Preview: IT4772 : Software Design and Modeling

Syllabus

Course Overview

Software design is the second phase of the software development lifecycle and is generally considered to consist of two steps, architectural design and detailed design. Architectural design describes how software is organized into classes. Software detailed design focuses on desired behaviors and structures of the classes.

This course is focused on the practice of software design and modeling. During your course-long project, you will apply foundational software design principles to create structural and behavioral models based on product requirements. The goal of the project is to create an effective software design, user interface, and prototype.

Course Competencies

(Read Only)

To successfully complete this course, you will be expected to:

1. Apply appropriate design strategies and methods in the software design process.
2. Translate functional and non-functional requirements into a software solution that meets organization and user needs.
3. Create models that describe a software application’s structure and behavior.
4. Evaluate the quality of a software design.
5. Contribute to effective design reviews.

Course Prerequisites

Prerequisite(s): Completion of or concurrent registration in IT3345; IT3348 or IT3349.

Syllabus >> Course Materials

Required

The materials listed below are required to complete the learning activities in this course.

Integrated Materials

Many of your required books are available via the VitalSource Bookshelf link in the courseroom, located in your Course Tools. Registered learners in a Resource Kit program can access these materials using the courseroom link on the Friday before the course start date. Some materials are available only in hard-copy format or by using an access code. For these materials, you will receive an email with further instructions for access. Visit the Course Materials page on Campus for more information.

Book

Dennis, A., & Tegarden, D., & Wixom, B. (2015). *Systems analysis and design with UML*(5th ed.). Hoboken, NJ: John Wiley & Sons. ISBN: 9781118804674

eBook

Sommerville, I. (2016). *Software engineering* (10th ed.). Boston, MA: Pearson. ISBN: 9780133943030.

Library

The following required readings are provided in the Capella University Library or linked directly in this course. To find specific readings by journal or book title, use Journal and Book Locator. Refer to the Journal and Book Locator library guide to learn how to use this tool.

- Anderson, J., & Franceschi, H. (2016). *Java illuminated: An active learning approach* (4th ed.). Burlington, MA: Jones and Bartlett Learning.
- Barrera, D. G., & Diaz, M. (2013). *Communicating systems with UML 2.* John Wiley & Sons.

- Cusick. J. (2013). *Durable ideas in software engineering. Concepts, methods, and approaches from my virtual toolbox.* Bentham Science Publishers.
- Giesenow, H. (2017). Microsoft Entity Framework: Understanding the EF modeling languages [Video]. Skillsoft Ireland.
- Giesenow, H. (2017). Microsoft Entity Framework: What is the entity framework [Video]. Skillsoft Ireland.
- Gomaa, H. (2016). *Real-time software design for embedded systems.* New York, NY: Cambridge University Press.
- Harrington, J. L. (2016). *Relational database design and implementation.* Elsevier Science.
- Hendry, B. (2017). Oracle Commerce: Platform fundamentals: Object to relational mapping [Video]. Skillsoft Ireland.
- Lano, K. (2016). *Agile model-based development using UML-RSDS.* CRC Press.
- McEwen, A., & Cassimally, H. (2013). *Designing the internet of things.* John Wiley & Sons.
- McKay, E. N. (2013). *UI is communication: How to design intuitive, user-centered interfaces by focusing on effective communication.* Morgan Kaufmann Publishers.
- Ramnath, R., & Loffing, C. (2014). *Beginning iOS programming for dummies.* Hoboken, NJL John Wiley & Sons.
- Scheibmayr, S. (2014). *Graphical user interface prototyping for distributed requirements engineering.* Internationaler Verlag der Wissenschaften.
- Skillsoft. (n.d.). Beginning object-oriented design [Tutorial].
- Stephens, R. (2015). *Beginning software engineering.* John Wiley & Sons, Incorporated.
- Torres, A., Galante, R., Pimenta, M. S., & Martins, A. J. B. (2017). Twenty years of object-relational mapping: A survey on patterns, solutions, and their implications on application design. *Information and Software Technology*, *82*, 1–18.
- Van der Pijl, P., Lokitz, J., & Solomon, L. K. (2016). *Design a better business: New tools, skills, and mindset for strategy and innovation.* John Wiley & Sons, Incorporated.
- Zlobin, G. (2013). *Learning Python design patterns.* Packt Publishing.

External Resource

Please note that URLs change frequently. While the URLs were current when this course was designed, some may no longer be valid. If you cannot access a specific link, contact your instructor for an alternative URL. Permissions for the following links have been either granted or deemed appropriate for educational use at the time of course publication.

- Moqups. (n.d.). Moqups: Online mockups made simple. Retrieved from https://moqups.com/

Suggested

The following materials are recommended to provide you with a better understanding of the topics in this course. These materials are not required to complete the course, but they are aligned to course activities and assessments and are highly recommended for your use.

Optional

The following optional materials are offered to provide you with a better understanding of the topics in this course. These materials are not required to complete the course.

Library

The following optional Skillsoft resources are available via the Capella University Library.

- Calnan, C. (n.d.). Designing with flexibility and efficiency in mind [Tutorial]. Skillsoft.
- Campbell, J. (2016). Software development fundamentals: User interface design [Video]. Skillsoft Ireland.
- Easttom, C. (2017). Defensive programming: Potential UI application risks [Video]. Skillsoft Ireland.
- Keenan, C. (n.d.). Object-oriented design: UML use cases [Video]. Skillsoft Ireland.
- Sampson, A. (2013). Generic design and modeling databases: Concepts and conceptual design [Tutorial]. Skillsoft Ireland.

External Resource

Please note that URLs change frequently. While the URLs were current when this course was designed, some may no longer be valid. If you cannot access a specific link, contact your instructor for an alternative URL. Permissions for the following links have been either granted or deemed appropriate for educational use at the time of course publication.

