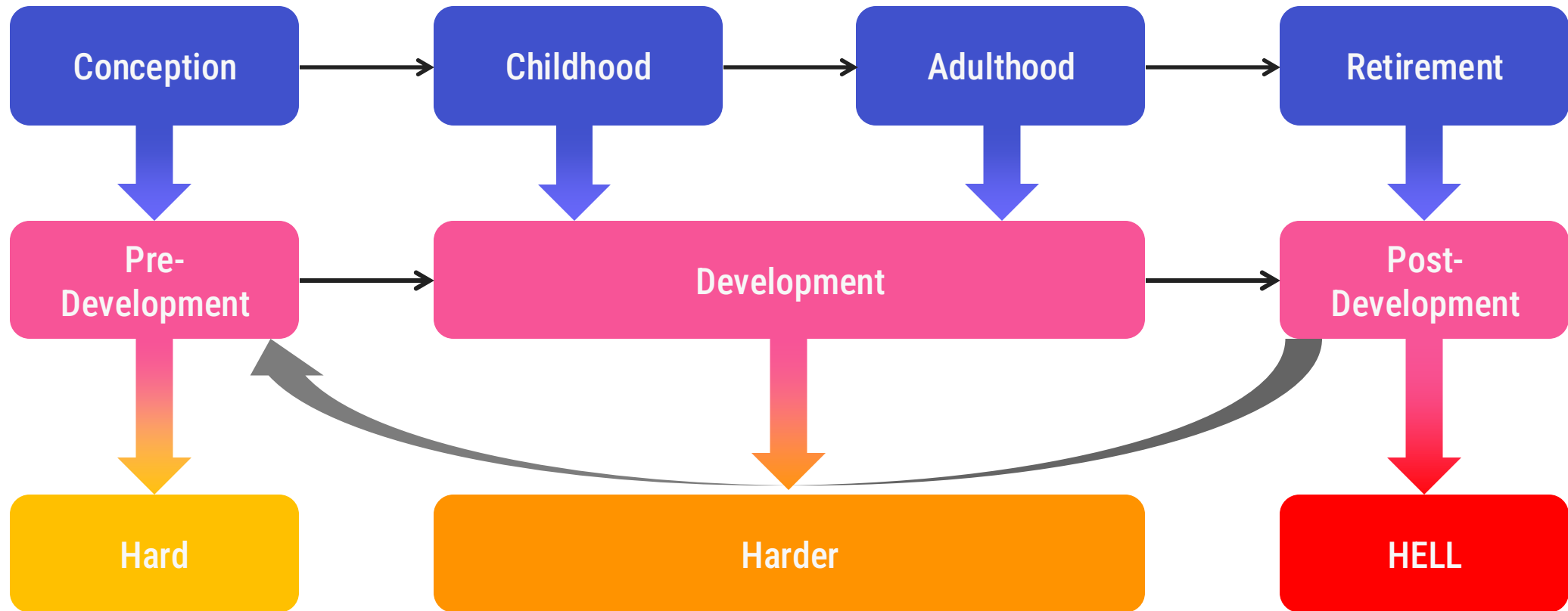


Question: How should we schedule these phases?

SOFTWARE DEVELOPMENT PROCESS

- **Software Development Process**
 - Aka Software Process or **Software Development Life Cycle (SDLC)**
 - Process of defining the order and frequency of software development phases (activities)
- **Software Development Process Model**: An abstract representation of a process
 - It describes development process
 - The description could be specific steps or flexible framework that can be tailored to different projects

SOFTWARE LIFE CYCLE



SOFTWARE DEVELOPMENT PROCESS

- Currently, several software development process models are proposed
- Example: Waterfall, spiral, agile process, etc.
- Each of these processes are designed for different purposes and project attributes (e.g., project size, time, domain knowledge)
- There might be no “right” method
- Before we discuss further in software development process, let's see who/what are involved with a software development

