Software Engineering

Lecture 9

Design and Implementation

Dr. Obuhuma James

Description

This topic explores the design and implementation activities of the software development process. In most cases, the two software development processes are normally interleaved and carried out at the same time. The two phases will be discussed separately in a manner that clearly outlines the Software Engineering aspects involved in each of them. Finally, the concept of open-source development as applied in software engineering is discussed.

Learning Outcomes

By the end of this topic, you will be able to:

- Differentiate between software design and implementation as applied in Software Engineering.
- Describe software design and software implementation strategies as used in Software Engineering.
- Comprehend on the concept of open-source development as applied in Software Engineering.

Overview of Design and Implementation

A series of previous topics have been laying a foundation on software engineering from a requirements gathering and analysis perspective. The end result has been system models and architectural designs, which are already startups for the design activity.

- Software design is a process of transforming user requirements into a format that is aimed at helping the programmer in the next phase of implementation.
- The implementation phase entails actual coding in a programming language that leads to the development of the real software.

In the software design phase, a shift from the problem space to the solution space is experienced. Software design and implementation attempts to fulfill the requirements as specified in the requirements specification document.

Software Design

Through software design, three levels of results are achieved:

1. Architectural designs – this forms the highest abstract version of the system. Architectural design is as covered in the previous topic.
2. High-level design – this breaks the concept of having a single entity with multiple components as for the case of architectural design into less-abstracted view of subsystems and modules. The interaction among the subsystems and modules is taken care of in the design. The prime goal is to bring out a modular implementation structure of a system together with its components.
3. Detailed design – this entails implementation of what is visualized to be a system and its subsystems. The approach is more detailed on modules and their implementations, by defining the logical structure of each module and their interfaces that facilitate communication with other modules.

Considering the main focus of system design being a solution space, which relates to the how to implement a given solution, the various inputs and outputs for system design are as follows

System Design Inputs include:

a) Statement of work
b) A plan for the determination of requirements
c) A current situational analysis status
d) System requirements plus a conceptual data model, DFDs, and metadata.

System Design Outputs include:

a) Proposed system's infrastructure and organizational changes.
b) A data schema which may be relational or other type of schema.
c) Metadata defining tables or files and columns or data items.
d) A function hierarchy diagram or web page map, graphically and visually describing the program structure.
e) Module program of pseudocode.
f) Proposed system prototype.

Software design can be achieved through a two broad design approaches, namely, top-down design and bottom-up design. Depending on whichever approach, either a structured design, function-oriented design, or object-oriented design strategy may be used.

Top-Down Design

The strategy uses a modular approach for development of system designs. The top-down nature comes from the fact that the strategy begins from the top or highest-level modules as it heads down to lowest-level modules. The main module is subdivided into submodules based on tasks performed by each module and so on until we achieve lowest level modules that cannot allow further subdivision. This is as shown in Figure 1.

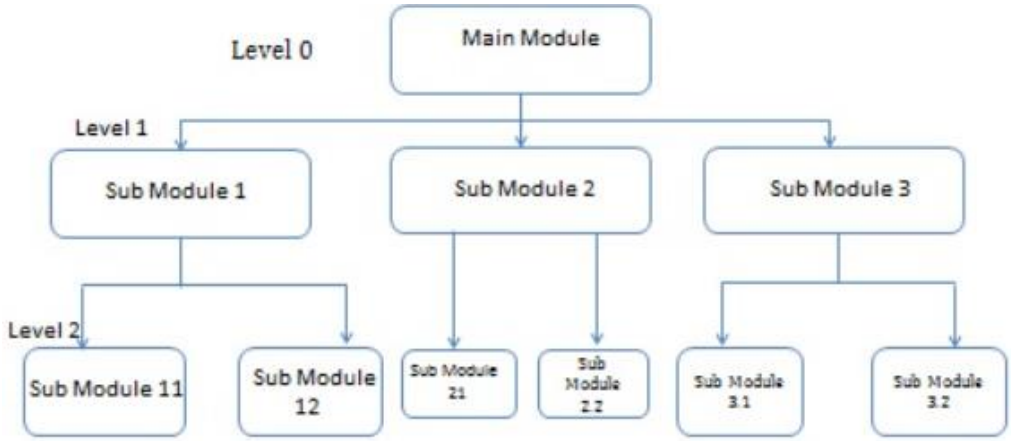

Figure 1. Top-Down Design Approach

Bottom-Up Design

The strategy uses a modular approach for development of system designs. It works as an exact inverse of the top-down strategy where the beginning point is the bottom or most basic level modules, all the way to highest level modules. The most basic modules are grouped together based on their functionalities to achieve the next higher-level modules and so on until the highest-level module is generated. This is as shown in Figure 2.