- Wiegers, K. (2001). Humanizing Peer Reviews. Retrieved from http://www.processimpact.com/articles/humanizing_reviews.html
- Wiegers, K. (2001). When Two Eyes Aren't Enough. Retrieved from http://www.processimpact.com/articles/two_eyes.html

Projects

Project >> Software Design and Construction

Project Overview

Your course project takes a problem and case-based approach to software design and modeling by placing you in the role of a software designer. You will design, model, prototype, and review a selected subsystem of a software product by completing the following assignments:

- u01a1 – Class Responsibility Collaborator Cards – Structural Model.
- u02a1 – Sequence Diagrams – Behavioral Model.
- u03a1 – Class Diagram – Structural Model.
- u03a2 – Entity Relationship Diagram (ERD) – Structural Model.

- u04a1 – User Interface and Prototype.
- u05a1 – Deployment Diagram – Structural Model.
- u05a2 – Design Review.

Project Scenario

Your course project work will be based upon the CapraTek scenario first introduced in IT4711. Having access to the SRS you completed in that course will speed your initial work, but is not required as the scenario provides all the information that you need and will be available for review in this course.

CapraTek, a longtime leader in computer server technology, is planning to develop an iOS app to interact with its Alfred Smart Hub. Alfred is an integrated wireless smart-home system that seamlessly connects household electronics, appliances, and devices. The app will control a smart thermostat sold to the consumer market. Your job as the software designer is to design, model, and prototype the application.

Unit 1 >> Software Design Fundamentals

Introduction

The software design process plays critical early role within the overall software development life cycle. During the software design phase, software engineers create behavioral and structural models that act as blueprints for the developers.

These models allow us to analyze and evaluate the design to ensure that it fulfills the requirements from the organization. They can also be used to look at alternative solutions and the potential tradeoffs of particular

choices and can be used to plan development activities.

Activities in Software Design

Software design fits in between the requirements analysis and software construction phase. There generally are two activities that are part of the software design process:

1. **Software Architectural Design** – High level design that looks at the structure and organization and identifies the various classes.
2. **Software Detailed Design** – Specifies each of the classes in detail to help the process of creating them.

Design Strategies

There are many different strategies to help guide the overall design process. Each employs a different methodology. Some of them are:

- **Structured design** – a classic method of software design that is focused on identifying, elaborating on, and refining major software functions.
- **Object-oriented design** – uses a method based on well-designed objects. Inheritance and polymorphism played a key role in early versions.
- **Component-based design** – a popular trend in object-oriented design in which single software components or independent units have well-defined interfaces and dependencies. Component-based design helps improve re-usability of the code as each of the components can be deployed individually.

Modeling

During the modeling phase of the software design process, designers use diagrams to describe and display the results of the requirement analysis. During this stage, the software designer creates the overall picture of the system for the project team members, displaying how the business goals will be met. Modeling should display the functional, structural, and behavioral perspectives of a system in order to depict a complete and comprehensive picture of the final product.

For models to be useful, information must be gathered and analyzed from each of these main viewpoints, and direct relationships should be made to the overall business goals. Without a comprehensive model to work from, designs cannot be accurately created. These models can be built from UML use case diagrams, use case narratives, and/or product requirements provided by the stakeholders.

One of the initial steps to help define the software architectural design is to create Class Responsibility Collaborator cards.

Class Responsibility Collaborator Cards

Class Responsibility Collaborator cards (CRCs) are used to help define the names of product classes, their responsibilities, and collaborating class names. This information can be gleaned from use cases to help create the CRC cards. CRC cards help us to validate the class model based on use cases. It can be helpful to look

through all use cases and ancillary documentation to identify noun and verb phrases that identify classes and responsibilities.

Although CRC cards were originally used as a means to teach object-oriented concepts, they have been used more so now for the purpose of modeling as a whole. A class helps represent a collection of similar objects. These objects can be a person, place, thing or concept that is related to the software. For example, within the CapraTek product scenario, we may have *user* or *thermostat* as an example of a class. We would use singular names as each of the classes represents a generalized version of the object. Although we may have multiple "users" interacting in the application, we would model the class "User." The information about that "user" helps describe a single "user" rather than multiple "users."

Responsibilities

A responsibility is anything that a class show, know, or perform. For example, the user class may have a first name, last name, they may register, login, et cetera. Sometimes a class has responsibilities to be able to do but does not have enough information to do it. It is important to note that the class should only be able to change the values of things it knows but it is unable to change the values of what other classes may know.

For example, a user class should only be able to directly perform tasks based on itself and its attributes (change password, register, login, et cetera). Classes can only directly access the items that they contain but not directly access items that other classes contain. The user class is unable to change the temperature directly as the temperature is part of the Thermostat class. Because of this, it has to collaborate with the card named Thermostat to be able to change the temperature. Because of this, "Thermostat" would be on the list of the collaborators for "User."

Terms to Know:

- Software architecture.
- Software properties.
- Design notations.
- Components.
- Class Responsibility Collaborator Cards.
- Class Model.
- Use Cases.
- Noun and Verb phrases.
- Stress Test.
- Alternative and Exception Flow.
- Notation

Learning Activities

u01s1 - Studies

Readings

- Dennis, A., & Tegarden, D., & Wixom, B. (2015). *Systems analysis and design with UML* (5th ed.). Hoboken, NJ: John Wiley & Sons.
  - Chapter 1, "Introduction to Systems Analysis and Design," pages 19–35.
  - Chapter 4, "Business Process and Functional Modeling," pages 119–156.
  - Chapter 5, "Structural Modeling," pages 163–175, 185–186.
- Sommerville, I. (2016). *Software engineering* (10th ed.). Boston, MA: Pearson.
  - Chapter 16, "Component-Based Software Engineering," pages 425–451.

