Requirements

Software Requirement specifies a need to be fulfilled by the software product. [come from **stakeholders**]

- Identifying requirements is often not easy
- can be divided to **FR** and **NFR**
- can be **prioritized** based on the importance and urgency [group requirements into priority categories]
- can be **discarded** if they are considered 'out of scope'

Brownfield project: develop a product to replace/update an **existing software product**
Greenfield project: develop a totally new system from **scratch**

Stakeholder: An individual or an organization that is involved or potentially affected by the software project. **Example:** users, sponsors, developers, interest groups, government agencies

Functional requirements: what the system should do.
Non-functional requirements specify the constraints under which the system is developed and operated.

- NFRs are **easier to mis**s [stakeholders tend to think of functional requirements first]
- Sometimes NFRs are critical to success of the software

Scope

- **Data requirements** e.g. size, volatility, persistency etc.
- **Environment requirements** e.g. technical environment in which the system would operate in or needs to be compatible with.
- **Others:** Accessibility, Capacity, Compliance with regulations, Documentation, Disaster recovery, Efficiency, Extensibility, Fault tolerance, Interoperability, Maintainability, Privacy, Portability, Quality, Reliability, Response time, Robustness, Scalability, Security, Stability, Testability, and more ...

Well-defined requirements

For individual requirements

- Unambiguous
- Testable (verifiable)
- Clear (concise, terse, simple, precise)
- Correct
- Understandable
- Feasible (realistic, possible)
- Independent
- Atomic **[Not divisible any further]**
- Necessary
- Implementation-free (i.e. abstract)

For set of requirements as a whole should be

- Consistent
- Non-redundant
- Complete

In a **brainstorming session** there are no "bad" ideas. The aim is to generate ideas; not to validate them.
User Surveys: To solicit responses and opinions from a large number of stakeholders regarding a current product or a new product.

Observation:

- Observing users in their natural work environment can **uncover product requirements**.
- Usage **data of an existing system** can also be used to gather information about how an existing system is being used. e.g. to find the situations where the user makes mistakes when using the current system.

Interviews: Interviewing **stakeholders** and **domain experts** can produce useful information about project requirements.

Focus groups: A kind of **informal interview** within an interactive group setting.

Prototype: A mock up, a scaled down version, or a partial system constructed [E.g., UI prototype using Wireframe D]

- to get users' feedback.
- to validate a technical concept
- to give a preview
- to compare multiple alternatives on a small scale before committing fully to one alternative.
- for early field-testing under controlled conditions.
- for discovering as well as specifying requirements
- can **uncover requirements**

Prose: A textual description to describe requirements.

- Especially **useful** when **describing abstract ideas** such as the vision of a product.
- **Avoid using lengthy prose** to describe requirements; they can be **hard to follow [understand]**.

Feature list: A list of features of a product

- grouped according to some criteria such as **aspect, priority, order of delivery**, etc.
- can be organized hierarchically.

User story: Short, simple descriptions of a feature told from the perspective of the person who desires the new capability, usually a **user** or **customer** of the system

- capture user requirements in a way that is convenient for **scoping, estimation**, and **scheduling**
- can capture non-functional requirements
- handy for recording requirements during **early stages** of requirements gathering
- differ from traditional requirements specifications mainly in the level of detail.
- only provide enough details to make a low risk estimate of **how long** the user story will take to implement. [When implement, meet customer to discuss more detail]
- It is **fine to add more details** such as conditions, priority, urgency, effort estimates

Writing a user story

- **format:** As a {user type/role} I can {function} so that {benefit} [{benefit} can be omitted if it is **obvious**.]
  Invalid Example: As a developer, I …, so that …..
  [not written from the perspective of the user/customer.]
- write using a physical medium or a digital tool [Trello]
- can write user stories at various levels.
- can add conditions of satisfaction to a user story

> *As a* lecturer, *I can* view the forum post count of each student *so that* I can identify the activity level of students in the forum.
>
> Conditions:
>
> ☑ Separate post count for each forum should be shown
> ☐ Total post count of a student should be shown
> ☑ The list should be sortable by student name and post count

Other useful info that can be added includes (but not limited to)

- **Priority:** how important the user story is
- **Size:** the estimated effort to implement the user story
- **Urgency:** how soon the feature is needed

Epics (or themes): High-level user stories, cover bigger functionality. You can break down these epics to multiple user stories of normal size.

