CMSC 128 Laboratory Handout 1

System Documentation

Prepared by: Kristine Elaine P. Bautista, Instructor 1, ICS, UPLB

Software Requirements Specification (SRS)

- a document that is created when a detailed description of all aspects of the software to be built must be specified before the project is to commence

Sample Outline of a Complete SRS

1. Introduction
- 1.1. Purpose
- 1.2. Document Conventions
- 1.3. Intended Audience and Reading Suggestions
- 1.4. Project Scope
- 1.5. References

2. Overall Description
- 2.1. Product Perspective
- 2.2. Product Features
- 2.3. User Classes and Characteristics
- 2.4. Operating Environment
- 2.5. Design and Implementation Constraints
- 2.6. User Documentation
- 2.7. Assumptions and Dependencies

3. System Features
- 3.1. System Feature 1
- 3.2. System Feature 2 (and so on)

4. External Interface Requirements
- 4.1. User Interfaces
- 4.2. Hardware Interfaces
- 4.3. Software Interfaces
- 4.4. Communication Interfaces

5. Other Nonfunctional Requirements
- 5.1. Performance Requirements
- 5.2. Safety Requirements
- 5.3. Security Requirements
- 5.4. Software Quality Attributes

6. Other Requirements

Appendix A: Glossary

Appendix B: Analysis Models

Appendix C: Issues List

Unified Modeling Language (UML)

- standardized general-purpose modeling language in the field of software engineering
- may be used to visualize, specify, construct and document the artifacts of a software-intensive system
- has different types of diagrams based on structure or behavior

Different Types of Diagrams in UML

- Structural Diagrams
  - describe structural or static relationships among schema objects, data objects, and software components
  - Examples:
    - *a. Class Diagrams*
    - b. Object Diagrams
    - c. Component Diagrams
    - d. Deployment Diagrams
- Behavioral Diagrams
  - describe the behavioral or dynamic relationships among other components
  - Examples:
    - *a. Activity Diagrams*
    - *b. Use Case Diagrams*
    - c. Collaboration Diagrams
    - d. Statechart Diagrams

Class Diagram

- provides a static or structural view of a system
- models classes including their attributes, operations, and their relationships and associations with other classes

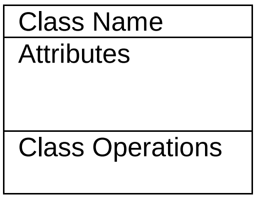


Figure 1. Class Representation

Activity Diagram

- depicts the dynamic behavior of a system or part of a system through the flow of control between actions that the system performs
- is similar to a *flowchart* except that it can show concurrent flows.

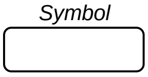
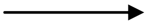
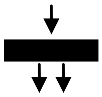
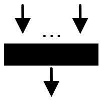
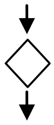
Symbol	*Name*	*Meaning*
	Action Node	corresponds to a task performed by the system
	Flow of Control	an arrow between two actions nodes means that the first action is complete the section action begins
	Initial Node	indicates the starting point of the activity
	Final Node	Indicates the end of the activity
	Fork	separation of activities into two or more concurrent activities
	Join	way of synchronizing concurrent flows of control
 (two or more outgoing arrows)	Decision Node	corresponds to a branch in the flow of control based on a condition

Table 1 Activity Diagram Representations

Use Case

- helps determine the functionality and features of the software from the user's perspective
- describes how a user interacts with the system by defining the steps required to accomplish a specific goal

Use Case Diagram Representations

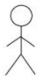
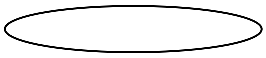
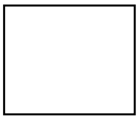
Symbol	*Name*
	Actor
	Use Cases
―――――――――――	Connector
<<includes> ---------------------------->	Includes Relationships
	Groups use cases done by a certain actor

Table 2 Use Case Diagram Representations

Gantt Chart

- developed by Henry Gantt in the 1910s
- essentially a bar chart, one bar for each task, with the horizontal units being units of time
- it is good for monitoring the progress of project as it moves along

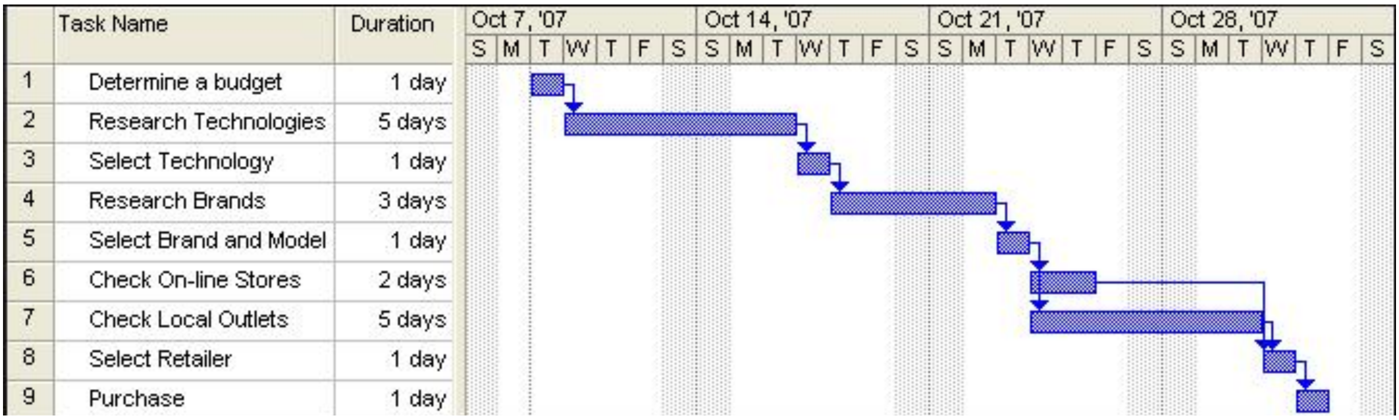
Parts of a Gantt Chart

- Task Identifier (ID)
- Task Name
- Start Time of Task
- Finish Time of Task
- Calendar–includes the bar chart that represents the duration of each tasks, critical path, etc.

Slack – measure of *excess* time and resources available to complete a task

Critical Path

- the longest path that contains all the tasks that must be done in the defined sequential order
- it is called *critical* because if any task in the critical path is delayed, the entire project will be completely late



	Task Name	Duration
1	Determine a budget	1 day
2	Research Technologies	5 days
3	Select Technology	1 day
4	Research Brands	3 days
5	Select Brand and Model	1 day
6	Check On-line Stores	2 days
7	Check Local Outlets	5 days
8	Select Retailer	1 day
9	Purchase	1 day

Figure 2 Gantt Chart Example

References:

- Pressman, R. S. (2010). *Software Engineering a Practitioner's Approach (Alternate Edition)*. 7th Edition. New York, NY: McGraw-Hill.
- Elmasri R. and Navathe S. V. (2000). *Fundamentals of Database Systems*. 3rd Edition. Boston, MA: Addison Wesley Longman, Inc.
- Satzinger, J. W., Jackson, R. B., and Burd, S. D. (2007). *Systems Analysis and Design in a Changing World*. 4th Edition. Cengage Learning.