FOUR P'S IN SOFTWARE DEVELOPMENT

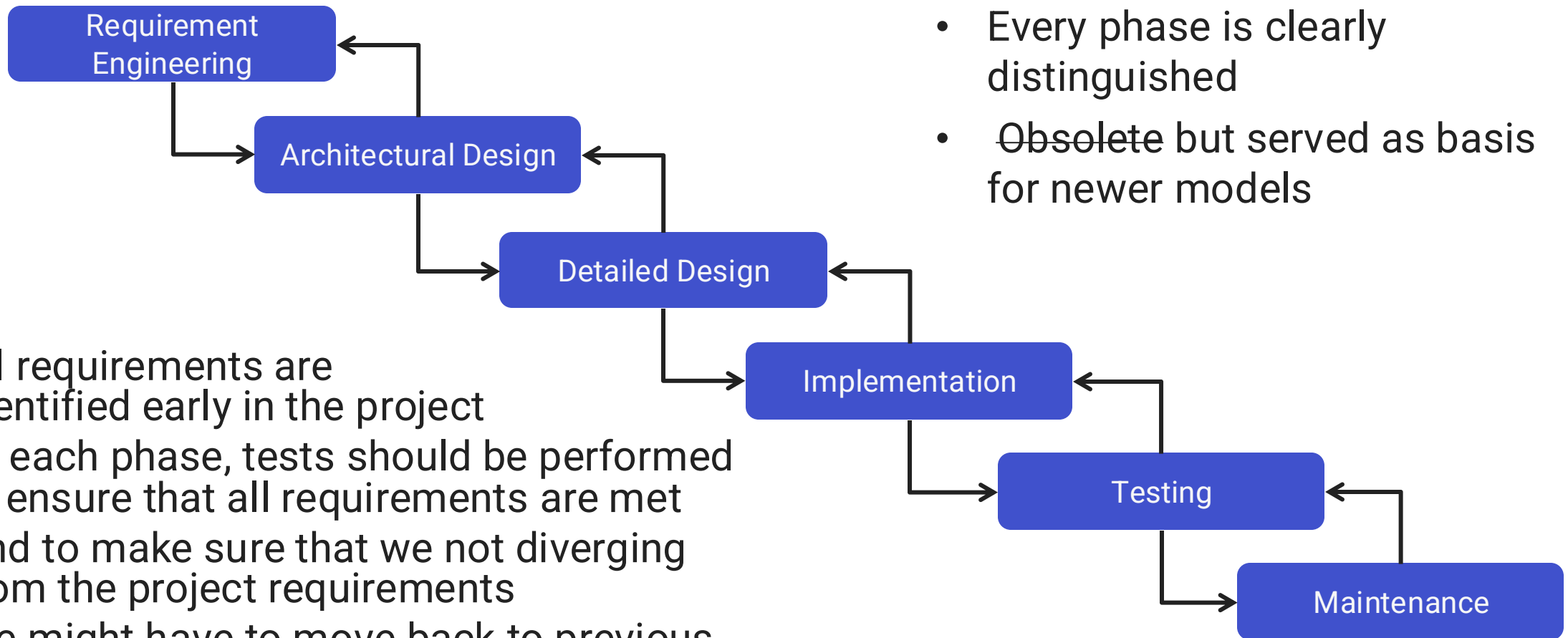
- **People:** Group of people that are involved with the project; **stakeholders**
- **Product:** The software product and the associated documents
- **Project:** Activities that carried out to produce the product
- **Process:** Framework that the team used to conduct the development activities. That is, the software development process

SOFTWARE DEVELOPMENT PROCESS **MODEL**

- **Sequential Models**

- Phases are executed in sequential manner
- Phase 2 can be executed only when Phase 1 is finished
- Example: Waterfall Model, V-Model

WATERFALL MODEL



- All requirements are identified early in the project
- At each phase, tests should be performed to ensure that all requirements are met
- And to make sure that we not diverging from the project requirements
- We might have to move back to previous phase

- Sequential
- Every phase is clearly distinguished
- Obsolete but served as basis for newer models