Library Resources

- Anderson, J., & Francheschi Jones, H. (2016). *Java illuminated: An active learning approach* (4th ed.). Burlington, MA: Jones and Bartlett Learning.
  - Read Chapter 3, "Object-Oriented Programming, Part 1: Using Classes."
    - In Object-Oriented Programming (OOP), it is typical for developers to use classes when developing software. The authors of this book explain the term class, and how it is used when programming in Java.
- Ramnath, R., & Loffing, C. (2014). *Beginning iOS programming for dummies*. Hoboken, NJ: John Wiley & Sons.
  - Read Chapter 2, "Object-Oriented Design Principles."
    - The author discusses how to recognize and use nouns and verbs from the materials. There are tips on how to include or eliminate nouns and verbs once they have been identified.
- Cusick. J. (2013). *Durable ideas in software engineering*. Oak Park, IL: Bentham Science Publishers.
  - Read pages 118–122.
    - The author explains why Class Responsibility Collaborator Cards are still in use in modern times. Other common verbiage such as Classes and Objects were explored.

Skillsoft Resources

- Go to the Beginning Object-Oriented Design tutorial.
  - Under the Object-Oriented Design Basics lesson, complete the following topic:
    - Class Responsibility Collaboration Card, (2:00).

Optional Media

- Click **CapraTek: Project Introduction** to complete the multimedia presentation.
- Click **CapraTek: Project Stakeholder and User Interviews** to complete the multimedia presentation.

u01s1 - Learning Components

- Explain the basic principles of what classes are and how they are used.

- Understand how to identify potential classes through the evaluation of noun and verb phrases.
- Study examples of CRC cards.
- Understand how to translate verb phrases to responsibilities.
- Explain CRC card notation including verbiage and structure
- Explain class limitations and boundaries.
- Define the relationships and inter-dependencies among classes.

u01s2 - Software Preparation and Technology Access

In this course, you will be using software and technology that is needed to complete designated activities and assignments. There is no additional cost for this software and technology. Some software packages will be made available to you at no additional cost through Capella's subscription with Microsoft, while other software packages are available for free download through open-source licensing.

Capella University requires learners to meet certain minimum computer requirements. Please note that some software required for a course may exceed these minimum requirements. Check the requirements for the software you may need to download and install to make sure it will work on your device. Most software will require a Windows PC. If you use a Mac, refer to Installing a Virtual Windows Environment.

The software and technologies below are strongly recommended to support you in completing the course objectives. If you have access to other tools that you believe may still meet course requirements or if you have any difficulties accessing this resource or completing the related assignments, please contact your course faculty member to discuss potential alternatives.

If you use assistive technology or any alternative communication methods to access course content, please contact DisabilityServices@Capella.edu with any access-related questions or to request accommodations.

For this course, follow the instructions provided through the links below to download and install software or register for an account, as required.

Microsoft Software

1. If you have a Capella MS Imagine account, go to Step 2. Otherwise, see the instructions for registering an account at MS Imagine – Registration.
2. Log into the Capella Microsoft Imagine WebStore.
3. Identify the version of MS Visio that is compatible with your operating system.
4. Download and install.

If you encounter any difficulties in the download and installation process, post a detailed question in the Ask Your Instructor section of the course. Your instructor should be able to help you or point you in the right direction for the answers you need.

u01a1 - Class Responsibility Collaborator Cards – Structural Model

Overview

One of the initial steps in the design process is often the creation of Class Responsibility Collaborator (CRC) cards. They are used to help define the names of the classes, their responsibilities, and collaborating class names. Class names can be taken from use cases to help create the CRC cards. CRC cards are used to help review or define tasks.

Note: A review of the CapraTek scenarios will assist you in identifying the noun phrases and verb phrases that help to identify classes and responsibilities.

Preparation

- View or review the following media pieces in the Resources:
  - CapraTek: Project Introduction.
  - CapraTek: User and Stakeholder Interviews.
- Download the CRC Card Template found in the Resources. Use it to complete this assignment.

Note: Having access to the SRS (in particular, the use cases that you wrote) that you completed in IT4711 will speed your work on this assignment; however, it is not required. All the information that you need can be found in the CapraTek scenarios.

Directions

Create CRC cards for all classes using the template provided. Make sure to do the following:

- Create class responsibility collaboration cards that reflect software requirements.
- Identify all system classes correctly based on product requirements.
- Identify all responsibilities of classes based on product requirements.
- Identify all collaborators for each class.

Course Resources
CRC Card Template [DOCX]

u01d1 - Design Strategies

It can be challenging to ensure that we identify all of the potential classes along with their relationships.

- What design strategies and methods could be used to help analyze the requirements to help identify the classes and their interconnections?
- What is the relevance of nouns and verbs when working with classes?

Response Guidelines

Comment on the post of at least two other learners. Offer insights, solutions, examples, or opinions that add depth and value to the conversation.

Discussions in This Course

The content topic should determine the length of your post; however, a minimum of 150 words is recommended. Refer to the Discussion Participation Scoring Guide for posting expectations. **Make your initial posts by Wednesday** to allow sufficient time for peers to respond. The expectation within the course discussions is to respond to at least two posts by the end of the unit, but it is highly recommended that you extend the dialog further. Responding over multiple days will help stimulate a lively discussion.

Course Resources
Undergraduate Discussion Participation Scoring Guide

u01d1 - Learning Components

- Explain the basic principles of what classes are and how they are used.
- Understand how to identify potential classes through the evaluation of noun and verb phrases.
- Understand how to translate verb phrases to responsibilities.

Unit 2 >> Software Design Behavioral Description

Introduction

Behavioral descriptions or dynamic views help describe the dynamic behavior of the software. There are many different notations that are useful for describing the functionality or behavior of various aspects of the software. Some of these include:

- **Activity diagrams** show control flow between one activity to another. They can be used to help represent concurrent activities.

