Software Engineering – Lecture 3
Software Specification

Instructor : Conf. dr. Cristina Mîndruţă
cristina.mindruta@e-uvt.ro
Site:
http://sites.google.com/site/ingswcm

Adopted books:

Software Engineering (9th Edition) by Ian Sommerville

http://staff.fmi.uvt.ro/~cristina.mindruta

Engineering software products by Ian Sommerville

The Essentials of Modern Software Engineering by Ivar Jacobson et.all

Software process activities

- Software specification

The process of establishing what *services* and *qualities* are required from the system, and the *constraints* on the system's *operation* and *development*.

- Software development (design and implementation)

The process of converting the system specification into an *executable system*.

- Software validation

Verification and validation (V & V) is intended to show that a system conforms to its *specification* and meets the requirements of the system *customer*.

- Software evolution (maintenance)

The process of software evolving as its requirements *change*.

Software systems

Some factors that drive the design of software systems :

- Business and consumer needs that are not met by current products
- Dissatisfaction with existing business or consumer software products
- Changes in technology that make completely new types of product possible

Topics covered

- Software requirements
- Software requirements document
- Requirements engineering process for project-based software
- Requirements engineering for software products

Software specification

Def. **Software specification** = the process of establishing what *services* and *qualities* are required from the system, and the *constraints* on the system's *operation* and *development*.

Specification of the software requirements.

Def. **Requirement for software system** =

- *service* that the customer requires from a system or
- *quality* of the system or
- *constraint* under which system *operates* or
- *constraint* under which system *is developed*.

Def. **Requirement (IEEE 610.12-1990)**

1. A condition or capability needed by user to solve a problem or achieve an objective.

2. A condition or capability to be met or possessed by a system or system component to satisfy a contract, standard, specification or other formally imposed documents.

3. A documented representation of a condition or capability as in (1) or (2)

Types of requirements

Two orthogonal classifications

User requirements

Statements in natural language plus diagrams of the ***services*** the system provides, the expected ***qualities*** of the system, and its ***operational constraints***.

System requirements

A structured document setting out *detailed* descriptions of the system's ***services,*** the expected ***qualities*** of the system, and ***operational*** and ***development constraints***.

Functional requirements

Describe *functionality* of the system (system services) .

Extra-functional requirements

•*Quality attributes*

Relate to emergent *qualities* of the system.