SOFTWARE DEVELOPMENT PROCESS **MODEL**

- **Iterative and Incremental Models**
 - Start with small portions of a software project
 - Repeatedly add portions into the projects
 - Example: Spiral Model, Unified Process, Prototyping
- Iterative vs. incremental
 - **Incremental** fundamentally means “**add onto something**”
 - Incremental development helps you improve your process
 - **Iterative** fundamentally means “**re-do**”
 - Iterative development helps you improve your product

SOFTWARE DEVELOPMENT PROCESS **MODEL**

- **Agile Processes:**
 - Evolved from iterative and incremental process
 - Intended to speed up software development and respond to *change*
 - Prefer code, person knowledge and customer involvement over documentation and contracts
- Example: Extreme Programming, Scrum, Rapid Application Development (RAD), etc.

SOFTWARE DEVELOPMENT PROCESS **MODEL**

- **Open-Source Process:** The process used to conduct open-source software development
- Open-source software projects are unlike ordinary (paid, closed source) software projects
 - The developers are attracted to the projects only after early software versions
 - Thus, requirements are rarely gathered
 - They select their own roles (not assigned by others)
 - Some projects may depend or even merged into other projects
 - Users are sometimes the coders
- That is, open-source projects are more self-organized compared to well-planned closed source ones

WHY DO WE NEED **SOFTWARE PROCESS**?

- Unlike other engineering/development problems
 - Buildings and constructions
 - Car making, city planning, gardening, etc.
- Software development, **changes** occur constantly
 - Requirements may be changed or added anytime
- Frequent changes are difficult to manage
 - Planning and cost estimation are difficult
- There could be more than one software system to consider
 - System under development (new versions)
 - Released systems (current version)

AGILE PROCESS

GOING AGILE

- Traditional software processes are “heavy” and does not cope well with changes
- They emphasize
 - Planning
 - Requirement documentation
 - Design documentation
- **Problem:** We might not know what we will really require
- **Agile Alliance** is formed in 2001 to produce development framework that is efficient and adaptable
- As a result, they produced the **Agile Manifesto**
- **Agile Process** or **Agile Development** refers to any software process that captures the values in Agile Manifesto

AGILE MANIFESTO

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

1. **Individuals and interactions** over processes and tools
2. **Working software** over comprehensive documentation
3. **Customer collaboration** over contract negotiation
4. **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

1. INDIVIDUALS AND INTERACTIONS

- Agile promotes cooperation and communication among experts to enhance their efficiency
- The development team should be self-organized and self-motivated
- Skilled individual should be able to adapt the tools to their needs, not the other way round
- Creativity should be used to solve problems, not predetermined rules and for the team to adapt to these rules

2. WORKING SOFTWARE

- Working software is the best indicator how well the project is going, not documents presentations
- Agile practices emphasize producing working program as soon as possible
- As the project progresses, more functionality will be incrementally added
- Stakeholders can see the progress through the produced working codes
- However, appropriate documentation is still necessary

3. CUSTOMER COLLABORATION

- All stakeholders should work as the same team
 - From developers to customers
 - Sometimes they are split by organizational layers
- Because:
 - All requirements may not be determined only in the beginning
 - The software may have to be changed, fixed or revised many times during the development, their combined skills would be benefit to the development
- Although contracts are important, interaction is also important especially when dealing with changes