> [Epic] As a lecturer, I can monitor student participation levels
>
> - As a lecturer, I can view the forum post count of each student so that I can identify the activity level of students in the forum
> - As a lecturer, I can view webcast view records of each student so that I can identify the students who did not view webcasts
> - As a lecturer, I can view file download statistics of each student so that I can identify the students who did not download lecture materials

Tips

- Clear your mind even you alr. have some idea of product
- Don't be too hasty to discard 'unusual' user stories:
- Don't go into too much detail
- Don't be biased by preconceived product ideas
- Don't discuss implementation details or whether you are actually going to implement it

Use Case: describes an interaction between the **system** and the **user(its actors)** for a specific functionality of the system.

- capture the functional requirements of a system
- **can not** capture NFRs, i.e., performance requirements
- can be expressed at different levels of abstraction [various levels of detail, Not just highest]
- **use case diagrams** [UML] that can illustrate use cases of a system visually
- no strict prefix requirement [i.e., no need to be UC only]

Use case in Testing

- **Use cases can be used for system testing and acceptance testing.**
- **To increase the E&E of testing, high-priority use cases are given more attention.**

Actor:

- A use case can involve multiple actors.
- An actor can be involved in many use cases.
- A single person/system can play many roles.
- Many persons/systems can play a single role.
- Use cases can be specified at various levels of detail.

Writing use case steps

- The main body is a sequence of steps that describes the interaction between the system and the actors.
- Describes **only the externally** visible behavior, **not internal** details, of a system
- A step **gives the intention** of the actor (not the mechanics). **UI details are usually omitted**. [leave as much flexibility to the UI designer as possible]
- numbering style is **not** a universal rule but a **widely used convention**
  Note: extension marked as ***a**. can happen at any step

Main Success Scenario (MSS)

- describes the most straightforward interaction for a given use case
- MSS should be **self-contained** [give us a **complete** usage scenario].
- assumes that **nothing** goes wrong
- Also called the **Basic Course of Action** or the **Main Flow of Events** of a use case.

Extensions: "add-on"s to the MSS that describe **exceptional / alternative** flow of events.

- scenario that can happen if certain things are not as expected by the MSS
- appear **below** the MSS
- not useful to mention events such as power failures or system crashes as extensions [system cannot function beyond such **catastrophic failures**]

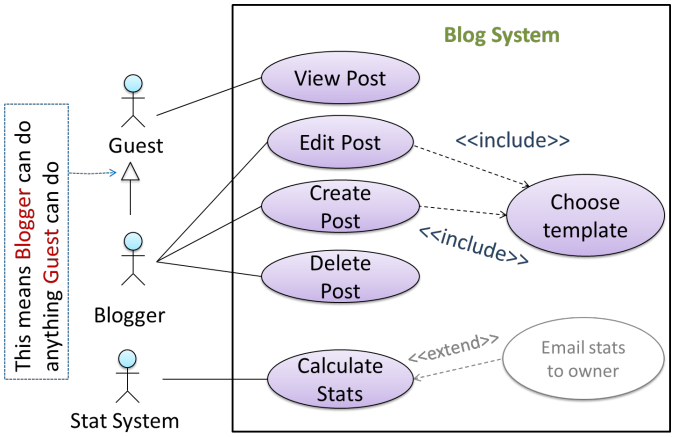
Inclusion: include **another use case** [use Underline Text]

Preconditions specify the specific state you expect the system to be in before the use case starts.

Guarantees specify what the use case promises to give us at the end of its operation.

Use case diagram

[↑ Also means all use cases available to Actor A are also available to Actor B]



- brief summary, do not over-complicate
- recommended not to use System as a user
- less detailed than textual use cases

Details

- **Inclusion:** dotted arrow and an <<include>> annotation [MSS->inclusion]
- **Extension:** dashed line arrow and <<extend>> arrows [extension->MSS]
- Inclusion and Extension arrow direction are **different**
- can use **actor generalization** in use case diagrams using a symbol similar to that of UML notation for **inheritance**.

Pros:

- simple notation and plain English descriptions, easy understood
- decouple user intention from mechanism → more freedom to optimize functionality
- Identifying all possible extensions encourages us to consider all situations
- Separating typical scenarios from special cases encourages us to optimize the typical scenarios.

Main Cons: Not good for capturing requirements that do not involve a user interacting with the system → should **not** be used as the **sole** means to specify requirements.

Glossary: A glossary serves to ensure that all stakeholders have a common understanding of the noteworthy terms, abbreviations, acronyms etc.

Supplementary requirements section: used to capture requirements that do not fit elsewhere. Typically, **this is where most Non-Functional Requirements will be listed**.