

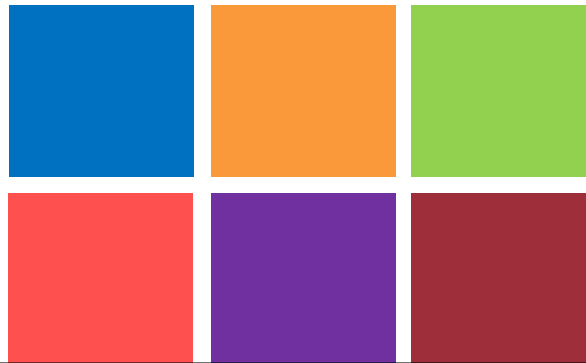
# BBM 382 – SOFTWARE ENGINEERING

SPRING 2019

## Lecture 1

Asst.Prof.Dr. Ayça TARHAN

Dr. Tuğba ERDOĞAN



## BBM 382 & 384 – Course homepage

<https://tinyurl.com/https-tinyurl-com-BBM382-84>

BBM382				Reading & UML Asgn. (see below)
Week #	Date	Content	Ch's	Due dates
1	February 28, 2019	Introduction, Waterfall SD	1, 2, 22, 23	
2	March 7, 2019	Prj. Mgt, Prj. Planning & Risk Mgt.		
3	March 14, 2019	Software Requirements and UML	4	RA-1
4	March 21, 2019	Exercise in class-System: BILSIS	4	
5	March 28, 2019	System Modeling	5,6	RA-2
6	April 4, 2019	Architectural Design	6	
7	April 11, 2019	Design & Imp. (UML modeling) / <b>UML assignment (see below)</b>	7	RA-3
8	April 18, 2019	MIDTERM		
9	April 25, 2019	Software Testing 1	8	
10	May 2, 2019	Software Testing 2	9	UML Assignment.
11	May 9, 2019	Software Evolution and Maintenance	9	
12	May 16, 2019	Change/Configuration Management	24	RA-4
13	May 23, 2019	Quality Management	3	
14	May 30, 2019	Software Process (Others)	3	

Components	Midterm Exam	25.00%
	Final Exam	40%
	UML assignment	15.00%
	Reading Assignments (4 RAs)	10.00%
	Attendance	10.00%

**Textbook:** Ian Sommerville, *Software Engineering*, Addison-Wesley; 10th edition, 2015.

Translation of 10th Ed (in Turkish):  
<https://www.nobelyayin.com/detay.asp?u=14776>



## CHAPTER 1 – INTRODUCTION

3



### Contents

- 1.1 Professional software development (covered)
- 1.2 Software engineering ethics
- 1.3 Case studies

4



## 1.1 PROFESSIONAL SOFTWARE DEVELOPMENT

5




## Software in Modern World

- We can't run the modern world without software.
  - **National infrastructures and utilities** are controlled by computer-based systems
  - **Most electrical products** include a computer and controlling software
  - **Industrial manufacturing and distribution** is completely computerized, as is the financial system.
  - **Entertainment**, including the music industry, computer games, and film and television, is software intensive

6





## Challenges

- Characteristics of software:
  - Intangible (abstract)
  - Changeable
- Characteristics of software development:
  - Human-intensive
  - Multi-disciplinary



## Software Failures – Main Reasons

### ■ *Increasing demands*

- Systems have to be built and delivered **more quickly**;
- **Larger**, even **more complex** systems are required;
- Systems have to have **new capabilities** that were previously thought to be impossible.
- Existing software engineering methods cannot cope and **new software engineering techniques** have to be developed to meet new these new demands.

### ■ *Low expectations*

- It is relatively **easy to write computer programs** without using software engineering methods and techniques.
- Many companies **do not use** software engineering **methods**.
- Consequently, their software is often **more expensive and less reliable** than it should be.