- **Communication diagrams** show the interactions between a group of objects. The focus is on the objects, how they are linked and the messages that they exchange with one another using those links.
- **Data flow diagrams** are used to show the data flow among operational elements. They help identify possible paths for any attacks or disclosure of confidential information.
- **Decision tables and diagrams** are used to represent complex combinations of conditions and actions.
- **Flowcharts** represent the flow of control and the actions that will need to be performed.
- **Sequence diagrams** show the interaction of a group of objects. There is the focus to show the time ordering of the messages that are passed between the objects to show the interaction timing constraints.
- **State chart diagrams** show the control flow from state to state and how the behavior of a component changes based on its current state.
- **Pseudocode** describes functionality using a structured programming-like language.

Sequence Diagrams

Sequence diagrams are a type of interaction diagram that enables designers and developers to be able to look at the behavior of several objects within a single use case narrative or diagram. They work well to show collaborations between objects but they do not provide precise behavior of what should occur.

In order to create sequence diagrams, developers need to identify the behavior or use case to explore. Then they need to identify the structural elements by modeling the components/objects, lifelines, activations, messages, and timing constraints. They help capture how objects in the context of collaboration interact with one another, and show how objects play the role defined in a collaboration.

The sequence diagrams also show the order of the interaction by using the vertical axis of the diagram to represent time of what messages are sent and when they are sent. By seeing how these elements interact over time, we are able to derive the interaction between them and visually see a specific series of events that occur based on a use case.

The vertical axis within a sequence diagram represents the time progressing down the page. The focus of the items and interactions are purely based on ordering rather than about the duration of how long each interaction takes. The horizontal axis shows the elements/objects that are involved within that interaction. Typically, the order of the objects involved in the operation are shown from left to right based on when they take part in the message sequence.

Terms to Know

- Components/objects.
- Lifelines.
- Activations.
- Messages.
- Timing constraints/interactions.
- Decision tables.
- Pseudocodes.
- Diagrams:
    - Data flow.

- Activity.
- Communication.
- Sequence.
- State chart.

Learning Activities

u02s1 - Studies

Readings

- Dennis, A., & Tegarden, D., & Wixom, B. (2015). *Systems analysis and design with UML* (5th ed.). Hoboken, NJ: John Wiley & Sons.
  - Chapter 6, "Behavioral Modeling," pages 202–234.
- Sommerville, I. (2016). *Software engineering* (10th ed.). Boston, MA: Pearson.
  - Chapter 5, "System Modeling,"
    - Pages 118–146.
    - Pages 176–184.
    - Pages 187–196.
  - Chapter 8, "Class and Method Design," pages 280–321.

Library Resources

- Barrera, D. G., & Diaz, M. (2013). *Communicating systems with UML 2*. Hoboken, NJ: John Wiley & Sons.
  - Read the Sequence Diagrams on 2-2c.
    - The authors analyzed communication protocols of a network using UML 2 diagrams. A simple sequence diagram was used to show the behavior of the protocol communication flow between the client and the server.
- Gomaa, H. (2016). *Real-time software designing or embedded systems*. New York, NY: Cambridge University Press.
  - Read Chapter 2, "Overview of UML, SysML, and MART."
    - Sequence diagrams show how objects communicate with other objects, and the messages that trigger the communications. The author of this book provides a good explanation of the concept as well as some examples.
- Stephens, R. (2015). *Beginning software engineering*. Indianapolis, IN: John Wiley & Sons.
  - Read pages 105–116.
    - The author introduces developers to building robust and dependable software. There are several diagrams that supplements the concepts presented in this section.

Optional Skillsoft Resource

- Keenan, C. (n.d.). Object-oriented design: UML use cases [Video]. Skillsoft Ireland.

u02s1 - Learning Components

- Understand how to translate CRC cards to sequence diagrams.
- Review examples of UML Sequence diagrams.
- Understand UML formatting and notation for sequence diagrams.
- Describe the logical flow between classes for a series of interactions.
- Understanding timing interactions and messaging between objects.

u02a1 - Sequence Diagrams – Behavioral Model

Overview

Now that we have completed our CRC cards, it is time to define the behavioral models. One such model is a sequence diagram. Your sequence diagrams should illustrate the collaboration between classes by defining the specific interaction among them based on messages and timing interactions.

Preparation

- Review the CRC cards from your last assignment.
- In this assignment, you will create a diagram. You may download Microsoft Visio for free using the instructions found in the study in Unit 1, or use any program you choose to create your sequence diagram.

Directions

Choose two of the following application scenarios and create a sequence diagram for each.

- User registration (new account).
- User interaction with thermostat.
- User login.

Make sure to do the following:

- Create a UML sequence diagram that completely demonstrates a scenario.
- Create a UML sequence diagram using proper UML notation.
- Identify all messages and interactions within the UML sequence diagrams.

u02d1 - UML Diagrams

As the software design and development processes evolve, so do the modeling tools and diagrams. UML is one of the most common ways to document a system. Although there are many different UML diagrams, not all of them are consistently used for all projects. For this discussion:

- Identify two of the most common UML diagrams and provide an analysis of each of them.
- Explain who the audience for each of the diagrams would be within the organization.

Response Guidelines

Comment on the post of at least two other learners. Offer insights, solutions, examples, or opinions that add depth and value to the conversation.

Course Resources
Undergraduate Discussion Participation Scoring Guide

u02d1 - Learning Components

- Understand how to translate CRC cards to sequence diagrams.
- Review examples of UML Sequence diagrams.
- Understand UML formatting and notation for sequence diagrams.

Unit 3 >> Software Structure and Architecture

Introduction

Software Architecture

Software architecture defines a set of classes and the relationships among them along with their properties. Software structures and architectures have been focused around creating software designs in a more generic way with varying levels of abstraction. These concepts are in place to help describe and reuse the design aspects.

There are different high-level facets of the software design that are often called views. A view represents a part of the software architecture that helps illustrate specific properties of the overall software. They attempt to address issues within software design, including:

- Ensuring that the design satisfies the functional requirements (logical view).
- Ensuring that we address concurrency issues and interactions between classes (process view).
- Ensuring that we consider the distribution issues (physical view).

