Software Engineering

Lecture 3(c):

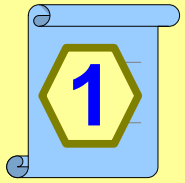
Introduction to Requirement analysis (1)

Outline

- The central role of requirement analysis in requirement engineering
- Types of requirement
- Requirement capturing

References

- Liskov & Guttag (2001):
  - Chapter 11
  - Modified to use UML use case diagram
- Sommerville (2011):
  - Chapter 4: 4.(1-2,4,5)



What is requirement engineering?

- RE is a process to:
  - capture,
  - **analyse**,
  - document, and
  - check ***what*** services a software provide
- Iterative:
  - incrementally refine the requirements

Iterative RE

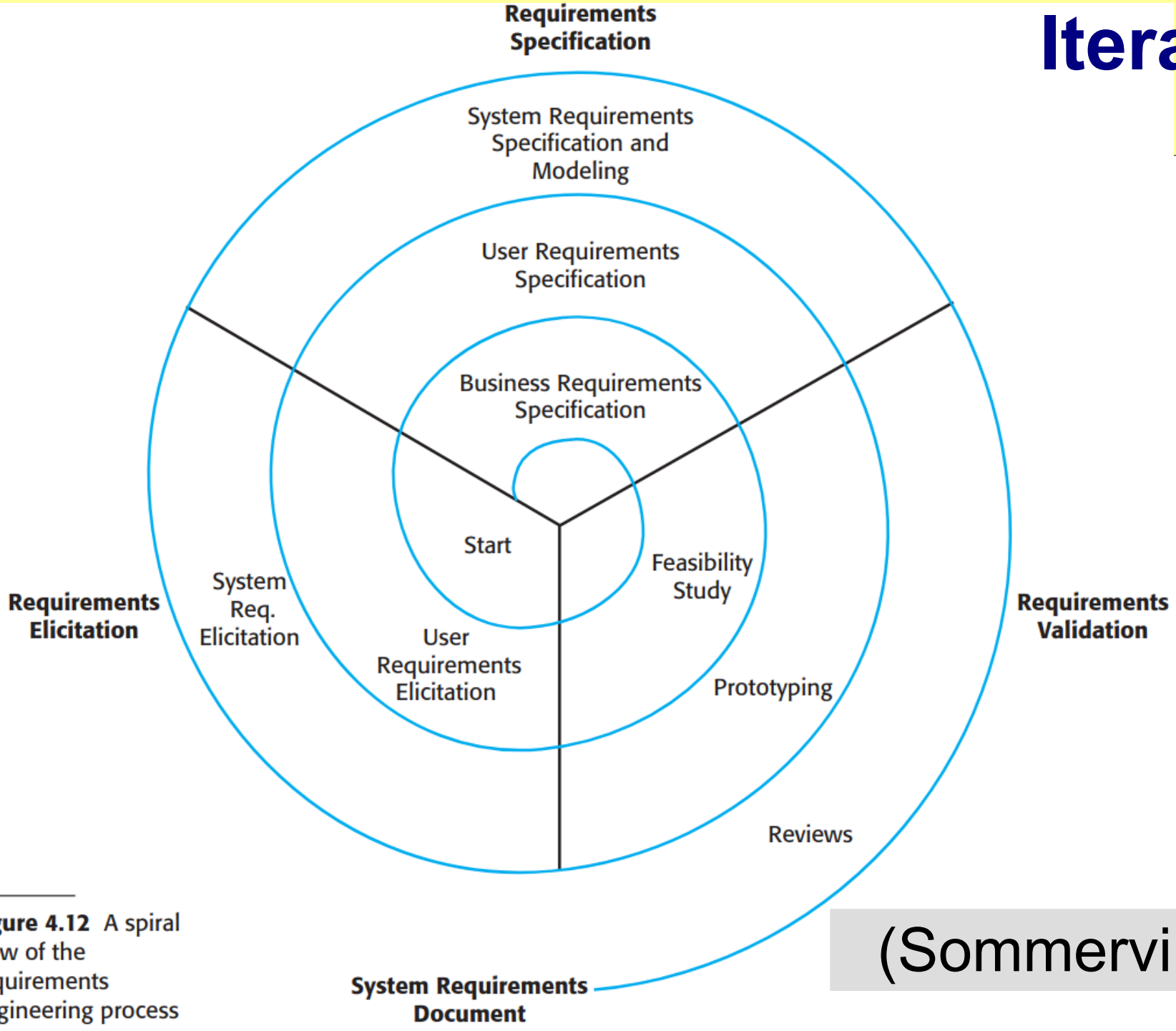


Figure 4.12 A spiral view of the requirements engineering process

(Sommerville, 2011)

Types of requirement

- Functional
- Non-functional

Functional requirements

- Statements about functions and data
- Data: statements about the *entities* of interests
  - written in a structured form:
    - Entity name: <attributes>
    - Relationship name (entities): <attributes>
- Derived from *normal* and *erroneous* user interactions:
  - normal: results in a normal program state
  - erroneous: results in errorneous program state

Example: KEngine requirements Section 12.4 (Liskov & Guttag, 2001)

```
        // EFFECTS: Uses the Web server to update the price of p a
nd
        //    all other positions for that stock.

  getPrices
        // CHECKS: There exists portfolio f in Open
        // EFFECTS: Uses the Web server to update the prices of
all positions in f.
        //    Also updates prices of all other positions for tho
se stocks.
```

12.4 REQUIREMENTS SPECIFICATION FOR A SEARCH ENGINE

This section explores a second example, a search engine that allows the user to run queries against a collection of documents. It describes both an abbreviated requirements analysis and the resulting requirements specification.

