1. **Gather requirements:** What does the app need to do? What problem are you trying to solve? Write down specifics.
2. **Describe the app:** Zwaby specifications~
3. **Identify the main objects:** Starting point in identifying needed Classes. Use stories/descriptions from step 2 to pick out the most important ideas/concepts/things in the application, and discard what is irrelevant.
4. **Describe the interactions:** E.g. a customer needs to open a bank account, rocket needs to explode when it touches an asteroid, etc. This lets you start to better understand the responsibilities of different objects, the behaviors they need to have, and when they interact with other objects, what they do, in what order 🡪 Sequence diagram.
5. **Create a Class Diagram:** Visual representation of the classes needed. Be very specific about object-oriented principles (inheritance, polymorphism, etc.).

**\* Main result expected:** Class diagram 🡪 Most common way to write down the Classes we need to make, the methods in those Classes, and the interaction between the different objects in the application. This is done multiple times in an agile iterative approach to software development.

**Gathering Requirements – What MUST the application do?**

1. **Functional Requirements:** what does the app need to do? Features, capabilities. Short, simple, succinct statements.

(e.g. System must display the heart rate, temperature, and blood pressure of a patient connected to the patient monitor.)

(e.g. Application must allow user to search by customer’s last name, telephone number, or order number.)

(e.g. Program must allow receipts to be generated via email.)

(e.g. Application must allow user to create 140-character message.)

(e.g. Application must continue to function without network connection.)

1. **Non-Functional Requirements:** what else?

* What help/documentation needs to be provided?
* What are the legal requirements? (What are the laws?)
* Performance requirements? (Response time, …)
* Support requirements? (If there is an issue with the app at 3am on a Sunday morning, what needs to happen?)
* Security requirements? (Depends on the app)

(e.g. System must respond to searches within 2 seconds.)

(e.g. Help desk available by telephone, Mon-Fri 8am-6pm.)

(e.g. Comply with all relevant regulations.)

(e.g. Be available 99.99% of time during business hours.)

If more formality is needed by the corporation, use **FERPS/FERPS+** 🡪 Functional, Usability, Reliability, Performance, and Supportability requirements + Design, Implementation, Interface, and Physical requirements.

**UML** – Unified Modeling Language – graphical notation specifically for drawing diagrams of an object-oriented system.

Class diagram example:

|  |
| --- |
| BankAccount |
| accountNumber |
| balance |
| dateOpened |
| accountType |
|  |
| open() |
| close() |
| deposit() |
| withdraw() |

How does the user accomplish a particular goal using our application?

Use Cases vs User Stories

**\* Use Cases** 🡪 Detailed record of conversation.

* **Title** – What is the goal? Short phrase, active verb.

(e.g. Register new member, Transfer funds, Purchase items, Create new page, Collect late payments, Process accounts, etc.) 🡪 Separate but distinct goals of the application.

* **Actor** – Who desires the goal? External entity that acts on our system.

(e.g. User, Customer, Member, Administrator, ACMESystem, etc.)

* **Scenario** – How is it accomplished? Steps/details needed to accomplish the goal.

(e.g. **Customer** *reviews* **items** in **shopping cart**. Customer *provides* **payment** and shipping information. System *validates payment* information and *responds* with confirmation of **order** and *provides* **order number** that Customer can use to *check on* **order status**. System will *send* Customer a confirmation of **order details** and tracking number in an **email**.) This can also be done in a numbered steps list.

\* You may add more details to a scenario.

Extensions 🡪 (e.g. Describe the steps for out-of-stock situations, Describe steps for order never finalized, etc.)

Preconditions 🡪 (e.g. Customer has added at least one item to shopping cart.)

**\* Actors**

Human 🡪 Zwaby administrator, cleaning company manager, cleaning professional, and homeowner (there are more details to the human actors, e.g. Requester and Approver).

Non-Human 🡪 Other computer systems/software (Azure, Stripe, Salesforce, Analytics~, Bench, etc…)

**\* Scenarios**

Describing a goal that an actor can accomplish in a single encounter. Stay focused on the user’s goal and intention.

Identifying scenarios 🡪 user goals (e.g. Purchase items, Create new document, Balance accounts, System validates payment information, etc.)

**Use Case Diagram** (UML) 🡪 Diagram of several use cases and multiple actors at the same time. It’s an overview of how they interact all in context.

Example Use Case goal titles (Knowledge base):

* Search Articles
* View Article
* Manage Users
* Create Article
* View Analytics

Example Actors:

* Visitor
* Contributor
* Administrator
* Analytics System (NH)

\* Use lines to connect Actors to Use Cases.

**User Story** 🡪 1-2 sentences, short and sweet. Focus on one specific goal of one specific user for a particular reason. It’s a placeholder for conversation.

* As a (*type of user*)
* I want (*goal*)
* so that (*reason*)

e.g. As a *Bank Customer*, I want *to change my PIN online*, so that *I don’t have to go into a branch.*

e.g. As a *User*, I want *to search by keyword*, so that *I can find and read relevant articles.*

**Creating a conceptual model of our system**

* **Identifying objects/classes** 🡪 Collect use cases/user stories and any other written requirements together.

Go through Use Case Scenario and start picking out all the **nouns** and make a list of them. These nouns are the possible/candidate objects (see Scenario example above). Some selected nouns might be *properties* of a Class and can therefore be eliminated from the list. Creating class diagrams here can be useful.

* **Identifying class relationships** 🡪 Relate the various objects in our list. You may add small notes to describe the most important and relevant relationships.
* **Identifying class responsibilities** 🡪 Narrowing down what objects will become Classes in the application. Distribute responsibilities between the various objects. An object should be responsible for itself as much as possible. Do not store all responsibilities in a single ‘master’ object.

Go through Use Case Scenario and start picking out all the **verbs/verb phrases** and make a list of them. Assign these verb phrases to objects in Object list (e.g. Check order status is assigned to the Order object as its responsibility).

\* **CRC Cards** (Class-Responsibility-Collaboration) 🡪 Use index cards to represent each Class.