- Looking at how the design is broken down into classes and the relationship of those classes (development view).

Some of models and notations that describe the structural aspect of a software design include:

- **Architecture description language** – These are textual formal languages that help describe the software architecture using components and connectors.
- **Class and object diagrams –** These represent a set of classes and show how they are interrelated.
- **Component diagrams** – These are used to show the physical and replaceable parts of a system that shows their relationships with one another.
- **Class responsibility collaborator cards** – These note the names of the classes, their responsibilities, and their collaborating classes.
- **Deployment diagrams** – These are used to represent the physical nodes and helps model the physical aspects of the software.
- **Entity relationship diagrams** – These are used to represent the conceptual models of the data stored in the databases.
- **Structure charts** – These show the calling structure of programs which shows which modules call other models.

Entity Relationship Diagrams (ERDs)

Along with the knowledge of an application's architecture, the software designer must understand how data is stored and managed within an environment. There will undoubtedly be data storage and management requirements that the new system must effectively handle in order to meet the overall business goals of the project. Therefore, to ensure the system's success, the software designer must be able to communicate to the organization's database professionals the data requirements of the system and the stakeholders. ERDs are designed to accomplish this task.

Class Diagrams

Once a comprehensive model is in place, software engineers can begin to transition from software architecture modeling to software detailed design. Where software architecture primarily describes how the system will meet the business goals, detailed designing primarily focuses on how the system will operate. The purpose of the software detailed design is to describe a system that operates in a way that meets the functional, structural, and behavioral needs of a group of users within an environment working toward accomplishing business goals as it is described within the system models.

Several steps must be taken before designs can be created from developed models. For example, because the design relies heavily on the model, it is critical that the model's accuracy and focus is validated and verified before a design can be conceptualized.

In order to create an effective design and to effectively communicate stakeholder needs to programmers and application developers, a software designer must have the ability to identify the potential programming constructs and application components from within a business-oriented system model. This skill requires a

strong understanding of the application architecture and the knowledge of how program language classes and methods are created and combined to accomplish software functionality.

The static structure of the design is summarized in an overall class diagram and then the dynamic aspects of the design are developed from there. The detailed design is developed by detailing the attributes and methods for each class in the class diagram. Much of the detail can be taken from the CRC cards and prior models that have been created.

Learning Activities

u03s1 - Studies

Reading

- Sommerville, I. (2016). *Software engineering* (10th ed.). Boston, MA: Pearson.
  - Chapter 7, "Design and Implementation," pages 176–204.
- Dennis, A., & Tegarden, D., & Wixom, B. (2015). *Systems analysis and design with UML* (5th ed.). Hoboken, NJ: John Wiley & Sons.
  - Chapter 9, "Data Management Layer Design," pages 326–361.

Library Resources

- Lano, K. (2016). *Agile model-based development using UML-RSDS*. Boca Raton, FL: CRC Press.
  - Read the chapter "Class Diagrams."
    - The author illustrates the use of UML in development through use-cases and class diagram examples.
    - *Please note that this ebook has a single user license. Please be considerate of other learners and limit your time viewing the book, or download the relevant materials for off-line reading.*
- Torres, A., Galante, R., Pimenta, M. S., & Martins, A. J. B. (2017). Twenty years of object-relational mapping: A survey on patterns, solutions, and their implications on application design. *Information and Software Technology*, *82*, 1–18.
  - The authors discuss the use of Object Relational Mapping tools, and how they help developers to tackle mismatch issues. In this write-up, a survey of nine object relational mapping solutions were reviewed.
- Harrington, J. L. (2016). *Relational database design and implementation* (4th ed.). Cambridge, MA: Morgan Kaufmann.
  - Read "Tables," pages 197–212.
    - Tables play a very important role when working with relational databases. The author explains a table structure, and provides examples of how they are used.

- Hendry, B. (2017). Oracle Commerce: Platform fundamentals: Object to relational mapping [Video]. Skillsoft Ireland.
- Giesenow, H. (2017). Microsoft Entity Framework: Understanding the EF modeling languages [Video]. Skillsoft Ireland.
- Giesenow, H. (2017). Microsoft Entity Framework: What is the Entity Framework [Video]. Skillsoft Ireland.

Optional Skillsoft Tutorials

- Go to the Beginning Object-Oriented Design tutorial.
  - Under the Object-Oriented Programming Basics lesson, complete the following topic:
    - Implementing Inheritance, (6:00).
- Go to the Generic Design and Modeling Databases: Concepts and Conceptual Design tutorial.
  - Under the Conceptual Design lesson, complete the following topic:
    - Overview of Creating a Conceptual Design, (3:00).
    - Overview of Entities, Attributes, and Relationships, (6:00).
    - Creating an Entity Relationship Diagram, (10:00).
- Calnan, C. (n.d.). Designing with flexibility and efficiency in mind [Tutorial]. Skillsoft.

u03s1 - Learning Components

- Study examples of UML class diagrams.
- Understand the principles of association, aggregation, inheritance and composition.
- Understand concepts behind classes, relationships, attributes and methods.
- Apply the process of object-relational mapping.
- Identify attributes and methods from the responsibilities and collaborations from CRC cards.
- Understand how constraints, cardinality and relationships tables are used and applied to a data model.
- Understand UML class diagram notation.
- Study examples of UML class diagrams.
- Review concepts behind classes, relationships, attributes and methods.
- See examples of how ERDs and class diagrams are mapped.
- Review concepts behind tables and their relationships, data types, constraints, and attributes.
- Review the concepts behind database normalization.
- Study examples of ERD notation.
- Study examples of object relational mapping of databases.
- Describe basic data modeling principles.

u03a1 - Class Diagram – Structural Model

Overview

It is time to expand your structural model by using your CRC cards to create a class diagram. Your class diagrams should effectively show the relationships among classes, attributes, and methods. There are four primary types of relationships between classes. These include association, aggregation, composition, and inheritance.