4. RESPONDING TO CHANGE

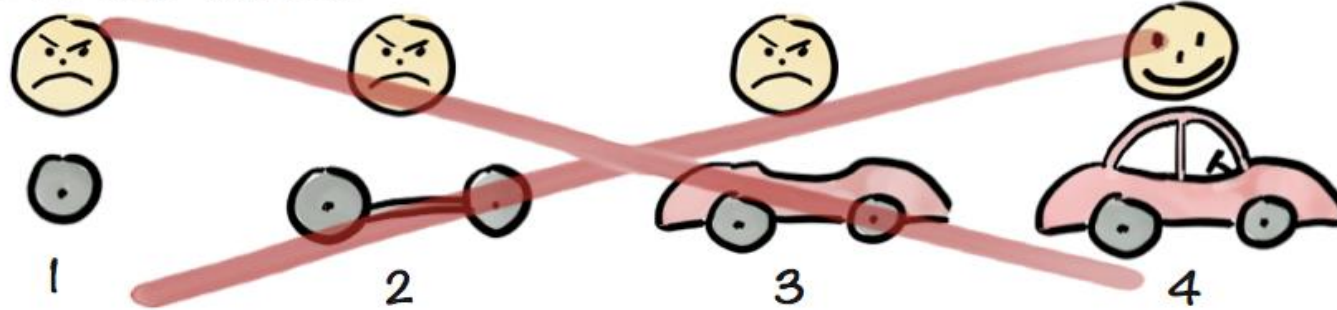
- Agile method embraces change and continuous development, even late in development
- Change is natural and inevitable
- When changes occur, we shall adapt to it rather than stick to the old plan
- An iteration in agile methods usually last one to six weeks to be able respond with change early

MVP

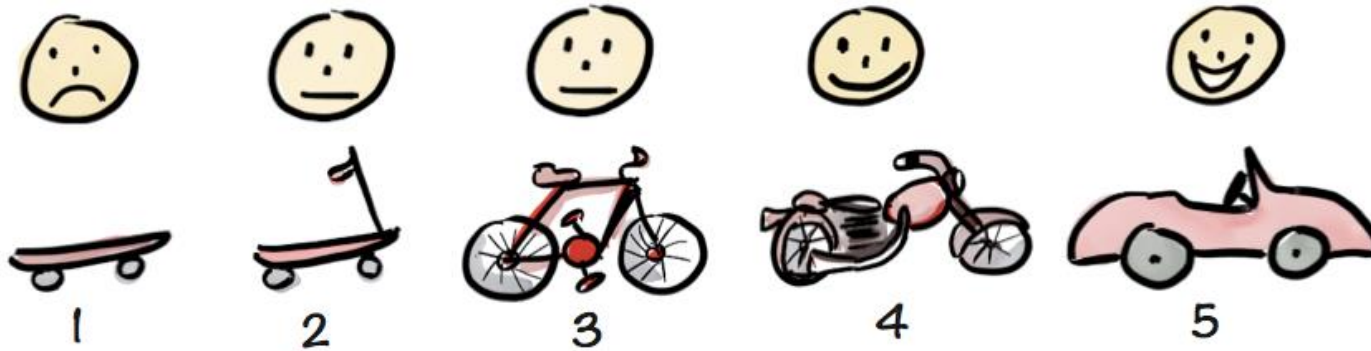
- **Minimum Viable Product:** The smallest thing that you can build that delivers customer value (and as a bonus captures some of the value back).
- Without the MVP, your product is not the product you want anymore
- In other words, they are the features that you cannot cut or ignore, otherwise, the product will not function at all

MINIMUM VIABLE

Not like this....



Like this!



by Henrik Kniberg

MoSCoW

- **MoSCoW prioritization** is used in software development to determine importance of requirements or functionalities
- MoSCoW categories:
 - **Must haves**: The functionalities must be implemented in current iteration
 - **Should haves**: The functionalities are important but is not an absolute necessary
 - **Could haves**: Desirable functionalities that are not necessary but preferred to have
 - **Won't haves**: Unimportant requirements that can wait until later releases

SCRUM

SCRUM

- In rugby, scrum is a method for restarting a play



- In software engineering, Scrum is an agile software development process that focuses on **project management** where it is difficult to plan ahead

WHY SCRUM?