*We need better software engineering education and training to address this problem.*

9



## Software Failures – Expectations!



10



## Software Engineering is NOT Programming!

### Programming Vs. Engineering

Programming	Software Engineering
Personal activity (instrument)	Team activity (orchestra)
One aspect of software development	Large systems must be developed similar to other engineering practices
Concerned about accomplishing the objective of the program itself	Concerned about the entire solution, its feasibility, and future use

11



## History of Software Engineering

- The notion of 'software engineering' was first proposed in 1968 at a conference held to discuss what was then called the 'software crisis' (Naur and Randell, 1969).
  - It became clear that individual approaches to program development did not scale up to large and complex software systems. These were unreliable, cost more than expected, and were delivered late.
- Throughout the 1970s and 1980s, a variety of new software engineering techniques and methods were developed, such as structured programming, information hiding and object-oriented development. Tools and standard notations were developed and are now extensively used.
- See <http://www.SoftwareEngineering-9.com/Web/History/>

12



## Professional Software Development

- Professional software, intended for use by someone apart from its developer, is usually developed by teams rather than individuals. It is maintained and changed throughout its life.
- Software engineering is intended to support professional software development, rather than individual programming. It includes techniques that support program specification, design, and evolution.
  - None of which are normally relevant for personal software development.

13



## Professional Software Development

- Many people think that software is simply another word for computer programs. However, software is not just the programs themselves but also all associated documentation and configuration data that is required to make these programs operate correctly.
  - A professionally developed software system usually consists of system documentation, which describes the structure of the system; user documentation, which explains how to use the system, and web- sites for users to download recent product information.
- This is one of the important differences between professional and amateur software development.

14



## Software Products

- Software engineers are concerned with developing **software products** (i.e., software which can be sold to a customer).
- There are two kinds of software products:
  - *Generic products*
  - *Customized (or bespoke) products*

15



## Kinds of Software Products

- *Generic products* These are **stand-alone systems** that are produced by a development organization and **sold on the open market** to any customer who is able to buy them.
  - Examples include software for PCs such as databases, word processors, drawing packages, and project-management tools. It also includes some specific purpose applications such as library information systems, accounting systems, or systems for maintaining dental records.
- *Customized (or bespoke) products* These are systems that are **commissioned by a particular customer**. A software contractor develops the software especially for that customer.
  - Examples of this type of software include control systems for electronic devices, systems written to support a particular business process, and air traffic control systems.

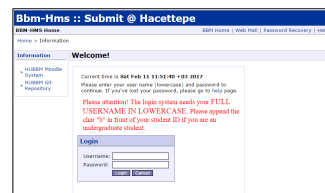
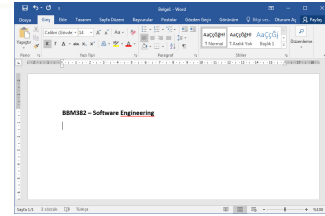
16





## Kinds of Software Products

- An important difference between these types of software is
  - In *generic products*, the organization that develops the software controls the **software specification** (e.g. MS Word) →
  - For *custom products*, the specification is usually developed and controlled by the organization that is buying the software (e.g. Submit system) →
- The mix of two is also possible!
  - Systems are being built with a generic product as a base, which is then adapted to suit the requirements of a customer.
    - Enterprise Resource Planning (ERP) systems, such as the SAP system, are the best examples.



17



## Quality

- When we talk about the **quality** of professional software, we have to take into account that the software is used and changed by people apart from its developers.
  - **Quality** is therefore not just **concerned with** what the software does; it has to include the software's behavior while it is executing and the structure and organization of the system programs and associated documentation.
- This is so-called **quality** or **non-functional software attributes**.
  - Examples of these attributes are the *software's response time to a user query* and the *understandability of the program code*.
  - The specific set of attributes that you might expect from a software system obviously depends on its application domain. Therefore, a banking system must be secure, an interactive game must be responsive, a telephone switching system must be reliable, and so on.

18



## Essential Attributes of Good Software

Product characteristics	Description
Maintainability	Software should be written in such a way so that it can evolve to meet the changing needs of customers. This is a critical attribute because software change is an inevitable requirement of a changing business environment.
Dependability and security	Software dependability includes a range of characteristics including reliability, security, and safety. Dependable software should not cause physical or economic damage in the event of system failure. Malicious users should not be able to access or damage the system.
Efficiency	Software should not make wasteful use of system resources such as memory and processor cycles. Efficiency therefore includes responsiveness, processing time, memory utilization, etc.
Acceptability	Software must be acceptable to the type of users for which it is designed. This means that it must be understandable, usable, and compatible with other systems that they use.

