Project Objectives:

- **Project objectives** are the guideposts when making decisions throughout the lifespan of the project. They clearly communicate the project's purpose. This is usually 2-3 simple sentences about what this specific part of the project was trying to accomplish.
- Project objectives should make reference to the deliverables of the project. If you can't get a sense for the deliverables that are needed to fulfill the project objective, it may be written at too high a level. On the other hand, if a project objective describes the characteristics of the deliverables, it's probably written at too low a level. If the statements describe the features and functions, for instance, they are requirements, not project objectives.
- Objectives are important for three major reasons:
    1. They are in business terms. Once they are approved, they represent an agreement between the project manager and the project sponsor and other major stakeholders on the main purpose of the project.
    2. They help frame the project. If you know the project objectives, you can determine the deliverables needed to achieve the objectives. This in turn helps nail down the overall project scope, helps you identify risks and allows you to provide estimates on effort, duration and cost. Once the project starts, you can validate that all of the work that you are performing will ultimately help you achieve one or more project objectives.
    3. They help you declare success. At the end of the project, you should be able to talk to your sponsor to determine whether everything expected in the project objectives has, in fact, been achieved. If all of the objectives were not fully met, you may still be able to declare partial success.
- One technique for writing an objective is to make sure that it is **SMART**:

  Specific

  Make sure your objective is clearly defined. Narrow your scope of the objective so that it has a very tangible and specific outcome. This helps you focus your intent. When writing this part of the objective think of the Who, What, Where, When and Why of it all.

  Measurable

  Make sure you can actually quantify the objective. If it's not measurable, you won't know when the project objective has been met. You want to make sure the objective is trackable to keep you and the team accountable.

  Achievable

  Make sure you can accomplish the objective. Identify the clear steps that need to happen to make sure the objective is completed. When writing this portion of the objective as yourself how will you accomplish it? What steps need to be taken in order to accomplish the specific objective you've defined?

  Realistic

  Don't set objectives that can't be achieved within the constraints of the project. Make sure your objective is practical. Do you have the budget to do this? Is there enough time? Does your team have the right knowledge or do you have time to invest in learning?

  Time-bound.

  When will this be done by? Having a clear end date defined helps everybody involved. It lets you know when you need to focus on that objective. It also helps you set a relationship between multiple objectives on a project as well.

  Note that an objective does not have to use the SMART technique to still be valid.

Use Cases:

- A **use case** describes how a user uses a system to accomplish a particular goal.
- It is a methodology used in system analysis to identify, clarify and organize system requirements. The use case is made up of a set of possible sequences of interactions between systems and users in a particular environment and related to a particular goal. The method creates a document that describes all the steps taken by a user to complete an activity.
- Every use case contains three essential elements:
    1. **The actor:** The system user. This can be a single person or a group of people interacting with the process.
    2. **The goal:** The final successful outcome that completes the process.
    3. **The system:** The process and steps taken to reach the end goal, including the necessary functional requirements and their anticipated behaviors.
- Other additional elements to consider when writing a use case include:
    - Stakeholders or anybody with an interest or investment in how the system performs.
    - Preconditions or the elements that must be true before a use case can occur.
    - Triggers or the events that cause the use case to begin.
    - Post-conditions or what the system should have completed by the end of the steps.
- Use cases describe the functional requirements of a system from the end user's perspective, creating a goal-focused sequence of events that is easy for users and developers to follow. A complete use case will include one main or basic flow and various alternate flows. The alternate flow, also known as an extending use case, describes normal variations to the basic flow as well as unusual situations.
- A use case should display the following characteristics:
    - Organizes functional requirements.
    - Models the goals of system/actor interactions.
    - Records paths, called **scenarios**, from trigger events to goals.
    - Describes one main flow of events and various alternate flows.
    - Multi-level, so that one use case can use the functionality of another one.
- The writing process includes:
    1. Identifying all system users and creating a profile for each one. This includes every role played by a user who interacts with the system.
    2. Selecting one user and defining their goal or what the user hopes to accomplish by interacting with the system. Each of these goals becomes a use case.
    3. Describing the course taken for each use case through the system to reach that goal.
    4. Considering every alternate course of events and extending use cases or the different courses that can be taken to reach the goal.
    5. Identifying commonalities in journeys to create common course use cases and write descriptions of each.
    6. Repeating steps two through five for all other system users.
- Some benefits of writing use cases include:
    - The list of goals created in the use case writing process can be used to establish the complexity and cost of the system.
    - By focusing both on the user and the system, real system needs can be identified earlier in the design process.
    - Since use cases are written primarily in a narrative language they are easily understood by stakeholders, including customers, users and executives.

- The creation of extending use cases and the identification of exceptions to successful use case scenarios saves developers time by making it easier to define subtle system requirements.
- By identifying system boundaries in the design scope of the use case, developers can avoid scope creep.
- Premature design can be avoided by focusing on what the system should do rather than how it should do it.
- Reveal how a system should behave while also helping identify any errors that could arise in the process.