•*Constraints*

- for the *system* to be developed.
- for the *development process*

Example: User and system functional requirements

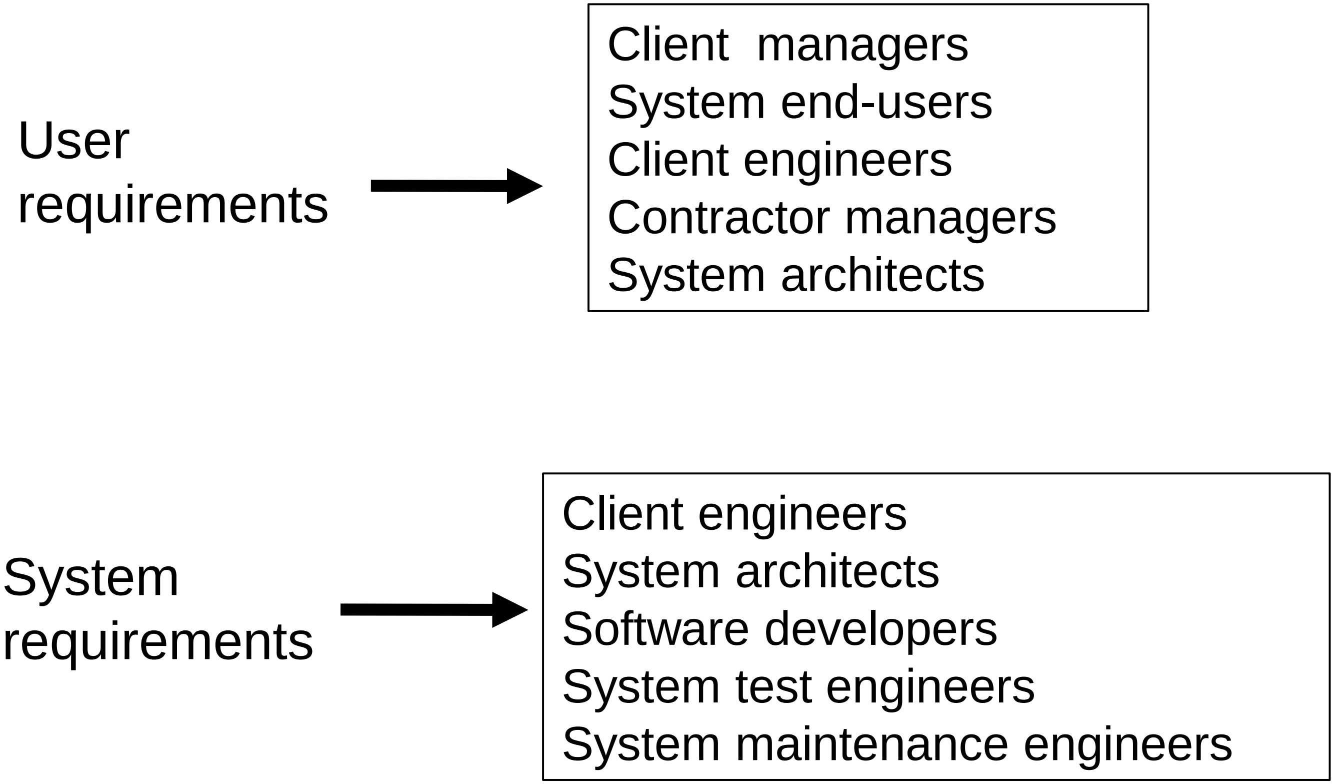
User requirement definition:

1.The software must provide a means of representing and accesing external files created by other tools.

System requirement specification:

1.1 The user should be provided with facilities to define the type of external files.
1.2 Each external file type may have an associated tool which may be applied to the file.
1.3 Each external file type may be represented as a specific icon on the user’s display.
1.4 Facilities should be provided for the icon representing an external file type to be defined by the user.
1.5 When an user selects an icon representing an external file, the effect of the selection is to apply the tool associated with the type of the external file to the file represented by the selected icon.

Requirements main readers



User requirements →

Client managers
System end-users
Client engineers
Contractor managers
System architects

System requirements →

Client engineers
System architects
Software developers
System test engineers
System maintenance engineers

Requirements’ completeness and consistency

In principle, functional requirements should be both complete and consistent.

- **Complete**
  - They should include descriptions of all facilities required.
- **Consistent**
  - There should be no conflicts or contradictions in the descriptions of the system facilities.

In practice, it is very difficult to produce a complete and consistent requirements document.

Quality requirements

Quality requirements may affect the overall architecture of a system rather than the individual components.