19



## Software Engineering

- **Software engineering** is an engineering discipline that is concerned with all aspects of software production from the early stages of system specification through to maintaining the system after it has gone into use.
  - *Engineering discipline* - Engineers make things work. They apply theories, methods, and tools where these are appropriate. However, they use them selectively and always try to discover solutions to problems even when there are no applicable theories and methods. Engineers also recognize that they must work to organizational and financial constraints so they look for solutions within these constraints.
  - *All aspects of software production* - Software engineering is not just concerned with the technical processes of software development. It also includes activities such as software project management and the development of tools, methods, and theories to support software production.

20



## Engineering Discipline

- **Engineering** is about getting repeatable results of the required quality within the schedule and budget.
  - This often **involves making compromises**—engineers cannot be perfectionists.
  - People writing programs for themselves, however, can spend as much time as they wish on program development.
- In general, **software engineers** adopt a systematic and organized approach to their work, as this is often the most effective way to produce high-quality software.
  - However, engineering is all about **selecting the most appropriate method for a set of circumstances** so a more creative, less formal approach to development may be effective in some circumstances.
  - Less formal development is particularly appropriate for the development of web-based systems, which requires a blend of software and graphical design skills.

21



## Software Engineering is Important

Software engineering is important for two reasons:

- More and more, individuals and society rely on advanced software systems. We need to be able to produce **reliable and trustworthy systems** economically and quickly.
- It is usually cheaper, in the long run, **to use software engineering methods and techniques** for software systems rather than just write the programs as if it was a personal programming project.
  - For most types of systems, the majority of costs are the costs of changing the software after it has gone into use.

22



## Software Engineering vs. Computer Science

- **Computer science** is concerned with the theories and methods that underlie computers and software systems, whereas software engineering is concerned with the practical problems of producing software.
- Some knowledge of computer science is essential for software engineers in the same way that some knowledge of physics is essential for electrical engineers.
- Computer science theory, however, is often most applicable to relatively small programs. Elegant theories of computer science cannot always be applied to large, complex problems that require a software solution.

23



## Software Engineering vs. System Engineering

- System engineering is concerned with all aspects of the development and evolution of complex systems where software plays a major role.
- System engineering is therefore concerned with hardware development, policy and process design and system deployment, as well as software engineering.
- System engineers are involved in specifying the system, defining its overall architecture, and then integrating the different parts to create the finished system. They are less concerned with the engineering of the system components (hardware, software, etc.)

**The BIGGER picture!**

24



## General Issues of Software

- *Heterogeneity* Increasingly, systems are required to operate as distributed systems across networks that include different types of computer and mobile devices. As well as running on general-purpose computers, software may also have to execute on mobile phones. You often have to integrate new software with older legacy systems written in different programming languages. The challenge here is to develop techniques for building dependable software that is flexible enough to cope with this heterogeneity.

25



## General Issues of Software

- *Business and social change* Business and society are changing incredibly quickly as emerging economies develop and new technologies become available. They need to be able to change their existing software and to rapidly develop new software. Many traditional software engineering techniques are time consuming and delivery of new systems often takes longer than planned. They need to evolve so that the time required for software to deliver value to its customers is reduced.
- *Security and trust* As software is intertwined with all aspects of our lives, it is essential that we can trust that software. This is especially true for remote software systems accessed through a web page or web service interface. We have to make sure that malicious users cannot attack our software and that information security is maintained.

26



## Software Engineering Diversity

- **Software engineering** is a systematic approach to the production of software that takes into account practical cost, schedule, and dependability issues, as well as the needs of software customers and producers.
  - How this systematic approach is actually implemented varies dramatically depending on the **organization developing the software, the type of software**, and **the people involved** in the development process.
  - There are no universal software engineering methods and techniques that are suitable for all systems and all companies. Rather, **a diverse set of software engineering methods and tools has evolved over the past 50 years.**

27



## Types of Software Applications

- There are many different types of application including:
  - *Stand-alone applications*
  - *Interactive transaction-based applications*
  - *Embedded control systems*
  - *Batch processing systems*
  - *Entertainment systems*
  - *Systems for modeling and simulation*
  - *Data collection systems*
  - *Systems of systems*

28



## Software Engineering Diversity

- You use **different software engineering techniques** for each type of system because the software has quite different characteristics.
  - For example, an embedded control system in an automobile is safety-critical and is burned into ROM when installed in the vehicle. It is therefore very expensive to change. Such a system needs **very extensive verification and validation** so that the chances of having to recall cars after sale to fix software problems are minimized. User interaction is minimal (or perhaps nonexistent) so there is no need to use a development process that relies on user interface prototyping.
  - For a web-based system, an approach based on **iterative development and delivery** may be appropriate, with the system being composed of reusable components. However, such an approach may be impractical for a system of systems, where detailed specifications of the system interactions have to be specified in advance so that each system can be separately developed.