- Association simply states that there is some kind of link or dependency between the two classes. Association is the most abstract method to describe the relationship between two classes.
- Aggregation is when there is a part of a relationship that a particular class is a part of a larger set of class(es). Aggregation shows that the classes can exist on their own.
- Composition is similar to aggregation, but in composition one class is fully dependent upon the related class. The child class cannot exist without the existence of the parent class.
- Inheritance is a very common type of relationship between classes. This allows us to define that one class is a specialized type of another class. This means that the child class will inherit all of the methods and attributes of the parent class.

Preparation

In this assignment you will create a diagram. You may download Microsoft Visio for free using the instructions found in the study in Unit 1, or use any program you choose to create your diagram.

Directions

Use your CRC cards to create a class diagram. Make sure to do the following:

- Create a UML class diagram that accurately integrates a set of classes using appropriate relationship types.
- Create a UML class diagram that incorporates all classes, responsibilities, and collaborators from CRC cards.
- Create a class diagram that is correctly formatted using proper UML notation.

Additional Requirements

Save your class diagram file as "class diagram" and entity relationship diagram as "ERD" and submit it in the courseroom. If you use a program other than Visio, cut and paste both diagrams into a single Word document for submission.

u03a2 - Entity Relationship Diagram (ERD) – Structural Model

Overview

You will now expand the structural model using your class diagram to create the Entity Relationship Diagram (ERD). At times, the database design will directly mimic the class structure when we look at the attributes within the various classes. Instead of classes in a class diagram, ERDs use tables. The conversion from class diagrams to ERD is called object-relational mapping. A common theme in conversion is that we will have a one-class-to-one-table mapping. The attributes of that class can then be mapped to the columns within the mapped table. The naming scheme of the columns for a table will be different than the attribute names in the classes due to standards.

The ERD will also indicate primary and foreign keys where a class diagram may not. The primary keys help uniquely identify a particular row within a table. The relationships between those rows are maintained through the use of foreign keys. The data types between the two diagrams can also be different depending on the underlying relational database management system.

Preparation

- Review your class diagram from your previous assignment.
- In this assignment, you will create diagrams. You may download Microsoft Visio for free using the instructions found in the study in Unit 1, or use any program you choose to create your diagram.

Directions

Create an ERD that describes the database architecture based upon your class diagram.

Note: Make sure to identify the relational database management system (RDBMS) in use.

For this assignment:

- Create an ERD that accurately depicts a database structure that is based on your class diagram.
- Use accurate ERD formatting and notation.
- Design the database model based upon the class diagram using object-relational mapping.
- Create a data model that accurately incorporates relationships, primary keys, foreign keys, cardinality, and data types.

Additional Requirements

Save your file as "ERD" and submit it in the courseroom. If you use a program other than Visio, cut and paste the diagram into a Word document for submission.

Object models and relational models do not always work very well together. Databases represent data in a tabular form, whereas object-oriented languages represent data based on a graph of objects. There are a few common mismatch problems that can exist with association, aggregation, inheritance, and composition and data navigation.

Given these potential problems, how you would use object-relational mapping to address at least two of these problems?

Response Guidelines

Comment on the posts of at least two other learners. Offer insights, solutions, examples, or opinions that add depth and value to the conversation.

Course Resources
Undergraduate Discussion Participation Scoring Guide

u03d1 - Learning Components

- Understand the principles of association, aggregation, inheritance and composition.
- Understand concepts behind classes, relationships, attributes and methods.
- Apply the process of object-relational mapping.
- Understand how constraints, cardinality and relationships tables are used and applied to a data model.

Unit 4 >> User Interface Design

Introduction

User interface (UI) design is an essential part of the software design process. It is the front-end software view that users interact with to manipulate and control the software and hardware. The UI should be designed primarily with the end user in mind. It should be aesthetically pleasing, simple to use, responsive, intuitive, and consistently displayed.

A good way to create user interfaces is to look at completed examples of user interfaces and model your user interfaces after those. The target audience is important to consider as the design can change based on the skill of the user. Users are typically involved in reviewing the prototyped user interfaces. If our end users are technically savvy, we can have an advanced user interface. However, if we are designing for casual or novice users, more information and simple designs are warranted.

It is important that the UI design show how the user should interact with the software as well as how the information from the software should be presented to the users.

Learning Activities

u04s1 - Studies

Readings

Use your *Systems Analysis and Design: An Object-Oriented Approach With UML* text to read the following:

- Chapter 7, "Moving on to Design," pages 240–275.
- Chapter 10, "Human Computer Interaction Layer Design," pages 367–410.

Library Resources

- McEwen, A., & Cassimally, H. (2013). *Designing the internet of things*. John Wiley & Sons.
  - Read pages 63–88.
    - The authors present a good discussion on the benefit of prototypes when designing. The discussion includes case studies and illustrated examples.
    - *Please note that this ebook has a single user license. Please be considerate of other learners and limit your time viewing the book, or download the relevant materials for off-line reading.*
- Scheibmayr, S. (2014). *Graphical user interface prototyping for distributed requirements engineering*. Frankfurt am Main, Germany: Peter Lang GmbH.
  - Read pages 4–8.
  - The author discusses the use of Graphical User Interface (GUI) prototyping as an effective tool which can quickly convey the concepts between the developer, and the client.
- Van Der Pijl, P., Lokitz, J., Solomon, L. K.. van Leishout, M. (2016). *Design a better business: Newtools, skills, and mindset for strategy and innovation*. Hoboken, NJ: John Wiley & Sons.
  - Read pages 155–181.
    - The authors discussed the importance of prototyping, and emphasized that prototyping provides ideas, form, and function.
    - *Please note that this ebook has a single user license. Please be considerate of other learners and limit your time viewing the book, or download the relevant materials for off-line reading.*
- McKay, E. N. (2013). *UI is communication: Howto design intuitive, user-centered interfaces by focusing on effective communication*. Cambridge, MA: Morgan Kaufmann Publishers.
  - Read Chapter 6, "UI Design Examples."
    - This chapter provides a great example on how to proceed when designing the user interface. Includes step-by-step exercises with prompts to stimulate the thinking process of creating the user interface.