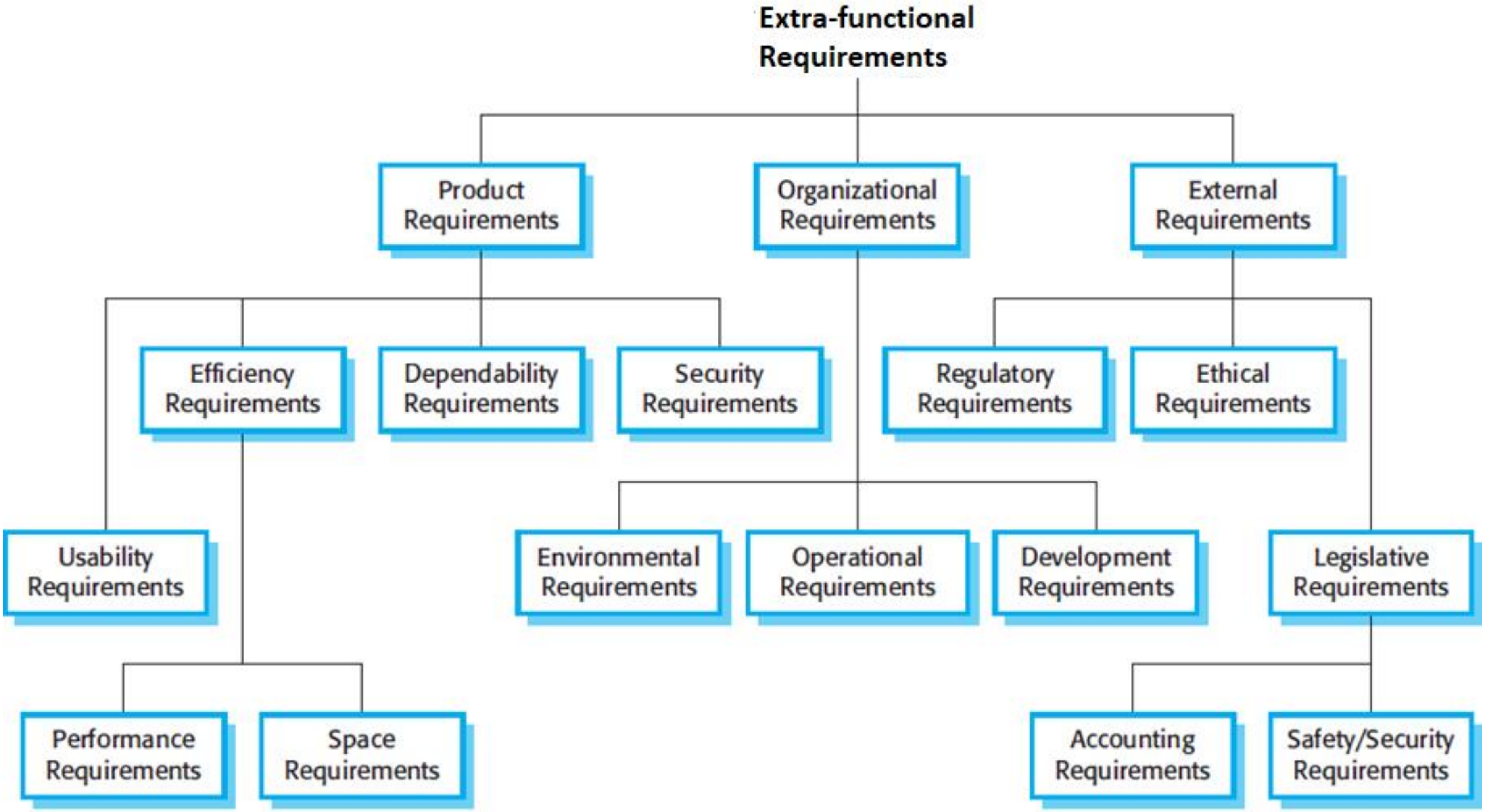
- Example: To ensure that performance requirements are met, you may have to organize the system to minimize communications between components.

Quality requirements may generate a number of related functional requirements that define system services that are required.

- Example: For a quality requirement related to security, a functional requirement may specify the exact encription algorithm to be used.

Quality requirements are often related to functional requirements, but should always be separately documented, with explicit documentation of their relation to functional requirements.

Types of extra-functional requirements



Extra-functional Requirements

Product Requirements

Organizational Requirements

External Requirements

Efficiency Requirements

Dependability Requirements

Security Requirements

Regulatory Requirements

Ethical Requirements

Usability Requirements

Environmental Requirements

Operational Requirements

Development Requirements

Legislative Requirements

Performance Requirements

Space Requirements

Accounting Requirements

Safety/Security Requirements

Adapted after ©Ian Sommerville **Engineering oftware Products** (2019) *and* **Software Engineering, 9th ed.** (2010)**. Chapter 4**

Goals and requirements

Extra-functional requirements may be very difficult to state precisely, and imprecise requirements may be difficult to verify.