29



## Software Engineering Diversity

- Nevertheless, there are **software engineering fundamentals** that apply to all types of software systems:
  - They should be **developed using a managed and understood development process**. The organization developing the software should plan the development process and have clear ideas of what will be produced and when it will be completed. Of course, different processes are used for different types of software.
  - Dependability and performance are important for all types of systems. Software should **behave as expected, without failures** and should be available for use when it is required. It should **be safe in its operation** and, as far as possible, should **be secure against external attack**. The system should **perform efficiently** and should **not waste resources**.

30



## Software Engineering Diversity

- Understanding and managing the software specification and requirements (what the software should do) are important.
  - You have to know what different customers and users of the system expect from it and you have to manage their expectations so that a useful system can be delivered within budget and to schedule.
- You should make as effective use as possible of existing resources. This means that, where appropriate, you should reuse software that has already been developed rather than write new software.

31



## Software Engineering and the Web

This radical change in software organization has, obviously, led to changes in the ways that **web-based systems** are engineered:

- Software reuse has become the dominant approach for constructing web-based systems. When building these systems, you think about how you can assemble them from pre-existing software components and systems.
- It is now generally recognized that it is impractical to specify all the requirements for such systems in advance. Web-based systems should be developed and delivered incrementally.
- User interfaces are constrained by the capabilities of web browsers. Although technologies such as AJAX mean that rich interfaces can be created within a web browser, these technologies are still difficult to use. Web forms with local scripting are more commonly used. Application interfaces on web-based systems are often poorer than the specially designed user interfaces on PC system products.

32





## CHAPTER 2 – SOFTWARE PROCESSES

33



### Contents

- 2.1 Software process models (covered)
- 2.2 Process activities
- 2.3 Coping with change
- 2.4 The Rational Unified Process

34



## Software Process

The systematic approach that is used in software engineering is sometimes called a **software process**. A software process is a sequence of activities that leads to the production of a software product:

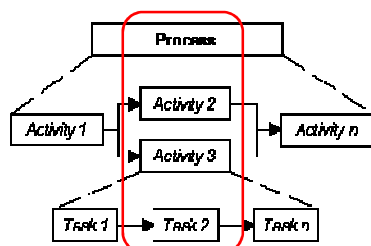
- **Software specification**, where customers and engineers define the software that is to be produced and the constraints on its operation.
- **Software development**, where the software is designed and programmed.
- **Software validation**, where the software is checked to ensure that it is what the customer requires.
- **Software evolution**, where the software is modified to reflect changing customer and market requirements.
- There are also supporting process activities such as **documentation** and **software configuration management**

35

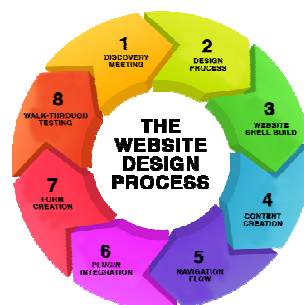


## Software Process

- A structured set of activities required to develop a software system.
  - Development of software from scratch in a standard programming language
  - Development by extending and modifying existing systems or by configuring and integrating generic (off-the-shelf) software or system components



Ref. <http://www.chambers.com.au/glossary/activity.php>



Ref. <http://www.dkgcreative.com.au/web-design-company-web-developer/web-design-process/>

36



## Software Process

- Different types of systems need different development processes.
  - For example, *real-time software* in an aircraft has to be completely specified before development begins.
  - In *e-commerce systems*, the specification and the program are usually developed together.
- Consequently, these generic activities may be organized in different ways and described at different levels of detail depending on the type of software being developed.
  - There are many different types of software. There is no universal software engineering method or technique that is applicable for all of these.