As usual, we begin our analysis by a scenario representing normal case behavior. Suppose the user starts a session with the search engine. The first question that comes up concerns whether the engine already has a collection of documents that it remembers from the last time it ran. Let's assume that the customer is interested only in new searches. Therefore, the first thing the user must do is identify some documents of interest. Let's assume that this is done by presenting a URL of a site containing documents; the engine will run searches against all of those documents. Furthermore, the customer is interested in multisite searches; therefore, the user can present additional URLs of document-containing sites, and the engine will increase its collection as a result. The customer indicates that the collection can be enlarged at any time, not just at the start of a session, and that there is no interest in removing documents from the collection.

The customer indicates that a user should be able to search the collection for a document with a particular title. However, the main purpose of the engine is to run queries against the collection, which means we have to decide what a query is. In consultation with the customer, we determine that a query begins by the user presenting a single word, which we will refer to as a *keyword*. The customer indicates that many words are uninteresting (e.g., "and" and "the") and will not be used as keywords. The customer expects the search engine to know what the uninteresting words are without any user intervention; thus, it must have access to

Example: KEngine requirements (2)

some storage, such as a file, that lists the uninteresting words.

The system responds to a query by presenting information about what documents contain the keyword. This information is ordered by how many times the keyword occurs in the documents. The system does not present the actual documents, but rather provides information so that the user can examine the matching documents further if desired.

However, the ability to query using a single keyword is quite limited, and the customer also requests the ability to "refine" a query by providing another keyword; the matching documents must contain all the keywords. The customer rules out more sophisticated queries, such as queries that match documents containing any one of their keywords or queries that require the keywords to be adjacent in the document in order for there to be a match. However, such queries are likely in a later release of the product.