- **Goal**

A general intention of the user (such as “ease of use”).

- **Verifiable extra-functional requirement**

A statement using some *measure* that can be *objectively tested*.

Goals are helpful to developers as they convey the intentions of the system users.

BUT

Software engineer *must* try to express them as **verifiable** extra-functional requirements.

Goals and requirements - Examples

Usability

- **A system goal**
  - The system should be easy to use by experienced controllers and should be organized in such a way that user errors are minimized.

- **A** (correspondent) **verifiable extra-functional requirement**
  - Experienced controllers shall be able to use all the system functions after a total of two hours training. After this training, the average number of errors made by experienced users shall not exceed two per day.

Requirements measures (examples)

Property	Measure
Speed	Processed transactions/second User/Event response time Screen refresh time
Size	M Bytes Number of ROM chips
Ease of use	Training time Number of help frames
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portability	Percentage of target dependent statements Number of target systems

Requirements interaction

Conflicts between different extra-functional requirements are common in complex systems.

Example: spacecraft system

- To minimize weight, the number of separate chips in the system should be minimized.
- To minimize power consumption, lower power chips should be used.
- However, using low power chips may mean that more chips have to be used. Which is the most critical requirement?

Problems with functional requirements derived from the application domain

Understandability

- Requirements are expressed in the language of the application domain;
- This is often not understood by software engineers developing the system.

Implicitness

- Domain specialists understand the area so well that they do not think of making the domain requirements explicit.

User and system requirements

Def. User requirements - should describe functional and extra-functional requirements in such a way that they are understandable by system users who don’t have detailed technical knowledge.

- Are defined using *natural language*, *tables* and *diagrams*, as these can be understood by all users.

Def. System requirements = specifications, more detailed than user requirements, of system services, system qualities, and constraints of operation and development of the system.

- Are intended to be a basis for designing the system.
- May be incorporated into the system contract.
- May be defined or illustrated using system models.

User and system requirements

Problems with natural language

- Lack of clarity Precision is difficult without making the document difficult to read.
- Requirements confusion Functional and extra-functional requirements tend to be mixed-up.
- Requirements amalgamation Several different requirements may be expressed together.

Guidelines for writing user requirements

- Invent a standard format and use it for all requirements.
- Use language in a consistent way. Use “shall” for mandatory requirements, “should” for desirable requirements.
- Use text highlighting to identify key parts of the requirement.
- Avoid the use of computer jargon.

System requirements and design

In principle:

- requirements should state WHAT the system should do
- design should describe HOW it does this.

Problems with NL (natural language) specification

Ambiguity

- The readers and writers of the requirement must interpret the same words in the same way. NL(natural language) is naturally ambiguous so this is very difficult.

Over-flexibility

- The same thing may be said in a number of different ways in the specification.

Lack of modularisation

- NL structures are inadequate to structure system requirements.

Alternatives to NL specification

Notation	Description
Structured natural language	Approach that depends on defining standard forms or templates to express the requirements specification.
Design description languages	Approach that uses a language like a programming language but with more abstract features to specify the requirements by defining an operational model of the system. This approach is not now widely used although it can be useful for interface specifications.
Graphical notations	A graphical language, supplemented by text annotations is used to define the functional requirements for the system. An early example of such a graphical language was SADT (Structured Analysis and Design Technique). Now UML is commonly used .
Mathematical specifications	Notations based on mathematical concepts such as finite-state machines or sets. These unambiguous specifications reduce the arguments between customer and contractor about system functionality. However, most customers don’t understand formal specifications and are reluctant to accept it as a system contract.