Internet Reading

Review the following Web sites for information on prototyping tools:

- Moqups. (n.d.). Moqups: Online mockups made simple. Retrieved from https://moqups.com/

Optional Skillsoft Resources

- Campbell, J. (2016). Software development fundamentals: User interface design [Video]. Skillsoft Ireland Limited.
  - Running time: 6 minutes.
- Easttom, C. (2015). Defensive programming: Potential UI application risks [Video]. Skillsoft Ireland Limited.
  - Running time: 3 minutes.
- Calnan, C. (n.d.). Designing with flexibility and efficiency in mind [Tutorial]. Skillsoft.

u04s1 - Learning Components

- Understand end user requirements.
- Review examples of user interfaces.
- Review user interface patterns of common solutions to UI design.
- Understand how to create a prototype.
- Explain how to translate UML structural model diagrams to user interfaces and prototypes.
- Understand concepts for creating effective navigation.
- Understand sequence diagrams.
- Review examples of user interface design evaluation.

u04s2 - Assignment Preparation

In preparation for this week's assignment, go to moqups.com and create a user account. Once logged in, you can begin using their wire framing and prototyping tool to complete this assignment. You should be able to create your prototype without having to write any code.

u04a1 - User Interface and Prototype

Overview

At this point, we have a good understanding of the various behavior and structural models that describe the software design. Using these we can now create the UI and its prototype to help evaluate and validate our models.

Ensure your prototype includes the functionality that is defined in our CRC cards within the responsibilities of each class. Your prototype should be based on your ERD and class diagrams, and include any user-facing attributes as part of any form data input field; such as first name, last name, username, password. This is important as the data needs to be captured at some point within the prototype. As you develop the prototype, it is important that each of the screen designs should be as consistent and user friendly as possible.

Directions

Do the following:

Part 1: Create a Prototype

Create the Application UI so it comprehensively reflects your behavioral and structural models using the moqups prototyping tool. The prototype should follow functional and design best practice principles.

Make sure to do the following:

- Design a functioning prototype that is appropriate for the target end user.
- Create a user interface that accurately reflects your structural models including your CRC cards, class diagram, and ERD.
- Create a UI that validates the application's models.
- Create effective prototype navigation based on a sequence diagrams.

Part 2: UI Evaluation

- Write a detailed UI evaluation that provides appropriate rationale for its design.

Additional Requirements

Submit a Word document with your UI evaluation and the URL for your prototype.

Note: Creating your prototype on moqups.com will generate a URL for your prototype.

The creation of prototypes can be quite helpful for stakeholders and software designers to help verify and validate the various models and requirements.

Research and find at least 3 examples of particular UI patterns that could be applied to your own prototype and share them with the class.

- What makes them effective UI designs?
- Discuss their appropriateness for application to the course project.

Response Guidelines

Comment on the post of at least two other learners. Do you agree with your peer's assessment of the patterns? Offer insights, solutions, examples, or opinions that add depth and value to the conversation.

Course Resources
Undergraduate Discussion Participation Scoring Guide

u04d1 - Learning Components

- Understand end user requirements.
- Review user interface patterns of common solutions to UI design.

Unit 5 >> Software Design Quality Analysis and Evaluation

Introduction

Once the software architecture has been defined and the prototype has been created, it is important to review all of the structural and behavioral models and UI design to measure the quality of the design. There are many different qualities that must be measured, including maintainability, portability, testability, usability, correctness, and robustness. There are many techniques used to analyze and evaluate software design quality. These include:

- **Software design reviews** – Done to determine the quality of the design and to review the architecture, design requirements tracing, as well as to evaluate security.
- **Static analysis** – Specific types of analysis are used to test design vulnerability if security is an issue. Mathematical models help designers predict the behavior of the software and validate its performance.
- **Simulation and prototyping** – Once a prototype is created, the prototype is reviewed to ensure that it fully reflects all of the structural and behavioral models.

Simulation and prototyping can be useful to look at UI patterns. UI patterns help designers create consistent solutions based on common design problems. Most of us know what a login form looks like. Creating a login

form that does not follow the standard UI pattern forces the end user to have to learn how to use your user interface. Given a particular problem, there can be potential design and UI patterns to solve it. When possible, and if it makes sense, designers should use those design and UI patterns as part of their design and prototypes. A particular design can make use of these design and UI patterns.

Deployment Diagrams

Deployment diagrams describe, from a high-level, what is deployed and where as well as how the software is going to interact and integrate with existing systems. They show system hardware, the software that is installed on that hardware, and any middleware that is used to connect machines. Deployment diagrams should be created when software is deployed to multiple machines. They help to indicate what is needed to build the software and where each of its components exist.

Design Review

Design reviews generally involve a presentation of the planned software design to the stakeholders. Its purpose is to ensure that the proposed design meets all of the requirements from stakeholders and users. They can also be used to ensure that the software design can be implemented in a timely fashion, that interfacing with other systems is fully considered, and that it follows acceptable software design principles.

Design reviews are useful to ensure that all individuals involved in the software development process are on the same page and understand how the software needs to be constructed. They can also help identify issues with the design or provide some alternative approaches to the design. For example, there may be design patterns that could be effectively applied to create some consistency in the design.

Learning Activities

u05s1 - Studies

Readings

- Sommerville, I. (2016). *Software engineering* (10th ed.). Boston, MA: Pearson.
  - Chapter 7, "Design and Implementation," pages 176–204.
  - Chapter 24, "Quality Management," pages 664–665.
- Dennis, A., & Tegarden, D., & Wixom, B. (2015). *Systems analysis and design with UML* (5th ed.). Hoboken, NJ: John Wiley & Sons.
  - Chapter 11, "Physical Architecture Layer Design," pages 418–425, 432–447.