Now we need to consider user and system errors, and also performance. The main performance issue is how to carry out the queries; the customer wants it to be done expeditiously. This requirement has two implications. First, the program must contain data structures that speed up the process of running a query. Second (and more important) is the question of whether querying requires visiting the Web sites containing the documents. The customer indicates that this should not happen; instead, the query should be based on information already known to the search engine. One implication of this decision is that the collection might not be up to date. A site might have been modified since the search engine was told about it, and queries will not reflect the modifications: they will miss newly added documents or find documents that no longer exist. The customer indicates that this is acceptable but that tracking modifications might be desired in a future release. The customer also indicates that all information about documents should be stored at the search engine, so that if a query matches a document, the user will be able to view the document even if it no longer exists at the site from which it was fetched. One point

Example: KEngine requirements (3)

to note about these decisions is that a trade-off is being made between speed of processing queries versus the space taken for storing documents at the search engine.

Now let's consider errors. There aren't any interesting system errors: the system has some persistent storage containing information about uninteresting words, but this storage is not modified and the customer is not concerned about media failures. Furthermore, the customer indicates that it is acceptable for the search engine to simply fail if something goes wrong.

There are interesting user errors, however. The user could enter an uninteresting word as a keyword or could enter a word not in any document; the customer indicates that the user should be told about the error in the first case, but that in the second case, the response will simply be an empty set of matches. The user might also present a URL for a site that doesn't exist, that doesn't contain documents, or that has already been added to the collection; all of these actions should result in the user being notified of the error. The customer indicates that it is acceptable if a document is found at multiple sites and that, in this case, the document will end up in the collection just once. Two documents are considered to be the same if they have the same title; again, a later release might handle things differently.

Now that we have a rough idea of what the search engine is supposed to do, we are ready to write the requirements specification. As we do so, we will uncover a number of issues that were overlooked in the analysis but must be resolved to arrive at a precise specification. Thus, the process of writing the requirements specification, including the definition of the data model, is an intrinsic and important part of the requirements analysis process.

The sets and relations for the search engine are defined in Figure 12.11 and the graph is given in Figure 12.12. A document has a title, some URLs (of the sites from which it was obtained), and a body; a body is a sequence of words. The `NK` node represents the uninteresting words; this set is fixed (its membership never changes). `Match` represents the set of documents that match the current query; `Key` is the set of keywords used in this query. `Key` and `NK` are disjoint

Example: KEngine normal FRs

documents of interest. Let's assume that this is done by presenting a URL of a site containing documents; the engine will run searches against all of those documents. Furthermore, the customer is interested in multisite searches; therefore, the user can present additional URLs of document-containing sites, and the engine will increase its collection as a result. The customer indicates that the collection can be enlarged at any time, not just at the start of a session, and that there is no interest in removing documents from the collection.

The customer indicates that a user should be able to search the collection for a document with a particular title. However, the main purpose of the engine is to run queries

Example: KEngine normal FRs

- *the first thing that the user should do is to identify the documents of interest...presenting the URL of a site*

→ ***obtain documents from an URL***

KEngine functions (1)

Functions		Descriptions
F1	Obtain documents	to retrieve web documents from a given URL, which could be the URL of a local folder or of a remote web site

Example: KEngine normal FRs (cont'd)

documents of interest. Let's assume that this is done by presenting a URL of a site containing documents; the engine will run searches against all of those documents. Furthermore, the customer is interested in multisite searches; therefore, the user can present additional URLs of document-containing sites, and the engine will increase its collection as a result. The customer indicates that the collection can be enlarged at any time, not just at the start of a session, and that there is no interest in removing documents from the collection.

The customer indicates that a user should be able to search the collection for a document with a particular title. However, the main purpose of the engine is to run queries against the collection, which means we have to decide what a query is. In consultation with the customer, we determine

some storage, such as a file, that lists the uninteresting words.

The system responds to a query by presenting information about what documents contain the keyword. This information is ordered by how many times the keyword occurs in the documents. The system does not present the actual documents, but rather provides information so that the user can examine the matching documents further if desired.