Formative evaluation

1. Define the software specification process.

2. How can an extra-functional requirement be made verifiable ? Give an example of a general extra-functional requirement and an example of a verifiable version for it.

3. What is the meaning of complete and consistent functional requirements ?

https://forms.gle/LZ9kGiQziimeDcbMA

Topics covered

- Software requirements
- Software requirements document
- Requirements engineering process for project-based software
- Requirements engineering for software products

The software requirements document

- The official statement of what is required of the system developers.
- Should include both a definition of user requirements and a specification of the system requirements.
- It is NOT a design document. As far as possible, it should set of WHAT the system should do rather than HOW it should do it.

IEEE requirements standard

Defines a generic structure for a requirements document that must be instantiated for each specific system.

- Introduction
  - Purpose of the requirements document
  - Scope of the product
  - Definitions, acronyms and abbreviations
  - References
  - Overview of the reminder document
- General description.
  - Product perspective
  - Product functions
  - User characteristics
  - General constraints
  - Assumptions and dependencies

IEEE requirements standard

(Cont.)

- Specific requirements.
  - Functional
  - Extra-functional
  - Interface

The most substantial part of the document

Wide variability in organizational practice – not a standard structure

- Appendices.
- Index.

Topics covered

- Software requirements
- Software requirements document
- Requirements engineering process for project-based software
- Requirements engineering for software products

Requirements engineering process

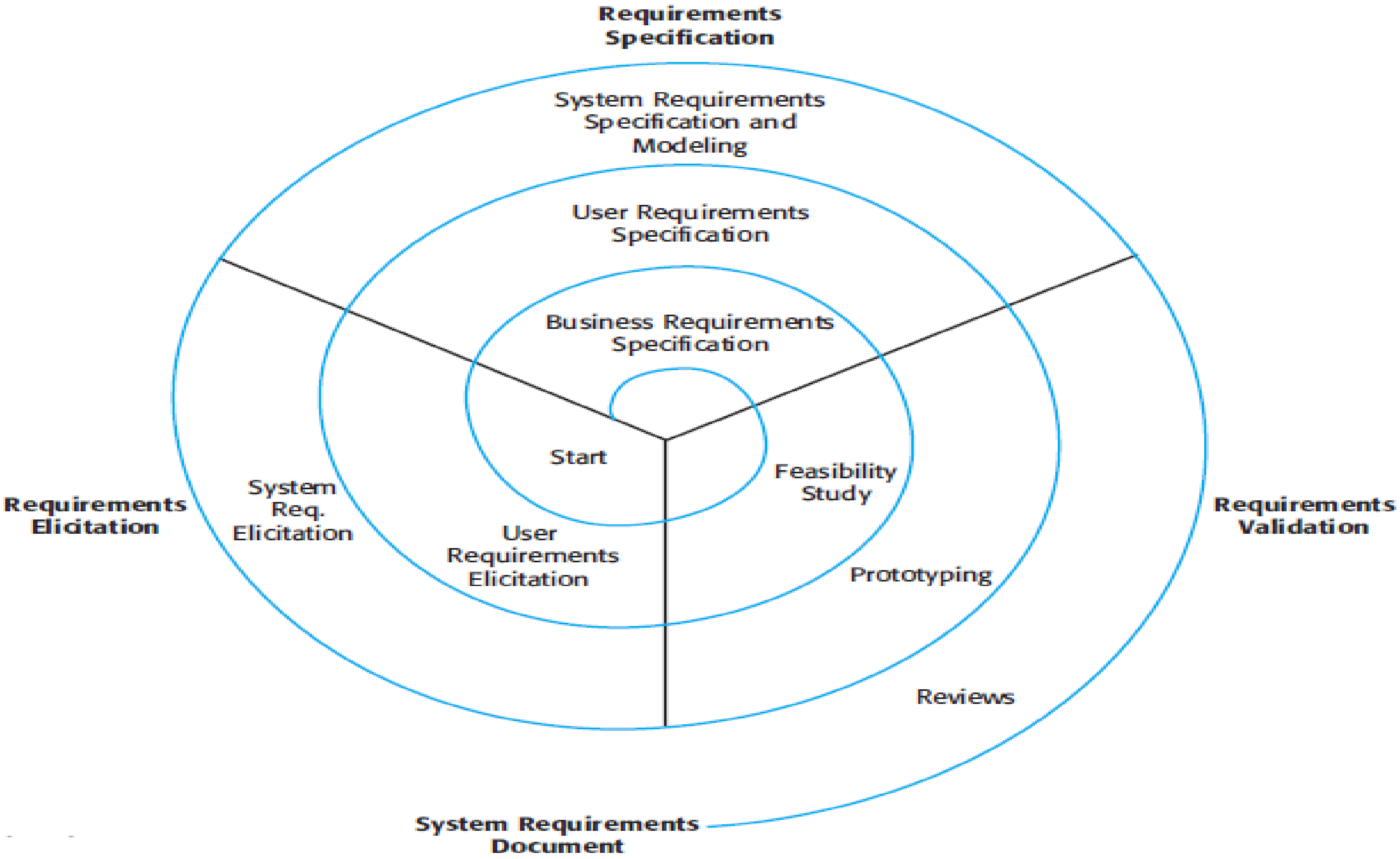
Def. Requirements engineering = the process of finding out, analysing, documenting, validation and managing software requirements.

