5 You can think of developing software as a process that takes a problem and produces a solution involving software. Normally, it's an iterative process, with each iteration taking a set of requirements through to a working and tested implementation and eventually building up a complete solution. Many developers are eager to go straight into coding despite not fully understanding what to program in the first place. Evidence suggest that diving straight into implementation work is a leading cause of project failure. In a survey from The Standish Group, the most common causes of project failures are related to issues in requirements and design. For example, about 13% of respondents noted incomplete requirements impaired their projects. Unless you want your projects to fail, take your time to form requirements and create a design. You might not get them perfect, but their importance to effectively making good software should not be overlooked. Throughout this module, you will see the importance of requirements and design for a successful software solution. We will cover how eliciting requirements involves actively probing a client's vision, by asking questions about issues that the client may not have considered. Besides identifying specific needs, you learn to ask about potential trade offs the client will need to make in the solution. With a clear idea of what you are trying to accomplish, you can pivot to Conceptual Design mock ups and eventually, Technical Design diagrams. By the end of this lesson, you will understand that design work involves outlining a solution. And this work may include evaluating different alternatives.

1:40 You may be eager to tackle implementation work and get something working, but the requirements and design activities are critical. Once you begin coding a solution and depend on certain assumptions, it can become difficult to change those assumptions. For the design phase, you will have to think like an architect, which means thinking about the structure and behavior of your software. Consider the following scenario. You're hired to design a house. Before you start laying the foundation, you must first understand what the homeowner wants. This starting point is known as eliciting requirements. The homeowner wants a single story house. It needs to have a gym, a bathroom, three bedrooms and a living room. Eliciting requirements involves not only listening to what the client is telling you, but asking questions to clarify what the client has not told you. For instance, did it strike you as odd that this house has no kitchen? That would be a natural follow up question. Do you anticipate needing a kitchen? Should the rooms all be same size? If not, which should be bigger or smaller? How big should the house be overall? Are there external design constraints? For example, building restrictions put in place by the community? Should the house face a particular direction to take advantage of passive solar energy or scenic views? Which rooms should be furthest apart? Which rooms should be close together? The art of eliciting requirements is found in asking revealing follow up questions. Once these questions are answered, you now have an initial set of requirements allowing you to start thinking of possible designs. The design activity involves taking requirements and outlining a solution. This activity involves producing a conceptual design and then a technical design, which results in two corresponding kinds of artifacts, conceptual mockups and technical diagrams.

3:14 Conceptual mockups provide your initial thoughts for how the requirements will be satisfied. At this point, you focus on the house design by identifying major components and connections and defer the technical details. For example, the components from your house project are, the lot on which the house will be situated, the house itself, the kitchen, the gym, the bathroom, the bedrooms and the living room. Connections in the house can relate the components. For example if the living room is openly accessible from the kitchen, the living room has a connection to the kitchen. Each component has a task it needs to perform, known as responsibility. For instance, the gym's responsibility is to provide the homeowner with space and power for fitness activities and equipment. Similarly, the kitchen's main responsibility is to provide space for storing kitchenware, appliances, food supplies, and power and water for meal preparation. As a main component, the house has the overall responsibility of providing enough power, water, and support for all the required components within it. Note, how we don't mention specifics about wiring and plumbing. These are technical details that cannot be fully addressed until the conceptual mockups are completely understood. For instance, determining the size of the electrical distribution panel for the house will require adding up the power requirements necessary to energize each of the rooms. I recommend finishing the conceptual design before moving on to forming the technical design. The clearer your conceptual design is, the better your technical designs will be.

4:35 Continuing with the architectural example, you've wowed the homeowner with your conceptual design and, together, now have a shared vision for the dream home that will now be built. After the conceptual mockups are done, it is time to define the technical details of the solution. From the conceptual design, you know all the major components and connections and their associated responsibilities. Describing how these responsibilities are met is the goal of technical design. In a technical design, you start specifying the technical details of each component. This is done by splitting components into smaller and smaller

components that are specific enough to be designed in detail. For example, the gym component will require further components like a floor. The floor will be responsible for supporting a lot of weight. Are homeowner is training to be an Olympic lifter. By breaking down components more and more into further components, each with specific responsibilities, you get down to a level where you can do a detailed design of a particular component, such as, describing how to reinforce the floor. Technical diagrams express how to address specific issues like this. Compromises might arise when creating an acceptable solution. What if reinforcing the floor of the gym requires putting in columns or beams in the basement below the gym? What if the homeowner also wanted a wide open space in the basement with good head room? Sometimes conflicts like this can happen. You and the homeowner will need to workout a compromise in the solution. Constant communication and feedback is key, so that the solution is acceptable. If components and connections and their responsibilities in your conceptual design prove impossible to achieve in the technical design. Or fail to meet the requirements, you will need to go back to your conceptual design and rework it. Once you come to a feasible design, you want to continuously check with your client that the conceptual mockups capture what they want. In the architectural example, such checks are important. Because you'd rather adjust the design on paper than demolish an actual wall later. The technical diagrams then become the basis for constructing the intended solution.

6:26 Let's apply what we have seen as a building architect to software design. Suppose you have a design task for a university course search website with the following requirement. As a learner, I want to search for relevant courses through a search page. Now, let's do a conceptual design.

6:43 In making a conceptual design of a building, we try to recognize appropriate components, connections and responsibilities and avoid technical details. An architect starts with a sketch of the building with the components, connections and responsibilities in mind. When it comes to conceptual design and software involving user interfaces, conceptual mockups can be a hand drawn sketch or a drawing made using computer tools. When we look at our requirement, as a learner, I want to search for relevant courses through a search page. We recognize search page and course as the components, and the search page has the responsibility of searching for relevant courses. By sketching a mockup of our user interface, we notice many missing components.

7:21 You're probably wondering about how a search keyword is entered in the search page. How is the search started? How is the list of search results displayed? These flaws in the initial mockup require further clarification with your client, or more conceptual design work. Eventually, we generate a more comprehensive conceptual design or user interface mockup.

7:43 The search page contains an input field and search button, and transitions to the result page. Course is a way of displaying the result. From this mock up, we recognize many connections. For example, for the search page to fulfill its responsibility to search, it needs, Input Field, Search button, and Results Page. This also translates to Search Page having connections to Input Field, Search button, and Results Page.

8:06 From a conceptual design, we move to making a technical design where, just like building design, you try to add a detail how those components, connections and responsibilities can be implemented. For example, we refine each component until it is specific enough to be designed in detail. For example, how does the search page fulfill its responsibility of searching a list of courses for relevant ones, given that a user has entered a keyword? Does the page need to talk to an external system? Suppose the university already has a Course Database component which your course Search Page can connect to. Since Search Page requires Course Database in order to fulfill its responsibility of search, a connection exists between Search Page and Course Database. Here, we can't really use a conceptual user interface mockup, since we are now designing internal software components. Later on in the specialization, you will learn about different technical diagrams that describe the structure and behavior of these components.

8:58 Components, when they are refined enough, turn into collections of functions, classes or other components. These pieces then represent a much simpler problem that the developers can individually implement. You can easily imagine that larger systems require more design time. With large systems, there are also more components, connections and responsibilities to keep track of. And since these components themselves will be big, they will be refined to many more components before the design can be detailed. You now have learned to take some time to think about the problem and outline the conceptual and technical design before actually implementing the solution. Design artifacts, like conceptual mockups, help to clarify design decisions with clients and users. Technical diagrams help to coordinate development work. However, recognizing the importance of design is just the beginning. Throughout the rest of this

specialization, we will explore various design techniques so you can get the most out of your design process.