Chapter 7 Notes

**SW Design and Implementation**

* Is the stage in the SWE Process at which an executable SW system is developed
* For small simple systems, all other processes are merged with the SW Design and Implementation process
* For large system, SW Design and Implementation is only one set of processes (requirements engineering, verification, etc.)

**SW Design**

* The activity in which you identify software components and their relationships based on customer’s requirements

**Implementation**

* The activity of realizing the design as a program

Although separate in most cases, design and implementation are both closely related and you should normally take implementation issues into account when developing a design.

**Object-Oriented Design using the UML (Section 7.1)**

* An object oriented system is made up of interacting objects that maintain their own local state and provide operations on the state.
* Object Oriented Design processes involve designing object classes and the relationships between these classes.
* OO system are easier to change than other systems
* OO Object help improve the understandability and hence, the maintainability of the design
* The process of developing a system design from concept to design do:
  + Understand and define the context and the exeternal interactions with the system (System Context and Interactions)
  + Design the system architecture (Architectural Design)
  + Identify the principal objects in the system (Object Class Identification)
  + Develop design models (Design Models)
  + Specify interfaces (Interface Specification)

**System Context and Interactions (Section 7.1.1)**

* Need to develop and understanding of the relationships between the SW that is being designed and its external environment.
  + This is essential for deciding how to provide the required system functionality and how to structure the system to communicate with its environment
  + Understanding of the context also lets you establish the boundaries of the system also lets you establish the boundaries of the system which will help you decide what features are implemented in the system being designed and what features are in other associated systems.
  + System context models and interaction models provide complementary views of the relationships between a system and its environment
    - System Context Model
      * A structural model that demonstrates the other systems in the environment of the system being developed
      * May be represented using associations
        + Associations simply show that there are some relationships between the entities involved in the association
    - Interaction Model
      * Dynamic Model that shows how the system interacts with its environment as it is used
      * Should be abstract and avoid too much detail
      * You can use an Use Case Model
        + Each possible interaction is named in an ellipse and the external entity involved in the interaction is represented by a stick figure
        + Each Use Case should be described in structured natural language that includes

System (name of system)

Use Case (name of use case)

Actors

Dat (summarizes what is supposed to happen)

Stimulus

Response

Comments

**Architectural Design (Section 7.1.2)**

* Use the information from the step above as a basis for designing the system architecture.
* In this step, you identify the major components of your system and their interactions and then many organize the components using an architectural pattern such as Layered or Client-Server model

**Object Class Identification (Section 7.1.3)**

* Purpose here is to identify the objects in your system
* How to do this…3 approaches
  + Use a grammatical analysis of a natural language description of the system to be constructed. Objects and attributes are nouns; operations or services are verbs
  + Use tangible entities (things) in the application domain such as aircraft, roles (doctor, etc.), events such as requests, interactions such as meetings, locations such as offices, and so on.
  + Use a scenario-based analysis where various scenarios of system use are identified and analyzed in turn. As each scenario is analyzed, the team responsible for the analysis must identify the required objects, attributes, and operations
* You can use your knowledge of the application domain to identify the objects
* After finding the objects, you can refine the design by trying to incorporate them into an inheritance hierarchy.

**Design Models (Section 7.1.4)**

* These models show the objects or object classes in a system. They show the associations and relationships between these entities.
* These models form a bridge between system requirements and the implementation of a system.
* When you use the UML to develop a design, you will normally develop 2 kinds of design model:
  + Structural Models
    - Describe the static structure of the system using object classes and their relationships
    - Important relationships that may be documented at this stage are generalization (inheritance) relationships, uses/used-by relationships, and composition relationships.
  + Dynamic Models
    - Describe the dynamic structure of the system and show the interactions between the system objects
    - Interactions that may be documented include the sequence of service requests made by objects and the state changes that are triggered by these object interactions
* 3 Models are useful for adding detail to use case and architectural models
  + Subsystem Models
    - Show logical groupings of objects into coherent subsystems
    - Shows how the system is organized into groups of related objects
    - These are Static (Structural) Models
  + Sequence Models
    - Show the sequence of object interactions
    - These are represented using a UML sequence or a collaboration diagram
    - When documenting a design, you should produce a sequence model for each significant interaction. (There should be a Sequence model for each Use Case Model)
    - Used to model the combined behavior of a group of object
    - Sequence Models are Dynamic Models
  + State Machine Models
    - Show how individual objects change their state in response to events
    - Represented in the UML using state diagrams
    - Summarize the behavior of an object or a subsystem in response to messages and events.
    - State Machine Models are Dynamic Models

**Interface Specification (Section 7.1.5)**

* Specification of interfaces between components in the design
* This is needed so that objects and subsystems can be designed in parallel
* Interface design is concerned with specifying the detail of the interface to an object or to a group of objects.
* Should not reveal details of data representation in interface design

**Design Patterns (Section 7.2)**

* A pattern is a description of the problem and the essence of its solution, so that the solution may be reused in different settings. It is a well-tried solution to a common problem
* Patterns include the following sections
  + Pattern name
  + Description
  + Problem Description
  + Solution Description
  + Consequences
* Gang of Four define 4 essential elements
  + A name that is meaningful to the pattern
  + A description of the problem area that explains when the pattern may be applied
  + A solution description
  + A statement of the consequences

**Implementation Issues (Section 7.3)**

* May appear during SW Implementation
* Aspects of implementation that are important to SWE
  + Reuse
    - Most modern SW reuses existing components or systems
  + Configuration Management
    - Many different versions of each software component are created. If you don’t keep track of these versions in a configuration management system, you are liable to include to include the wrong versions of these components in your system
  + Host-Target development
    - Production software does not usually execute on the same computer as the SW development environment. The host and target systems are sometimes of the same type but are usually different