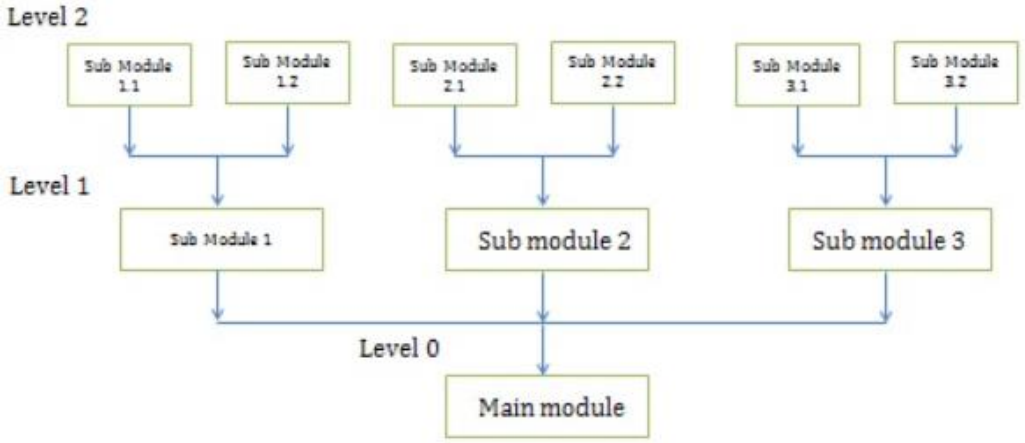

Figure 2. Bottom-Up Design Approach

Object-Oriented Design Strategy

This is the most common design strategy in use. It involves several different system models that require a lot of development and maintenance effort which may not be cost effective for small systems. Many different object-oriented design processes exit with their usage dependent on organisations employing them. However, according to Sommerville [1], the following activities are common in all these processes:

1. Definition of context and modes of system usage.

A proper understanding of the relationships between the software being designed and its external environment is essential for decision making on required system functionality and how to structure the system's communication with its environment [1]. Establishment of boundaries of the system results from a proper contextual understanding. Thus, context models like context diagrams leading to DFDs and use case diagrams covered in topic 6 are essential towards showing other systems in the environment of the system under development. In the same line, interaction models covered in topic 7 outline how the system should interact with its environment during its usage period.

2. Design of the system architecture.

Architectural designs result from interactions between the system being proposed and its well determined and understood environment. Through architectural design, major system components and their interactions are outlined [1]. The components may be organized using any architecture and its associated pattern as covered in topic 8 on architectural design.

3. Identification of principal system objects.

System objects help in the creation of classes, where a class is a group of related objects. According to Sommerville [1], object identification is often a difficult task with no magic formula in place. It just depends on the skill, experience, and domain knowledge of system designers. Every class generated from a group of objects must have a set of properties identifying it and method or operations that can be performed on it. Two system models can be built at this point, namely, object diagrams and class diagrams. Class diagrams were covered in topic 7.

4. Development of design models.

Design models show objects and object classes and relationships between them [1]. Thus, the concept of static and dynamic models as discussed in topic 7 is applied in deciding the system models to be developed in this context. Examples of design models include subsystem models, sequence models, state machine models, and many other models as covered in previous topics on system modeling.

5. Specification of object interfaces.

Object interfaces have to be specified for better and parallel design of objects and other components [1]. UML uses class diagrams as covered in topic 7 for interface specification.

Design Patterns

A design pattern is a way of reusing abstract knowledge about a problem and its solution [1]. It is a description of the problem and the essence of its solution and should be sufficiently abstract to be reused in different settings. Pattern descriptions usually make use of object-oriented characteristics such as inheritance and polymorphism. Design patterns portray the following elements:

1. Name – a meaningful pattern identifier.
2. Problem description.
3. Solution description – not a concrete design but a template for a design solution that can be instantiated in different ways.
4. Consequences
5. The results and trade-offs of applying the pattern.

According to Sommerville [1], to use patterns in your design, you need to recognize that any design problem you are facing may have an associated pattern that can be applied.

- Observer pattern – tell several objects that the state of some other object has changed
- Facade pattern – tidy up the interfaces to a number of related objects that have often been developed incrementally
- Iterator pattern – provide a standard way of accessing the elements in a collection, irrespective of how that collection is implemented
- Decorator pattern – allow for the possibility of extending the functionality of an existing class at run-time

Software Implementation

Software implementation is the process of realizing the design as a program [1]. This is where actual coding is performed. During coding, the lines of code keep multiplying to the extent that program management becomes tough. Thus, structured programming becomes a better approach for Software Engineers to be able to manage the ever-growing codes. Through structured programming, the use of subroutines and loops is adopted for better code clarity and improvement on efficiency. It also helps in code organization and reduction in coding times. Three main concepts are embraced through structured programming:

- Top-down analysis – software development problems are broken down into small pieces, each of which is individually solved with clear steps outlined.
- Modular programming – relates to top-down analysis by creating smaller chunks of codes targeting to address a specific task. Such task-oriented chunks of code are referred to as modules, subprograms or subroutines.
- Structured coding – in relation to the top-down analysis, the concept of structured coding leads to subdivision of modules into further smaller units of code based on their execution order. It is worth noting that structured coding uses control structures to organize its instructions in definable patterns as opposed to structured programming that uses control structures for program flow control.