37



## Process Descriptions

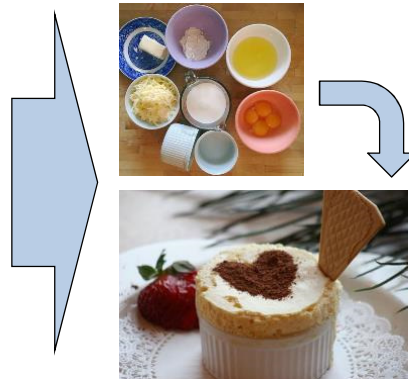
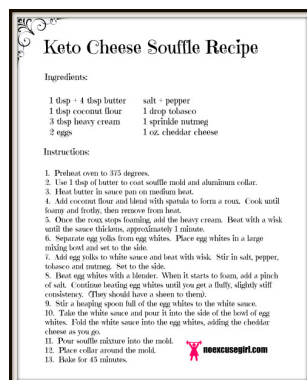
- When we describe and discuss processes, we usually talk about the activities in these processes
  - e.g., specifying a data model, designing a user interface, and the ordering of these activities.

### ■ Analogy:

Sufle receipe  
(process desc.)

&

Sufle cooking  
(process)



38

### PSP0 Process Script

<b>Purpose</b>	To guide the development of module-level programs	
<b>Entry Criteria</b>	<ul style="list-style-type: none"> <li>- Problem description</li> <li>- PSP0 Project Plan Summary form</li> <li>- Time and Defect Recording logs</li> <li>- Defect Type standard</li> <li>- Stopwatch (optional)</li> </ul>	
<b>Step</b>	<b>Activities</b>	<b>Description</b>
1	Planning	<ul style="list-style-type: none"> <li>- Produce or obtain a requirements statement.</li> <li>- Estimate the required development time.</li> <li>- Enter the plan data in the Project Plan Summary form.</li> <li>- Complete the Time Recording log.</li> </ul>
2	Development	<ul style="list-style-type: none"> <li>- Design the program.</li> <li>- Implement the design.</li> <li>- Compile the program, and fix and log all defects found.</li> <li>- Test the program, and fix and log all defects found.</li> <li>- Complete the Time Recording log.</li> </ul>
3	Postmortem	Complete the Project Plan Summary form with actual time, defect, and size data.
<b>Exit Criteria</b>	<ul style="list-style-type: none"> <li>- A thoroughly tested program</li> <li>- Completed Project Plan Summary form with estimated and actual data</li> <li>- Completed Time and Defect Recording logs</li> </ul>	

## Software Process Descriptions:

### Example (Personal Software Process)

```

graph TD
    Requirements --> Planning
    Planning --> Design
    Design --> Code
    Code --> Compile
    Compile --> Test
    Test --> PM
    PM --> FinishedProduct[Finished product]
    Scripts -- guide --> Code
    Logs --> Test
    Logs --> PM
    ProjectSummary[Project summary] --> FinishedProduct
    ProjectSummary --> SummaryReport[Project and process data summary report]
  
```

39

## Software Process Descriptions

- Process descriptions may also include:
  - **Products**, which are the **outcomes of a process activity**;
    - For example, the outcome of the activity of architectural design may be a model of the software architecture.
  - **Roles**, which reflect the **responsibilities of the people involved** in the process;
    - Examples of roles are project manager, configuration manager, programmer, etc.
  - **Pre- and post-conditions**, which are **statements that are true before and after a process activity** has been enacted or a product produced.
    - For example, before architectural design begins, a pre-condition may be that all requirements have been approved by the customer; after this activity is finished, a post-condition might be that the UML models describing the architecture have been reviewed.

40



## Process Standardization

- Software processes can be improved by process standardization where the diversity in software processes across an organization is reduced.
  - This leads to **improved communication and a reduction in training time**, and makes **automated process support** more economical.
  - Standardization is also an important first step in introducing new software engineering methods and techniques and good software engineering practice.

41



## Plan-driven and agile processes

- **Plan-driven processes** are processes where all of the process activities are planned in advance and progress is measured against this plan.
- **In agile processes**, planning is incremental and it is easier to change the process to reflect changing customer requirements.
  - As Boehm and Turner (2003) discuss, **each approach is suitable for different types of software**.
  - In practice, most practical processes include elements of both plan-driven and agile approaches.
  - There are no right or wrong software processes – they are tailored according to specific needs of software development.