Goal: Create and maintain a system requirements document.

Activities of the process

- Feasibility studies
- Requirements elicitation and analysis
- Requirements validation
- Requirements management

The requirements engineering process


Requirements Specification
System Requirements Specification and Modeling
User Requirements Specification
Business Requirements Specification
Start
Feasibility Study
Requirements Elicitation
System Req. Elicitation
User Requirements Elicitation
Prototyping
Requirements Validation
Reviews
System Requirements Document

Adapted after ©Ian Sommerville **Engineering oftware Products** (2019) *and* **Software Engineering, 9th ed.** (2010)**. Chapter 4**

Feasibility studies

A feasibility study decides whether or not the proposed system is worthwhile.

- Short focused study that checks :
  - If the system contributes to organizational objectives;
  - If the system can be engineered using current technology and within budget;
  - If the system can be integrated with other systems that are used.

Requirements elicitation (discovery)

Implies:

- understanding the application domain
- finding services that the system should provide
- defining qualities of the system
- determining system’s operational constraints.

Participants: technical staff, customers

Methods

- *Interviews :* getting an overall understanding of what users do and how they might interact with the system.
- *Surveys :* about used software systems
- *Ethnography :* observing and analysing how people actually work.
- *Scenarios and user stories :* real-life examples of how a system can be used.