**Reuse (Section 7.3.1)**

* SW Reuse is possible at different levels
  + Abstraction Level
    - Instead of reusing software directly, you use knowledge of successful abstractions in the design of your software. Examples are Design and Architectural Patterns
  + Object Level
    - Directly reuse objects from a library rather than writing the code yourself.
    - You must find libraries and discover if the objects and functions deliver the functionality you need
  + Component Level
    - Reuse components (are collections of objects that cooperate with each other to provide related functions and services). You often have to adapt and extend the component by adding some code of your own
  + System Level
    - Reuse entire application systems
    - Usually involves some kind of configuration of these systems. This may be done by adding and modifying code (if you are reusing a SW product line) or by using the system’s own configuration interface.
* Advantages
  + Develop new systems more quickly
  + Cheaper
  + Lower risk
* Costs associated with Reuse
  + Costs of time spent looking for SW to reuse and assessing whether or not is meets your needs. You may have to tests the SW to make sure that it will work in your environment
  + Costs of buying the reusable SW
  + Costs of adapting and configuring the reusable SW components or systems to felect the requirements of the system that you are developing
  + Costs of integrating reusable SW elements with each other and with new code that you have developed

**Configuration Management (Section 7.3.2)**

* Ex. Of why this is important
  + When people are developing SW, you have to make sure that team members don’t interfere with each other’s’ work. If two people are working on the same component, their changes must be coordinated.
  + You must also ensure that everyone can access the most up-to-date versions of SW components
  + When something goes wrong, you should be able to recover to a previous version of the system
* Is the process of managing a changing software system
* Goal is to support the system integration process so that all developers can access the project code and documents ina controlled way, find out what changes have been made, and compile and link components to create a system.
* 3 fundamental configuration management activities
  + Version management
    - Support is provided to keep track of the different versions of SW components. T
  + System Integration
    - Support is provided to help developers define what versions of components are used to create each version of the system. This description is then used to build a system automatically by compiling and linking the required components
    - Ex is Unix/Linux Make file
  + Problem Tracking
    - Support is provided to allow users to report bugs and other problems, and to allow developers to see who is working on these problems

**Host-Target Development (Section 7.3.3)**

* SW may be developed one computer but may run on a separate machine that can be completely different
* It is more common for the development and execution platforms to be different
* Simulators may be used to execute the program on the development machine but this is expensive and may not be available for some hardware architectures
* A SW development platform should provide the following tools (and more) to support software engineering processes
  + An integrated compiler and syntax-directed editing system that allows you to create, edit, and compile code
  + A language debugging system
  + Graphical editing tools, such as tools to edit UML Models
  + Testing tools such as JUnit that can automatically run a set of tests on a new version of a program
  + Project support tools that help you organize the code for different development projects
* All of the features above are usually placed within a single IDE like Eclipse
* You also need to decide how the developed SW will be deployed onto the target platform. This is easy for embedded systems where the SW is usually placed on one computer. Issues may include
  + The hardware and software requirements of a component
    - If a component is designed for a specific hardware architecture or relies on some other software system, it must obviously be deployed on a platform that provides the required hardware and software support
  + Availability requirements of the system
    - High-availability systems may require components to be deployed on more than one platform. This means that, in the event of platform failure, an alternative implementation of the component is available.
  + Component communications
    - If there is a high level of communication traffic between components, it usually makes sense to deploy them on the same platform or on platforms that are physically close to one other. This reduces communications latency.

**Open Source Development (Section 7.4)**

* An approach to development in which the source code of a software system is published and volunteers are invited to participate in the development process.
* Usually cheap to acquire open source SW
* 2 Main open sources issues
  + Should the product that is being developed make use of open source components?
  + Should an open source approach be used for the software’s development?

**Open Source Licensing (Section 7.4.1)**

* Legally, the developer of the code still owns to code. They can place restrictions on how it is used by including legally binding conditions in an open source software license.
* 3 General Models
  + GNU General Public License (GPL)
    - “Reciprocal”
    - If you use open source software that is licensed under the GPL license, then you must make that SW open source also.
  + GNU Lesser General Public License (LGPL)
    - You can write components that link to open source code without having to publish the source code of these components. But if you change the licensed component, then you must publish this as open source
  + Berkeley Standard Distribution License (BSD)
    - Non-reciprocal
    - Not obliged to republish any changes to modifications made to open source code. You can include the code in proprietary systems that are sold. If you use open source components, you must acknowledge the original creator of the code
* Companies that manage projects that use open source code should:
  + Establish a system for maintaining info about open source components that are downloaded and used. You have to keep a copy of the license for each component that was valid at the time the component was used.
  + Be aware of the different types of licenses and understand how a component is licensed before it is used
  + Educate people about open source
  + Have auditing system in place so that terms of a license are not broken in a team
  + Participate in the open source community

**Key Points**

* SW Design and Implementation are interleaved activities. The level of detail in the design depends on the type of system being developed and whether you are using a plan-driven or agile approach
* The process of OOD includes activities to design the system architecture, identify objects in the system, describe the design using different object models, and documents the component interfaces.
* A range of different models may be produced during an OOD process. These include static models (class models, generalization models, association models) and dynamic models (sequence models, state machine models)
* Component interfaces must be defined precisely so that other objects can use them. A UML interface stereotype may be used to define interfaces
* When developing SW, you should always consider the possibility of reusing existing SW, either as components, services, or complete systems
* Configuration Management is the process of managing changes to an evolving SW system, it is essential when a team of people are cooperating to develop SW.
* Most software development is Host-Target development. You use an IDE on a host machine to develop the SW which is transferred to a target machine for execution
* Open source development involves making the source code of a system publicly available. This means that many people can propose changes and improvements to the software.