Library Resources

- Zlobin, G. (2013). *Learning Python design patterns*. Birmingham, UK: Packt Publishing

- The author uses a tutorial-based approach to introduce developers to various Python design patterns.

Optional Internet Resources

- Wiegers, K. (2001). Humanizing peer reviews. Retrieved from http://www.processimpact.com/articles/humanizing_reviews.html
  - This article addresses some of the social and cultural aspects of peer reviews, including ways to overcome resistance to the process, benefits that reviews provide for various team members, the role of management, and 10 signs of management commitment to the review process.
- Wiegers, K. (2001). When two eyes aren't enough. Retrieved from http://www.processimpact.com/articles/two_eyes.html
  - Several different types of activities called "peer reviews" are described here, including inspections, team reviews, walkthroughs, pair programming, peer deskchecks, and passarounds. A table suggests which types of peer reviews are best suited for achieving specific review objectives.

Optional Media

Skillsoft Resources (Review)

- Go to the Generic Design and Modeling Databases: Concepts and Conceptual Design tutorial.
  - Under the Conceptual Design lesson, complete the following topics:.
    - Overview of Creating a Conceptual Design (3:00).
    - Overview of Entities, Attributes, and Relationships (6:00).
    - Creating an Entity Relationship Diagram (10:00).

u05s1 - Learning Components

- Identify key system nodes that need to be mapped.
- Explain how to translate UML structural model diagrams to user interfaces and prototypes.
- Study a design review that has sequence diagrams and prototypes.
- Study examples of deployment diagram UML.
- Explain UI or design patterns.
- Understand the deployment related aspects of application architecture.
- Identify communication nodes.
- Study examples of software communication flow.
- Review the CapraTek scenario to understand users and software connection points.
- Study examples of ERD and class object mapping.
- Review examples of ERD and identify system generated values vs user generated data.
- See examples of design patterns that are applied to a prototype.
- Study a quality design review

u05d1 - Peer Design Review Groups

Note: For this assignment, your instructor will put you into groups of 3 or 4 learners. You will use files shared in this discussion to complete the u05a2 assignment.

Zip up the following files or documents from your previous assignments and attach the zipped file in a reply to the team post. Include a link to your prototype in your reply.

- Class diagram.
- CRC cards.
- ERD.
- Sequence diagram.

Choose one of your teammate's designs to review. Your review will be completed using PowerPoint and will be submitted as your u05a2 assignment. Your design review should be conducted from the point of view of a software designer that was not an active participant in the project.

Response Guidelines

Make sure to post your documents by Tuesday evening to assure your teammates have time to dedicate to a proper review.

Course Resources
Undergraduate Discussion Participation Scoring Guide

u05a1 - Deployment Diagram – Structural Model

Overview

Once we have completed the design and quality analysis phase of our product design, it is time to consider the deployment diagram. Your diagram should include middleware components, such as operating systems, as well as the software and hardware with which users will directly interact. Also, the iOS app is connected to the existing Alfred! system, so it is important to identify what and who will connect to and interact with the application.

Your deployment diagram should also illustrate the ways that the communication exists between each node. Then they include the nodes to the diagram such as database servers, mobile devices, et cetera. Once the nodes are defined, communications associations are added to the diagram.

Preparation

- Review the CapraTek scenario and its ancillary documents to review any physical system requirements. This will help provide you with a high-level view of what the nodes will be within the application architecture.
- In this assignment, you will create a diagram. You may download Microsoft Visio for free using the instructions found in the study in Unit 1, or use any program you choose to create your diagram.

Directions

Create a UML Deployment Diagram that illustrates communication flow among nodes.

Do the following:

- Create a UML deployment diagram that accurately depicts the planned deployment of the application architecture.
- Use correct UML deployment diagram notation.
- Illustrate all communication flow among nodes.
- Describe who and what will connect to and interact with the application.

u05a2 - Design Review

Overview

Now that your user interface and prototype is complete, it must be evaluated and validated with the models and prototype to ensure that all of the features and requirements have been addressed.

You designed and developed various models and a prototype based on the requirements of the organization. It is common to have other software designers and other stakeholders review the models and prototypes created to ensure that all of the requirements have been met and prototypes reflect the various models created.

Preparation

For this assignment, your instructor has put you into teams of 3 or 4 in u05d1.

Directions

Conduct a design review of your chosen design from u05d1. Do so from a software designer's point of view (one who has not been involved with the design process thus far). Your role is to evaluate and validate the design. Do the following:

Create an 8–10-slide design review PowerPoint presentation for product stakeholders. Make sure to include detailed speakers notes that completely explain concepts in each of the slides. Your presentation should detail your assessment of the software design so that it is easily understood by product stakeholders. Make sure to do the following:

- Compare responsibilities on the CRC cards with those executed in the prototype.
- Compare class diagrams to the ERD to validate the mapped objects.
- Compare the ERD to the prototype to ensure that all user-facing data is incorporated in the prototype.
- Compare the interactions in the sequence diagram with those in the prototype.
- Explain one UI and one design pattern that you would apply to the models or prototype that would improve the design.
- Evaluate the overall quality of the design models and prototype.

u05d2 - Design and UI Patterns

Design and UI patterns directly help to create simple consistent solutions for common problems. With design patterns, there are common creational, structural, and behavioral design patterns. Of these design patterns, which would you apply to your models and prototype? Explain your choice and describe how it would be used.

Response Guidelines

Comment on the post of at least two other learners. Offer insights, solutions, examples, or opinions that add depth and value to the conversation.

Course Resources
Undergraduate Discussion Participation Scoring Guide

u05d2 - Learning Components

- Explain how to translate UML structural model diagrams to user interfaces and prototypes.
- Study a design review that has sequence diagrams and prototypes.
- Explain UI or design patterns.