Use Scenarios:

- **Use scenarios** are scenarios in which your project was trying to tackle and address. The purpose of this is to be able to look at what your project can do from different angles.
- A usage scenario, or scenario for short, describes a real-world example of how one or more people or organizations interact with a system. They describe the steps, events, and/or actions which occur during the interaction. Usage scenarios can be very detailed, indicating exactly how someone works with the user interface, or reasonably high-level describing the critical business actions but not indicating how they're performed.
- There are several differences between use cases and scenarios.
  First, a use case typically refers to generic actors, such as customer, whereas scenarios typically refer to examples of the actors such as John Smith and Sally Jones. There's nothing stopping you from writing a generic scenario, but it's usually better to personalize the scenarios to increase their understandability.
  Second, usage scenarios describe a single path of logic whereas use cases typically describe several paths.

Personas:

- A **persona** defines an archetypical user of a system, an example of the kind of person who would interact with it. The idea is that if you want to design effective software, then it needs to be designed for a specific person.
- Personas are incredibly useful when you don't have easy access to real users because they act as "user stand-ins", helping to guide your decisions about functionality and design.
- Here are some techniques for writing effective personas:
  - Write specific personas. You will have a much greater degree of success designing for a single person. The "generic user" will bend and stretch to meet the moment, but your true goal should be to develop software which bends and stretches.
  - You want to know what the persona's goals are so that you can see what your system needs to do, and not do.
  - Sometimes you want to identify negative personas, people that you are not designing for.
  - If you identify more than three primary personas your scope is likely too large.
  - You want a finite number of personas, your goal is to narrow down the people that you are designing the system for.

User Stories:

- A **user story** is a note that captures what a user does or needs to do as part of her work. Each user story consists of a short description written from the user's point of view, with natural language. Unlike the traditional requirement capturing, user stories focus on what the user needs instead of what the system should deliver. This leaves room for further discussion of solutions and the result of a system that can really fit into the customers' business workflow, solving their operational problems and most importantly adding value to the organization.

- A user story is a lightweight method for quickly capturing the "who", "what" and "why" of a product requirement. In simple terms, user stories are stated ideas of requirements that express what users need. User stories are brief, with each element often containing fewer than 10 or 15 words each. User stories are "to-do" lists that help you determine the steps along the project's path. They help ensure that your process, as well as the resulting product, will meet your requirements.
- A user story is defined incrementally, in three stages, known as the **3 C's**:
    - **Card:** The brief description of the need. User stories are written as cards. Each user story card has a short sentence with just enough text to remind everyone of what the story is about.
    - **Conversation:** The conversations that happen during backlog grooming and iteration planning to solidify the details.
      I.e. These are the details behind the story that come out during conversations with the product owner.
    - **Confirmation:** The tests that confirm the story's satisfactory completion.
      I.e. Acceptance tests confirm a story was coded correctly.
- A good user story should be **INVEST**:
    - **Independent:** Should be self-contained in a way that allows it to be released without depending on one another.
    - **Negotiable:** Only capture the essence of the user's need, leaving room for conversation. User story should not be written like a contract.
    - **Valuable:** Delivers value to the end user.
    - **Estimable:** User stories have to be able to be estimated so it can be properly prioritized and fit into sprints.
    - **Small:** A user story is a small chunk of work that allows it to be completed in about 3 to 4 days.
    - **Testable:** A user story has to be confirmed via pre-written acceptance criteria.
- A user story should follow this format:
  As a [description of a user], I want [some functionality] so that [some benefit].
- E.g.
  As a frequent flyer, I want to rebook a past trip so that I save time booking trips I take often.
- User stories should be detailed.
- You can break up large user stories into multiple smaller user stories.
- User stories are usually accompanied by the **acceptance criteria**. The acceptance criteria are the conditions that a software product must meet to be accepted by a user, a customer, or other system. They are unique for each user story and define the feature behavior from the end-user's perspective. Well-written acceptance criteria help avoid unexpected results at the end of a development stage and ensure that all stakeholders and users are satisfied with what they get.
- An acceptance criteria should follow this format:
  Given [how things] begin when [action takes] then [outcome of the action].
- E.g.
  User story: As a user, I want to be able to recover the password to my account, so that I will be able to access my account in case I forgot the password.
  Acceptance criteria: Given that the user has navigated to the login page, when the user has selected the forgot password option and has entered a valid email to receive a link for password recovery, then the system sends the link to the entered email. Then, given that the user received the link via the email, when the user navigates through the link received in the email, then the system enables the user to set a new password.

Use Cases vs User Stories:

- While both a use case and a user story are used to identify system users and describe their goals, the two terms are not interchangeable; they serve different purposes. While a use case is more specific and looks directly at how a system will act, a user story focuses on the result of the activities and the benefit of the process being described. I.e. User stories are centered on the result and the benefit of the thing you're describing, whereas use cases describe how your system will act.
- More specifically, the use case describes a set of interactions that occur between a system and an actor to reach a goal. The user story will describe what the user will do when they interact with the system, focusing on the benefit derived from performing a specific activity.
- In addition, user stories are often less documented than use cases and deliberately neglect important details. User stories are primarily used to create conversations by asking questions during scrum meetings, whereas use cases are used by developers and testers to understand all the steps a system must take to satisfy a user's request.