Traditional methods are like relay races (work is performed sequentially)



Agile methods are like rugby (work is performed in parallel)



SCRUM OVERVIEW

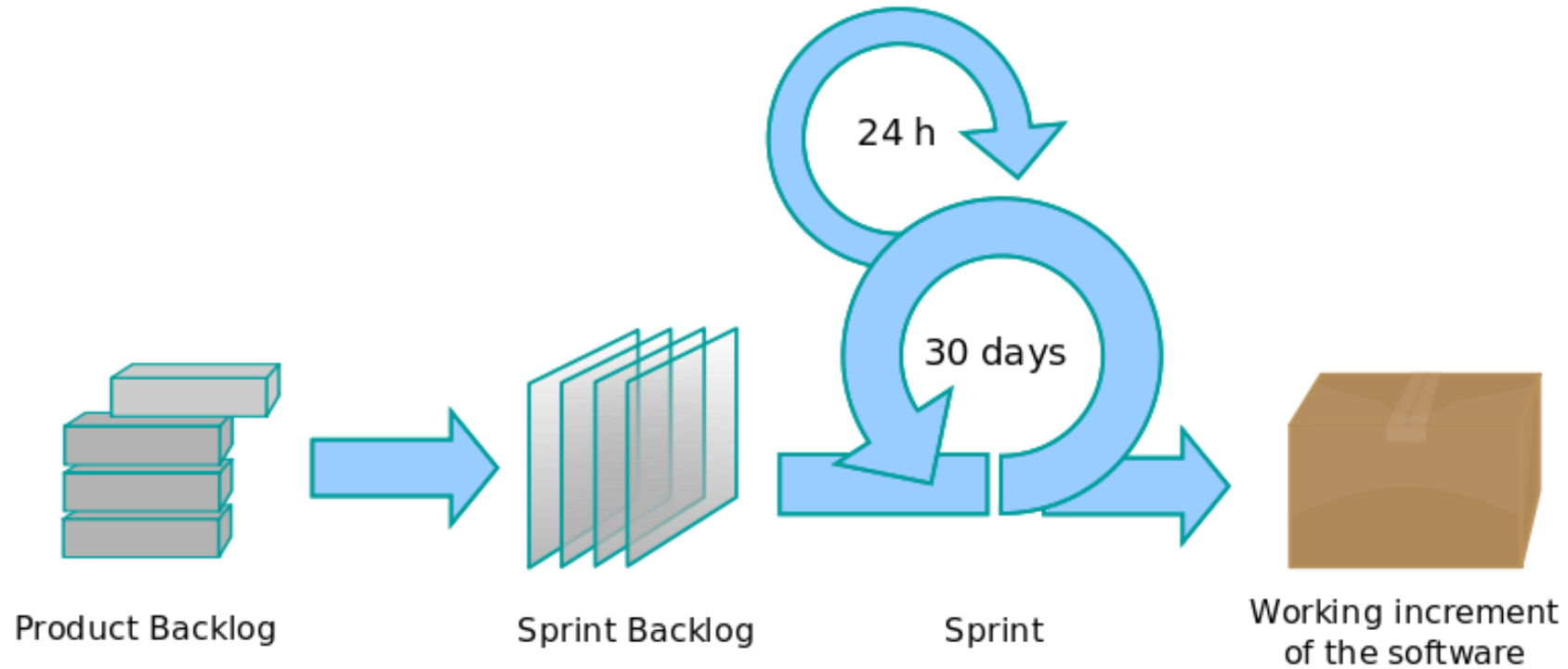
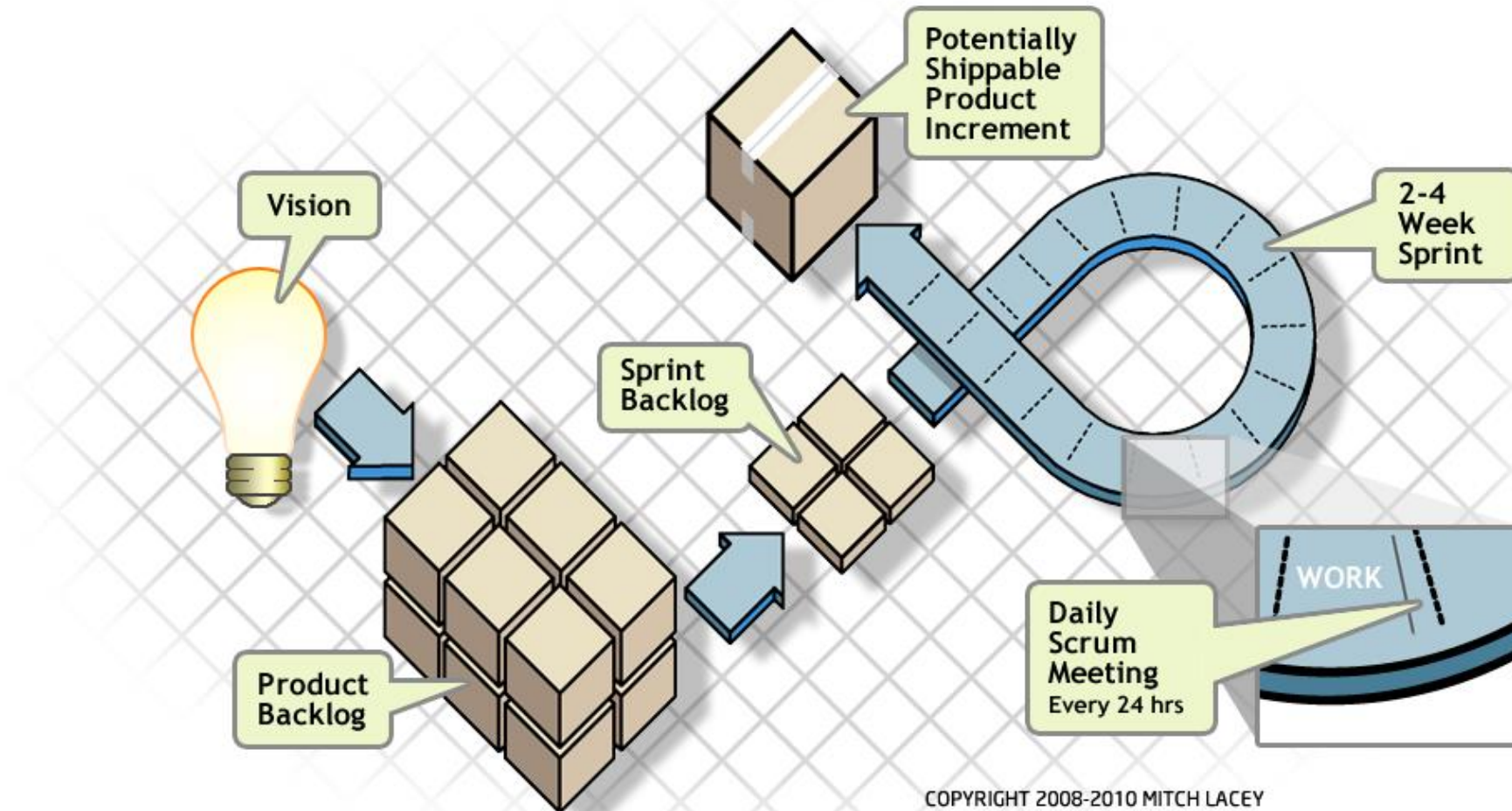