However, the ability to query using a single keyword is quite limited, and the customer also requests the ability to

- *the system run queries against the collection...presenting information about documents containing the keyword*

→ ***search for documents by <u>keyword</u>***

Example: KEngine normal FRs (cont'd)

The system responds to a query by presenting information about what documents contain the keyword. This information is ordered by how many times the keyword occurs in the documents. The system does not present the actual documents, but rather provides information so that the user can examine the matching documents further if desired.

However, the ability to query using a single keyword is quite limited, and the customer also requests the ability to "refine" a query by providing another keyword; the matching documents must contain all the keywords. The customer rules out more sophisticated queries, such as

- *the customer requests the ability to “refine” a query by providing another keyword (the matching documents must contain all the keywords)*

→ ***incrementally* search for documents by *keywords***

KEngine functions (2)

Functions		Descriptions
F1	Obtain documents	to retrieve web documents from a given URL, which could be the URL of a local folder or of a remote web site
F2	Search for documents	to search the documents collection for the documents that contain the keywords of a query; allowing the user to refine query with more keywords

Example: KEngine normal FRs (cont'd)

documents of interest. Let's assume that this is done by presenting a URL of a site containing documents; the engine will run searches against all of those documents. Furthermore, the customer is interested in multisite searches; therefore, the user can present additional URLs of document-containing sites, and the engine will increase its collection as a result. The customer indicates that the collection can be enlarged at any time, not just at the start of a session, and that there is no interest in removing documents from the collection.

The customer indicates that a user should be able to search the collection for a document with a particular title.

However, the main purpose of the engine is to run queries against the collection, which means we have to decide what

- *the user should be able to search the (documents) collection for a document given a title*

→ ***display a document***

KEngine functions (3)

Functions		Descriptions
F1	Obtain documents	to retrieve web documents from a given URL, which could be the URL of a local folder or of a remote web site
F2	Search for documents	to search the documents collection for the documents that contain the keywords of a query
F3	Display a document	to retrieve a document from the documents collection given its title

KEngine errorneous

- System errors: none
- User errors:
  - E1: *Enters a wrong, empty-target, or duplicate URL*

    → informs with an error
  - E2: *Enters a non-keyword*

    → informs with an error
  - E3: *Enters a non-existent word*

    → returns an empty result

KEngine functions (4)

Functions	Descriptions
Obtain documents	E1: If the user enters a wrong, empty-target, or duplicate URL, the system informs with an error

KEngine functions (5)

Functions	Descriptions
Obtain documents	E1: If the user enters a wrong, empty-target, or duplicate URL, the system informs with an error
Search for documents	E2: If the user enters a non-keyword, the system informs with an error E3: If the user enters a non-existent word, the system returns with an empty result

KEngine data requirements

- *a document has a title, some URLs, and a body; a body is a sequence of words*

  → Document: title, Url, body

- *many words are uninteresting and are not used as keywords*

→ Keyword, Non-keyword

appears-in(Keyword,Document): frequency

- *a query begins by having a single keyword*
  - *can be refined with another keyword*

→ Query: keywords

has(Query,Keyword)

- *a query result consists of matches, which are documents containing all the query keywords, ordered by the keyword frequencies*

→ Match: document, sum-freq

has(Query, Match)

refers-to(Match, Document)

Non-functional requirements

- Constraints on the functions or emergent properties
- NFR may lead to other FRs
- Should be quantified when possible:
  - e.g. range of response time is 1-5 secs