42



## 2.1 SOFTWARE PROCESS MODELS

43



### Software Process Models

- A **software process model** is an abstract representation of a software process.
  - Each process model represents a process from a particular perspective, and thus provides only partial information about that process.
    - For example, a **process activity model** shows the activities and their sequence but may not show the roles of the people involved in these activities.
    - In this section, we introduce a number of very **general process models** and present these from an architectural perspective.
      - That is, we see the framework of the process but not the details of specific activities.

44



## Software Process Models

- These generic models are not definitive descriptions of software processes. Rather, they are **abstractions** of the process that can be used to explain different approaches to software development.
  - You can think of them as **process frameworks** that may be extended and adapted to create more specific software engineering processes.

45



## Software Process Models

- **The waterfall model**
  - Plan-driven model
  - Separate and distinct phases of specification and development
- **Incremental development**
  - Specification, development and validation are interleaved
  - May be plan-driven or agile
- **Reuse-oriented software engineering**
  - The system is assembled from existing components
  - May be plan-driven or agile

46



## Software Process Models (..continued)

### ■ The waterfall model

- Plan-driven model. / This approach takes the fundamental process activities of specification, development, validation, and evolution, and represents them as **separate process phases in sequence**.

### ■ Incremental development

- May be plan-driven or agile. / This approach interleaves the activities of specification, development, and validation. The system is developed as a series of versions (**increments**), with each version adding functionality to the previous version.

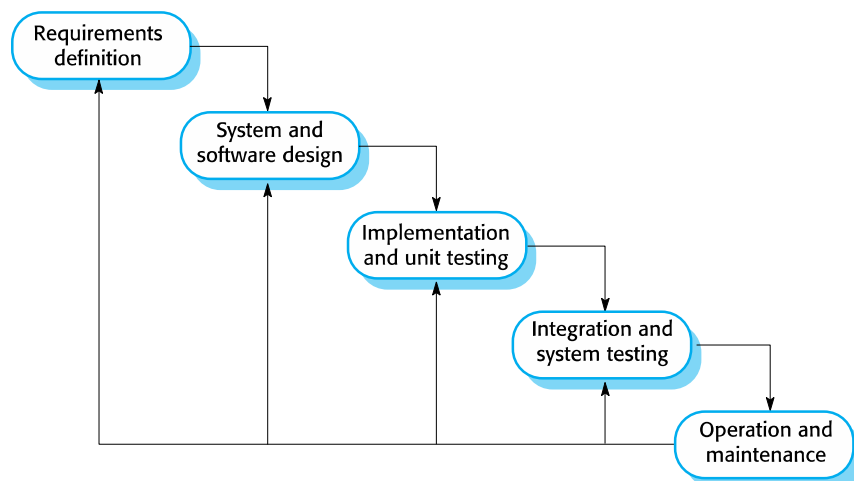
### ■ Reuse-oriented software engineering

- May be plan-driven or agile. / This approach is based on the existence of a significant number of **reusable components**. The system development process focuses on integrating these components into a system rather than developing them from scratch.

47



## The Waterfall Model



Because of the cascade from one phase to another, this model is known as the 'waterfall model.'

In a **plan-driven process**—in principle, you must plan and schedule all of the process activities before starting work on them.

48





## Waterfall Model Phases

- There are separate identified **phases** in the waterfall model:
  - Requirements analysis and definition
  - System and software design
  - Implementation and unit testing
  - Integration and system testing
  - Operation and maintenance
- The **main drawback** of the waterfall model is the difficulty of accommodating change after the process is underway. In principle, a phase has to be complete before moving onto the next phase.