Image: http://en.wikipedia.org/wiki/File:Scrum_process.svg

SCRUM OVERVIEW



COPYRIGHT 2008-2010 MITCH LACEY
[HTTP://WWW.MITCHLACEY.COM](http://www.mitchlacey.com)

SCRUM COMPONENTS

- **Scrum Roles**
 - Scrum Master, Scrum Team, Product Owner
- **Scrum Artifacts**
 - Product Backlog, Sprint Backlog
 - Product Increment
 - Burndown Chart
- **Scrum Activities**
 - Kickoff Meeting
 - Sprint Planning Meeting
 - Sprint (“Iteration” in a Unified Process)
 - Daily Scrum Meeting
 - Sprint Review and Retrospective Meetings

SCRUM COMPONENTS

- **Scrum Roles**
 - Scrum Master, Scrum Team, Product Owner
- **Scrum Artifacts**
 - Product Backlog, Sprint Backlog
 - Product Increment
 - Burndown Chart
- **Scrum Activities**
 - Kickoff Meeting
 - Sprint Planning Meeting
 - Sprint (“Iteration” in a Unified Process)
 - Daily Scrum Meeting
 - Sprint Review and Retrospective Meetings

SCRUM ROLES

- **Product Owner**
 - Knows what needs to be built
 - Accepts or rejects work results
- **Scrum Master**
 - Represents management to the project
 - Typically filled by a project manager or team leader
 - Responsible for enacting scrum values and practices
 - Main job is to remove impediments
- **Scrum Team**
 - Typically, 5-9 people
 - Cross-functional skills (analysts, programmers, designers, testers)
 - Members should be full-time members
 - Team is self-organizing
 - Membership can change only between sprints.

SCRUM ARTIFACTS

- **Product Backlog**
 - **Definition:** Requirements for a system, expressed as a prioritized list of Backlog Items
 - Is managed and owned by a Product Owner
 - Spreadsheet (typically)
 - Usually created during the Project Kickoff Meeting
 - Can be changed and re-prioritized
- **Sprint Backlog**
 - **Definition:** A subset of Product Backlog Items, which defines the work to be done in a Sprint
 - Is created ONLY by Team members
 - Each item has its own status
 - Should be updated every day
- **Product Increment**
 - Integrated version of the software product
 - Required result of each sprint
- **Burndown Chart**
 - Graph of open tasks over time
 - Used for project controlling purposes

PRODUCT BACKLOG EXAMPLE


	Item #	Description	Est	By
Very High				
	1	Finish database versioning	16	KH
	2	Get rid of unneeded shared Java in database	8	KH
	-	Add licensing	-	-
	3	Concurrent user licensing	16	TG
	4	Demo / Eval licensing	16	TG
		Analysis Manager		
	5	File formats we support are out of date	160	TG
	6	Round-trip Analyses	250	MC
High				
	-	Enforce unique names	-	-
	7	In main application	24	KH
	8	In import	24	AM
	-	Admin Program	-	-
	9	Delete users	4	JM
	-	Analysis Manager	-	-
	10	When items are removed from an analysis, they should show up again in the pick list in lower 1/2 of the analysis tab	8	TG
	-	Query	-	-
	11	Support for wildcards when searching	16	T&A
	12	Sorting of number attributes to handle negative numbers	16	T&A
	13	Horizontal scrolling	12	T&A
	-	Population Genetics	-	-
	14	Frequency Manager	400	T&M
	15	Query Tool	400	T&M

Image: http://epf.eclipse.org/wikis/scrum/Scrum/workproducts/product_backlog_68345C16.html

SPRINT BACKLOG EXAMPLE (WITH USER STORIES)

User Story	Tasks	Day 1	Day 2	Day 3	Day 4	Day 5	...
As a member, I can read profiles of other members so that I can find someone to date.	Code the ...	8	4	8	0		
	Design the ...	16	12	10	4		
	Meet with Mary about ...	8	16	16	11		
	Design the UI	12	6	0	0		
	Automate tests ...	4	4	1	0		
	Code the other ...	8	8	8	8		
As a member, I can update my billing information.	Update security tests	6	6	4	0		
	Design a solution to ...	12	6	0	0		
	Write test plan	8	8	4	0		
	Automate tests ...	12	12	10	6		
	Code the ...	8	8	8	4		

**Estimated
Remaining
Work**



Taken from: <http://www.mountangoatsoftware.com/scrum/sprint-backlog/>

PRODUCT BACKLOG REFINEMENT

- Product Backlog can be revised anytime during the development
 - Remove stories that are not needed
 - Add new stories that are now needed
 - Re-prioritizing backlog items
 - Split the items down to smaller sizes
 - Understanding the items
 - Estimating efforts
- Process of revising the Product Backlog is called **Product Backlog Refinement (PBR)**