Types of NFR

- Performance
- Modifiability
- Reusability
- Delivery schedule

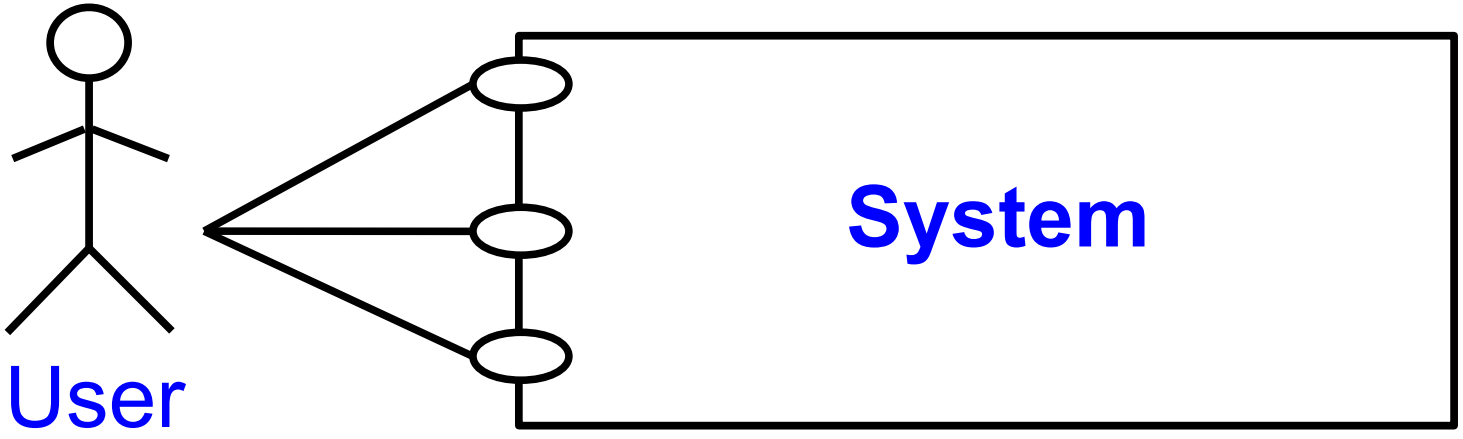
Requirement capturing

- To capture the *detailed* requirements
- Techniques:
  - interview
  - document capturing
  - **use case**
  - prototyping

Use case

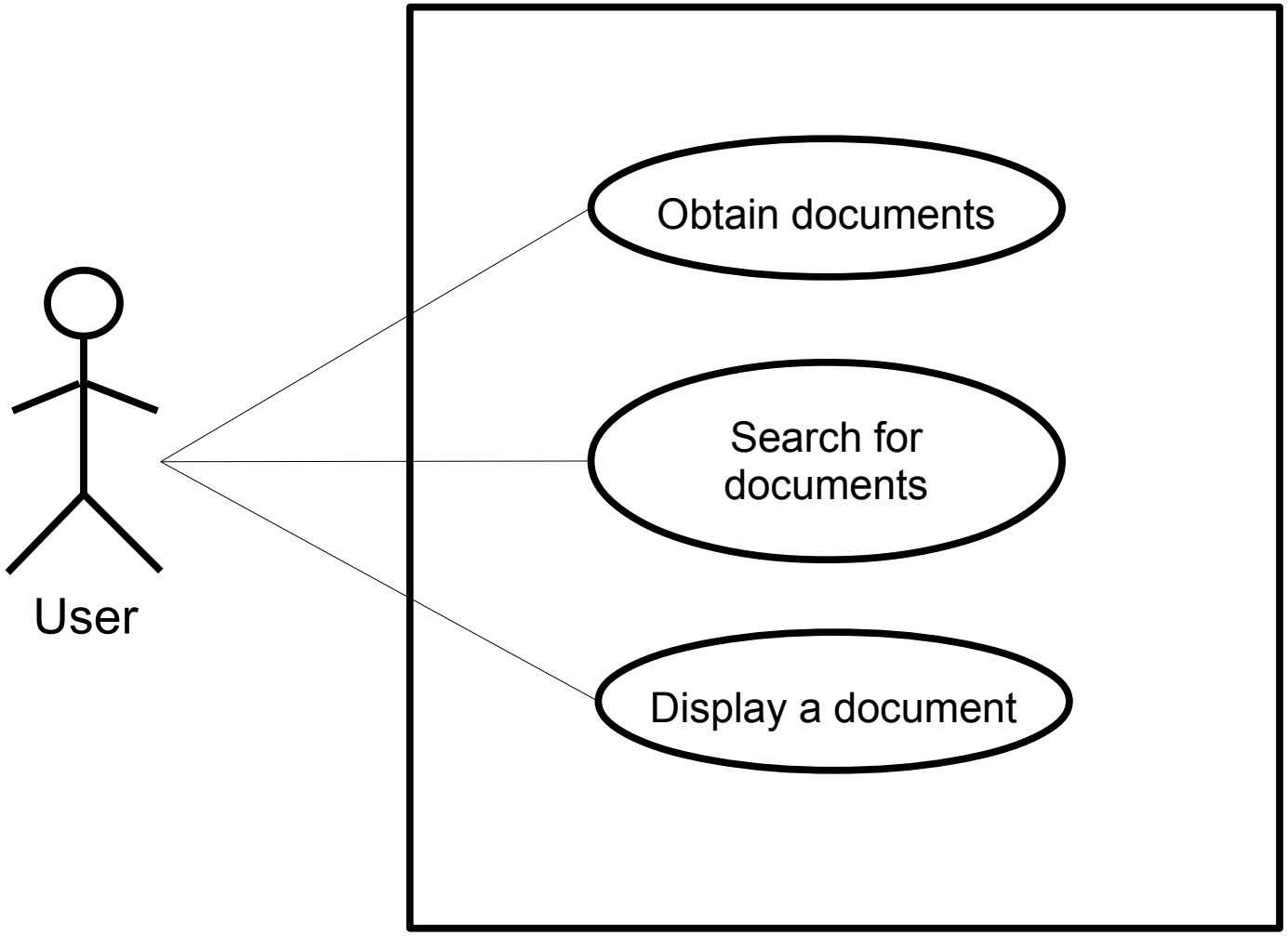
- One or more related user interactions with system
  - an interaction is structured as a *scenario*
- Capture the details for each function
- Types: normal and extended
  - extended type include alternative and erroneous interactions
- Use cases can be linked
- Documented using a *use case description*