49



## Waterfall Model Phases

- *Requirements analysis and definition.* The system's services, constraints, and goals are established by consultation with system users. They are then defined in detail and serve as a **system specification**.
- *System and software design.* The systems design process allocates the requirements to either hardware or software systems by establishing an **overall system architecture**. Software design involves identifying and describing the fundamental software system abstractions and their relationships.
- *Implementation and unit testing.* The software design is realized as a set of programs or **program units**. Unit testing involves verifying that each unit meets its specification.
- *Integration and system testing.*
- *Operation and maintenance.*

50



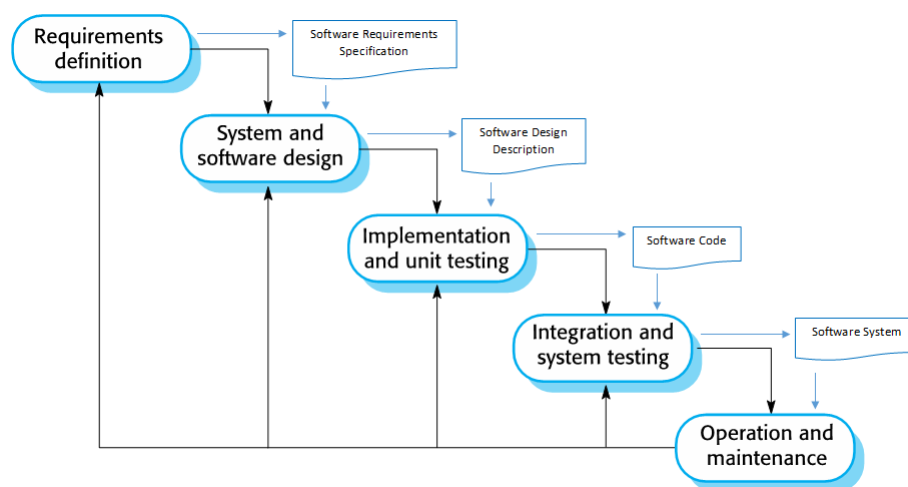
## Waterfall Model Phases

- *Requirements analysis and definition.*
- *System and software design.*
- *Implementation and unit testing.*
- *Integration and system testing.* The individual program units or programs are integrated and tested as a **complete system** to ensure that the software requirements have been met. After testing, the software system is delivered to the customer.
- *Operation and maintenance.* The system is installed and put into practical use. Maintenance involves correcting errors which were not discovered in earlier stages of the life cycle, improving the implementation of system units and enhancing the system's services as new requirements are discovered.

51



## Waterfall Model Outputs



52



## The Waterfall Model

- In principle, the result of each phase is one or more documents that are approved ('signed off'). The following phase should not start until the previous phase has finished.
  - In practice, these stages overlap and feed information to each other.
  - During design, problems with requirements are identified. During coding, design problems are found and so on.
- The software process is not a simple linear model but involves feedback from one phase to another. Documents produced in each phase may then have to be modified to reflect the changes made.

53



## The Waterfall Model - Benefits

- The waterfall model is **consistent with other engineering process models**.
- **Documentation** is produced at each phase.
  - This makes the process visible so managers can monitor progress against the development plan.
- As is **easier to use a common management model in other engineering projects** for the whole project, software processes based on the waterfall model are still commonly used.

54



## Waterfall Model - Problems

- It is **difficult to accommodate change** after the process is underway.
  - A phase has to be complete before moving onto the next phase.
- **Inflexible partitioning of the project into distinct stages** makes it difficult to respond to changing customer requirements.
  - Few business systems have stable requirements.
- For the majority of systems this model **does not offer significant cost-benefits** over other approaches to system development.

55



## Waterfall Model - Usage

- In principle, the waterfall model should only be used when the requirements are well understood and unlikely to change radically during system development.
- The waterfall model is **mostly used for large systems engineering projects** where a system is developed at several sites.
  - In those circumstances, the plan-driven nature of the waterfall model helps coordinate the work.

56



Richard H. Thayer, **Software System Engineering:  
A Tutorial**, IEEE Computer, 2002.

Read & Summarize in max 2 A4 pages.

DO NOT COPY AND PASTE!

Submit your summary via Submit System

Due Date: **14.March.2019** 23:59 (no extension)

## READING & SUMMARY ASSIGNMENT - 1

57



FOR STUDENT READING...