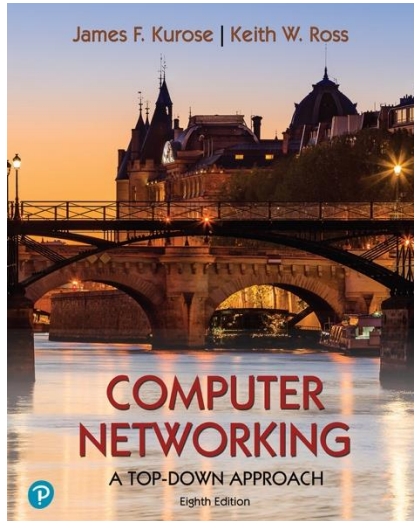
THE ROAD **AHEAD**

- We have discussed typical software development life cycle
- We will move on to developing web applications
 - Short recap on computer networks
 - Version control system
 - Front-end development
 - Back-end development

TENTATIVE OUTLINE

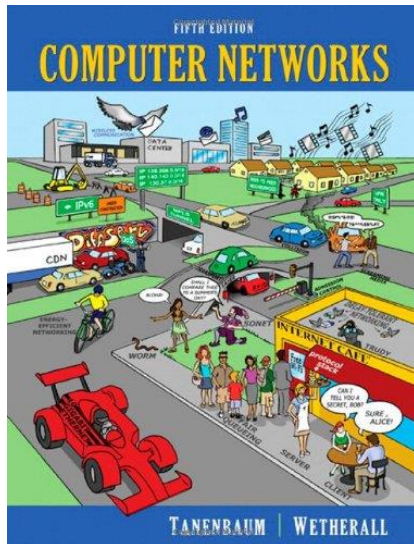
- Week 1: Introduction to software development process
- Week 2: Computer networks
- Week 3: Version control system
- Week 4: Creating a web page with HTML
- Week 5: Page styling with CSS
- Week 6-7: Introduction to JavaScript
- Week 8-9: Front-end development with vue.js
- Week 10-12: Back-end development with FastAPI
- Week 13: Web Server and Deployment
- Week 14: SSL

TEXTBOOKS



Computer Networking: A Top-Down Approach, 8th edition

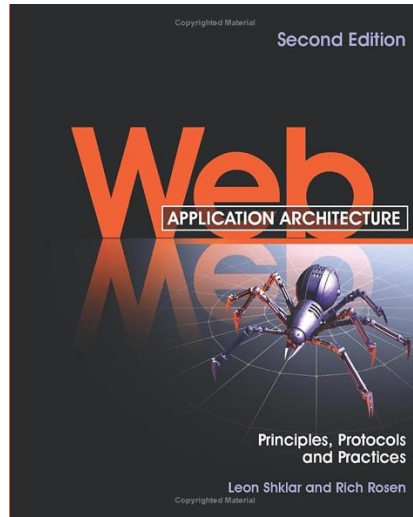
Jim F. Kurose and Keith W. Ross
Pearson, 2020



Computer Networks, 5th edition

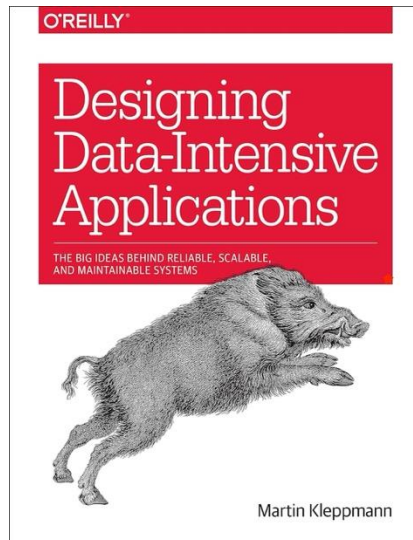
Andrew S. Tanenbaum
Prentice-Hall, 2010

RECOMMENDED BOOKS



Web Application Architecture: Principles, Protocols and Practices , 2nd edition

Leon Shklar and Rich Rosen
John Wiley and Sons, 2009



Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems, 1st edition

Martin Kleppmann
O'Reilly Media, 2017