University of Hertfordshire

7COM1025
Programming for Software Engineers
Lecture 13

Dr. Renato C. de Amorim

THE ONE HUNDRED UNDER FIFTY

REQUIREMENTS

The 1st step in producing a good design for a piece of software is to understand what it actually needs to do.

Business requirements: Describes the value of the software in business terms. How it advances the needs of the organisation.

Functional requirements: describe the behaviour of the software. What the software is supposed to accomplish.

Non-functional requirements: describe the quality standards the software must achieve. How well the software works for users.

FUNCTIONAL REQUIREMENTS

Define the intended functionality for the API and should be developed in collaboration with the clients of the API.

They must represent the voice and needs of the user.

Do not assume you know what the users want just because you are a developer like them (API development).

You must identify target users and get the functional requirements from them using:

interviews, meetings, questionnaires, etc.

Make sure you know:

(i) What tasks they expect to achieve with the API.
(ii) What an optimal workflow would be from their perspective.
(iii) All potential inputs, including types and valid ranges.
(iv) All expected outputs, including type format and ranges.
(v) What file formats or protocols must be supported.
(vi) What (is any) mental models do they have for the problem domain.
(vii) What domain terminology do they use.

FUNCTIONAL REQUIREMENTS

Examples (ATM):
1.1 The system shall prevent further interaction if it's out of cash or is unable to communicate with the financial institution.

1.2 The system shall validate that the inserted card is valid for financial transactions on this ATM

1.3 The system shall validate that the PIN entered by the user is correct.

1.4 The system shall dispense the requested amount of money, if it is available, and debit the user's account by the same amount.

1.5 The system shall notify the user if the transaction could not be completed. In that case, no money shall be taken from the user's account.

NON-FUNCTIONAL REQUIREMENTS

Functional requirements can also be supported by non-functional requirements.

These qualities can be just as critical to the user as the actual functionality of the API. Examples:

Performance. Are there constraints on the speed of certain operations?
Platform compatibility.
Security. Are there data security, access or privacy concerns?
Scalability. Can the system handle real-world data inputs?
Flexibility. Will the system need to be extended after release?
Usability. Can the user easily understand, learn and use the API?
Concurrency. Does it need to use multiple processors?
Cost.

USE CASES

A use case describes the behaviour of an API based on interactions of a user or another piece of software.

Focusing on use cases helps you design an API from the perspective of the client.

A use case described a goal-oriented narrative description of a single unit of behaviour. It includes a distinct sequence of steps.

It can also provide pre and post conditions

USE CASE EXAMPLE

Name: Enter PIN
Version: 1.0.
Description: User enters PIN number to validate her Bank account information.
Goal: System validates User's PIN number.
Stakeholders:

1. User wants to use ATM services
2. Bank wants to validate the User's account.

Basic Course:

1. System validates that ATM card is valid for use with the ATM machine.
2. System prompts the user to enter PIN number.
3. User enters PIN number.
4. System checks that the PIN number is correct.

Extensions:

a. System failure to recognize ATM card:
    a-1. System displays error message and aborts operation.
b. User enters invalid PIN:
    b-1. System displays error message and lets User retry.

Trigger: User inserts card into ATM.
Postcondition: User's PIN number is validated for financial transactions

USE CASES

Good use cases:
- Use domain terminology.
- Don't over-specify use cases.
- Use cases don't define all requirements.
    They don't represent system design, list of features, algorithm specifics or anything that isn't user oriented.
- Use cases don't define a design
- Don't specify a design in use cases.
- Use cases can direct testing
- Expect to iterate
- Don't insist on complete coverage