58



## More on Software Quality Attributes

### software quality attributes.

- accessibility
- accountability
- accuracy
- adaptability
- administrability
- affordability
- agility
- auditability
- availability
- credibility
- standards compliance
- process capabilities
- compatibility
- composability
- configurability
- correctness
- customizability
- degradability
- demonstrability
- dependability
- deployability
- distributability
- durability
- evolvability
- extensibility
- fidelity
- flexibility
- installability
- integrity
- interchangeability
- interoperability
- learnability
- maintainability
- manageability
- mobility
- modularity
- nomadicity
- operability
- portability
- precision
- predictability
- recoverability
- relevance
- reliability
- repeatability
- reproducibility
- responsiveness
- reusability
- robustness
- safety
- scalability
- seamlessness
- serviceability (a.k.a. supportability)
- securability
- simplicity
- stability
- survivability
- sustainability
- tailorability
- testability
- timeliness
- understandability
- usability

59



## Types of Software Applications

- There are many different types of application including:
  - *Stand-alone applications* These are application systems that run on a local computer, such as a PC.
    - They include all necessary functionality and do not need to be connected to a network.
    - Examples of such applications are office applications on a PC, CAD programs, photo manipulation software, etc.

60



## Types of Software Applications

- *Stand-alone applications*
- *Interactive transaction-based applications* These are applications that execute on a remote computer and that are accessed by users from their own PCs or terminals.
  - These include web applications such as e-commerce applications where you can interact with a remote system to buy goods and services.
  - This class of application also includes business systems, where a business provides access to its systems through a web browser or special-purpose client program and cloud-based services, such as mail and photo sharing.
  - Interactive applications often incorporate a large data store that is accessed and updated in each transaction.

61



## Types of Software Applications

- *Stand-alone applications*
- *Interactive transaction-based applications*
- *Embedded control systems* These are software control systems that control and manage hardware devices.
  - Numerically, there are probably more embedded systems than any other type of system.
  - Examples of embedded systems include the software in a mobile (cell) phone, software that controls anti-lock braking in a car, and software in a microwave oven to control the cooking process.

62



## Types of Software Applications

- *Stand-alone applications*
- *Interactive transaction-based applications*
- *Embedded control systems*
- *Batch processing systems* These are business systems that are designed to process data in large batches.
  - They process large numbers of individual inputs to create corresponding outputs.
  - Examples of batch systems include periodic billing systems, such as phone billing systems, and salary payment systems.

63



## Types of Software Applications

- *Stand-alone applications*
- *Interactive transaction-based applications*
- *Embedded control systems*
- *Batch processing systems*
- *Entertainment systems* These are systems that are primarily for personal use and which are intended to entertain the user.
  - Most of these systems are games of one kind or another.
  - The quality of the user interaction offered is the most important distinguishing characteristic of entertainment systems.

64





## Types of Software Applications

- *Stand-alone applications*
- *Interactive transaction-based applications*
- *Embedded control systems*
- *Batch processing systems*
- *Entertainment systems*
- *Systems for modeling and simulation* These are systems that are developed by scientists and engineers to model physical processes or situations, which include many, separate, interacting objects.
  - These are often computationally intensive and require high-performance parallel systems for execution.

65



## Types of Software Applications

- *Stand-alone applications*
- *Interactive transaction-based applications*
- *Embedded control systems*
- *Batch processing systems*
- *Entertainment systems*
- *Systems for modeling and simulation*
- *Data collection systems* These are systems that collect data from their environment using a set of sensors and send that data to other systems for processing.
  - The software has to interact with sensors and often is installed in a hostile environment such as inside an engine or in a remote location.

66



## Types of Software Applications

- *Stand-alone applications*
- *Interactive transaction-based applications*
- *Embedded control systems*
- *Batch processing systems*
- *Entertainment systems*
- *Systems for modeling and simulation*
- *Data collection systems*
- **Systems of systems** These are systems that are composed of a number of other software systems.
  - Some of these may be generic software products, such as a spreadsheet program. Other systems in the assembly may be specially written for that environment.

67



## Case Studies – Three types of systems \*

( \* Refer to case studies in Sec 1.3 for more details)

- **An embedded system** This is a system where the software controls a hardware (medical) device and is embedded in that device. Issues in embedded systems typically include physical size, responsiveness, power management, etc. / **Insulin pump control system**
- **An information system** This is a system whose primary purpose is to manage and provide access to a database of information. Issues in information systems include security, usability, privacy, and maintaining data integrity. / **Medical records system**
- **A sensor-based data collection system** This is a system whose primary purpose is to collect data from a set of sensors and process that data in some way. The key requirements of such systems are reliability, even in hostile environmental conditions, and maintainability. / **Wilderness weather station**

68