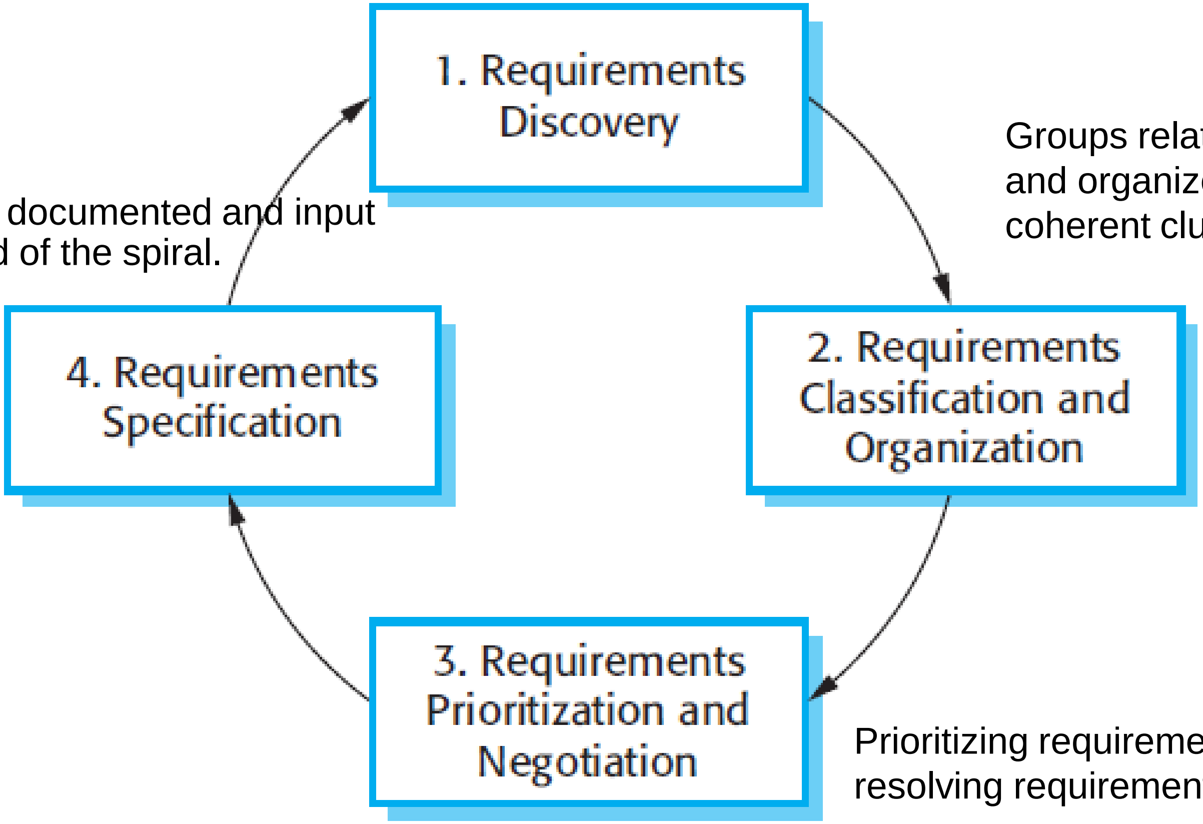
Sources of information:

- documentation,
- system stakeholders,
- specifications of similar systems.

Requirements elicitation and analysis

Activities

- Feasibility study
- **Requirements elicitation and analysis**
- Requirements validation
- Requirements management


1. Requirements Discovery
2. Requirements Classification and Organization
3. Requirements Prioritization and Negotiation
4. Requirements Specification