UC illustration



User interacts with system by performing its functions

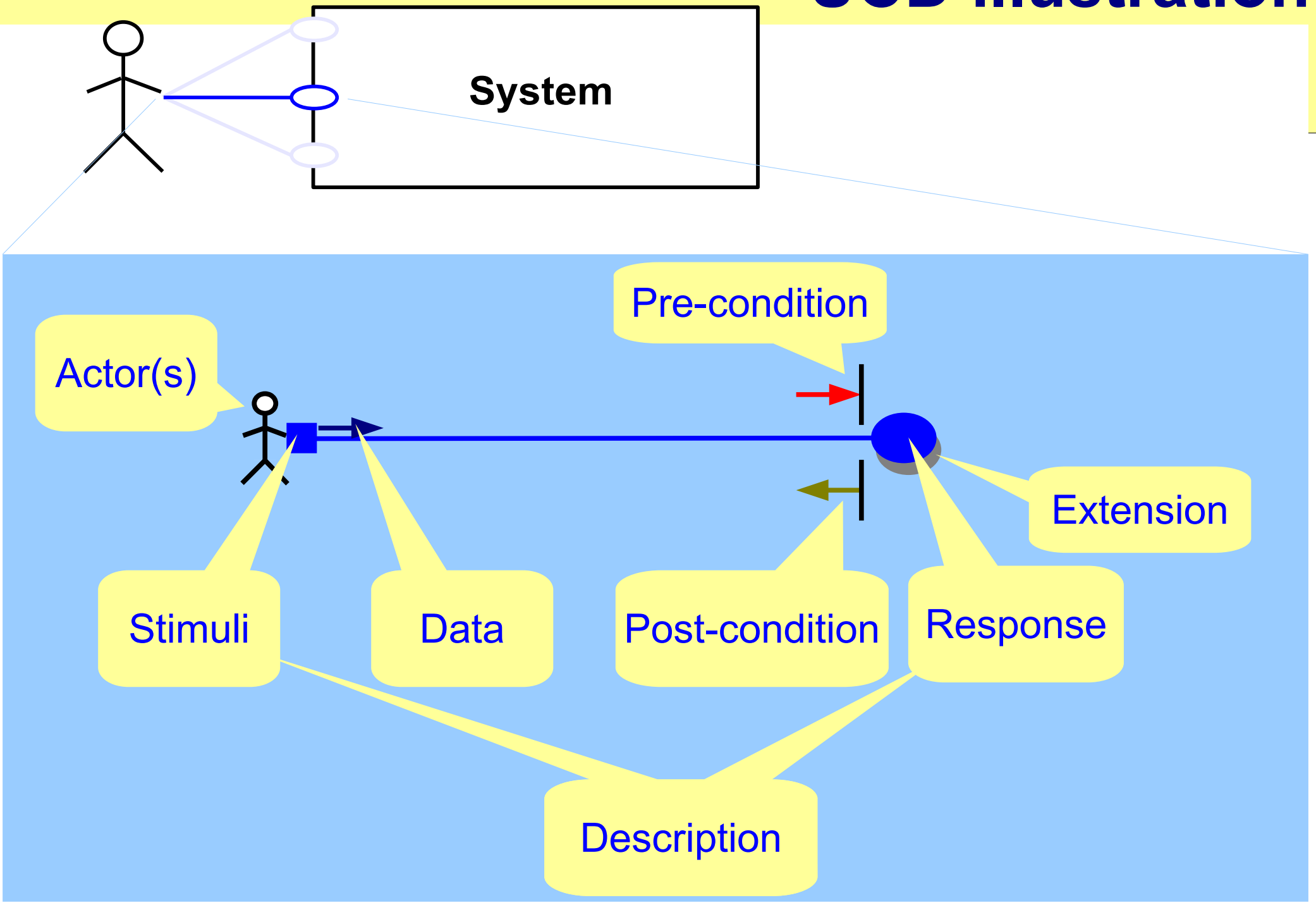
Example: KEngine



Use case description (UCD)

<Function name>	*<Basic | Extended>*
Actors	*Name(s) of user(s) that interact*
Description	*Short description of the use case*
Data	*Data requirements*
Stimuli	*User action that causes the system to perform this function*
Response	*Description of function (what the system does in response to stimuli)*
Pre-conditions	*synonymous to REQUIRES*
Post-conditions	*synonymous to EFFECTS*
Extension	*errors (if any)*

UCD illustration



Example: KEngine F2 (basic)

Search for documents	**Basic**
Actors	User
Description	A user enters a keyword query and requests the system to execute it
Data	The input data include a keyword query
Stimulus	User command that requests the system to execute the query

(cont'd)

Response	The system searches in the collection for the documents containing all the query keywords and return them as the result. Each document is considered a match, containing an aggregate of all the frequencies of the query keywords.
Pre-conditions	A document collection has been obtained and analysed to determine the keywords and non-keywords
Post-conditions	Query result containing the matching documents

F2 (extended)

Search for documents	Extended
Actors	User
Description	A user enters a keyword query and requests the system to execute it
Data	The input data include a keyword query
Stimulus	User command that requests the system to execute the query
Response	The system searches in the collection for the documents containing all the query keywords and return them as the result. Each document is considered a match, containing an aggregate of all the frequencies of the query keywords.
Pre-conditions	A document collection has been obtained and analysed to determine the keywords and non-keywords
Post-conditions	Query result containing the matching documents

(cont'd)

Additional scenarios	User enters a non-keyword → System informs with an error User enters a non-existent word → System returns an empty result

two erroneous interactions

Summary

- Requirements may be functional or non-functional
- A requirement capturing technique is to use use case description (UCD)

Q & A