Software Implementation Issues

According to Sommerville [1], the following three issues affect software implementation and should hence be put into consideration:

- Reuse – most modern software is constructed by reusing existing components or systems. When you are developing software, you should make as much use as possible of existing code.
- Configuration management – during the development process, you have to keep track of the many different versions of each software component in a configuration management system.
- Host-target development – production software does not usually execute on the same computer as the software development environment. Rather, you develop it on one computer (the host system) and execute it on a separate computer (the target system).

Software Implementation Tools

Software implementation requires a number of development tools to be available:

- An integrated compiler and syntax-directed editing system for creating, editing and compiling code.
- A debugging mechanism or system.
- A graphical editing tools, such as tools for editing UML models.
- Testing tools, that can automatically run a set of tests on a new version of a program.
- Project support tools for code organization.

In most case, software development tools are often grouped and packaged into an integrated development environment (IDE). Thus, an IDE is a set of software tools that supports different aspects of software development, within some common framework and user interface [1]. They are created to support development in a specific programming language. For instance, the NetBeans IDE supports the Java programming language.

Open-source Development

The current world is shifting towards open-source development, an approach to software development in which the system's source code is published and volunteers are invited to participate in the development process. Open-source development is rooted in the Free Software Foundation (www.fsf.org), that advocates for nonproprietary source codes that should be freely available to the public for examination and modification at will. Legally, the code developer owns the code.

- He/she may put restrictions on how it is used through inclusion of legally binding conditions in an open-source software license.
- Some open-source developers believe that if an open source component is used to develop a new system, then that system should also be open source.
- Others are willing to allow their code to be used without this restriction. In such cases, the developed systems may be proprietary and sold as closed source systems.

Open-source software uses the GNU General Public License (GPL) which is a reciprocal kind of license, meaning that if you use open source software that is licensed under the GPL license, then you must make that software open source. The GNU Lesser General Public License (LGPL) is a variant of the GPL license where you can write components that link to open-source code without having to publish the source of these components. On the other hand, the Berkley Standard Distribution (BSD) License is a non-reciprocal license, which means you are not obliged to re-publish any changes or modifications made to open-source code. You can include the code in proprietary systems that are sold.

It is hence advisable for system developers to embrace good license management practices. Such that

- There should be a system in place for maintaining information about open-source components that are downloaded and used.
- The developers should be aware of the different types of licenses and understand how a component is licensed before it is used.
- The developer should be aware of evolution pathways for components.
- There should be education of people about open source.
- There should be system auditing in place.

- There should be encouragement efforts towards participation in the open-source community.

Summary

The topic has explored design and implementation, a major software development process activity. Two main development activities are featured, namely, software design and software implementation. Software design creates a blueprint of what is to be transformed into a final system through software implementation. Software design approaches and strategies have been discussed. Similarly, software implementation approaches, tools, and issues have been covered. The topic concludes with a mention of the concept of open-source development.

Check Points

1. Differentiate between the top-down and bottom-up approaches as used in software design.
2. Discuss the five common activities involved in the object-oriented design process.
3. Describe the three main concepts embraced through structured programming.
4. Differentiate between structured programming and structured coding.
5. Discuss the three issues associated with software implementation.
6. Discuss the concept of open-source development as used in Software Engineering.

Learning Resources

Core Textbooks

1. Sommerville, I., Software Engineering, 10th Edition, Addison Wesley, 2016.

Other Resources

2. Pries, K. H., & Quigley, J. M. (2010). Scrum project management. CRC press.
3. Schwaber, K. (2004). Agile project management with Scrum. Microsoft press.

References

[1] Sommerville, I., Software Engineering, 10th Edition, Addison Wesley, 2016.

[2] Pries, K. H., & Quigley, J. M. (2010). Scrum project management. CRC press.

[3] Schwaber, K. (2004). Agile project management with Scrum. Microsoft press.

[4] Gotterbarn, D., Miller, K., & Rogerson, S. (1997). Software engineering code of ethics. Communications of the ACM, 40(11), 110-118.

[5] Pressman, R. S. (2005). Software engineering: a practitioner's approach. Palgrave macmillan.

[6] Kendall, K. E. and Kendall, J. E., Systems Analysis and Design, 8th Edition, Pearson, 2011.