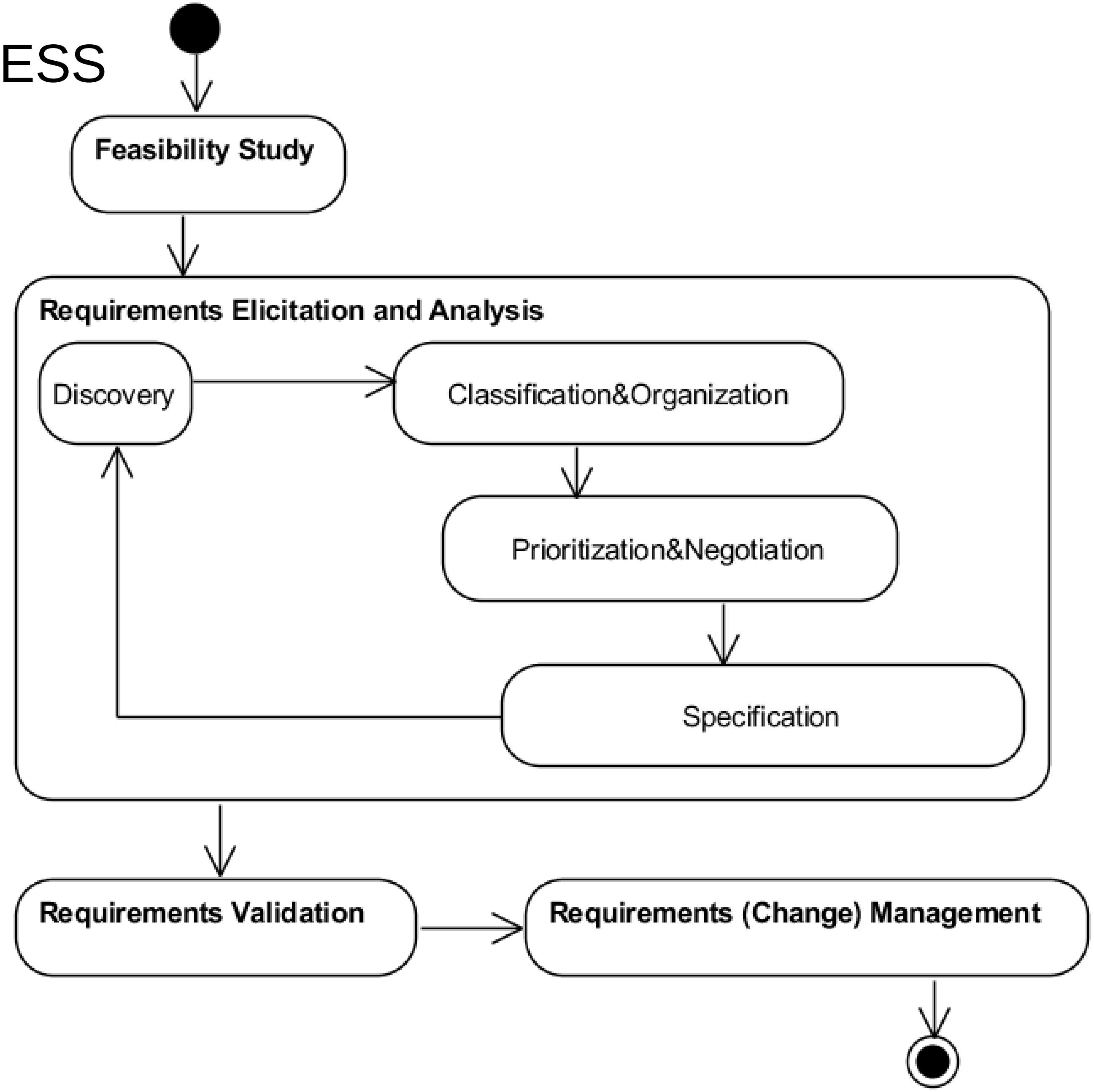
Interacting with stakeholders to discover their requirements.

Groups related requirements and organizes them into coherent clusters.

Prioritizing requirements and resolving requirements conflicts.

Requirements are documented and input into the next round of the spiral.

The whole PROCESS (simplified)


Feasibility Study
Requirements Elicitation and Analysis
Discovery
Classification&Organization
Prioritization&Negotiation
Specification
Requirements Validation
Requirements (Change) Management

Adapted after ©Ian Sommerville **Engineering oftware Products** (2019) *and* **Software Engineering, 9th ed.** (2010)**. Chapter 4**

Expressing requirements

1. **Formally** spelling the requirement

ID. name - *input-process-output*

Example:

Req. ID&name	Input	Process	Output
2.4 Student enrolment	-List of courses -Submit request	-Enrol student to selected courses	-Display acceptance message -Ask for confirmation message

2. Using **use case** narrative

Fundamentally, a use case describes:

- The *function*
- *Preconditions* for the function to be realized
- *Main flow* of events in the function (scenario)
- *Error conditions* and the description of *alternative flows*
- Information about other *concurrent activities*
- *Postconditions* determined by the realization of the function

- Feasibility study
- Requirements elicitation and analysis
- Requirements validation
- Requirements management

Requirements analysis

Use case

Represents ***a service (function)*** offered by the system in its context composed of actors.

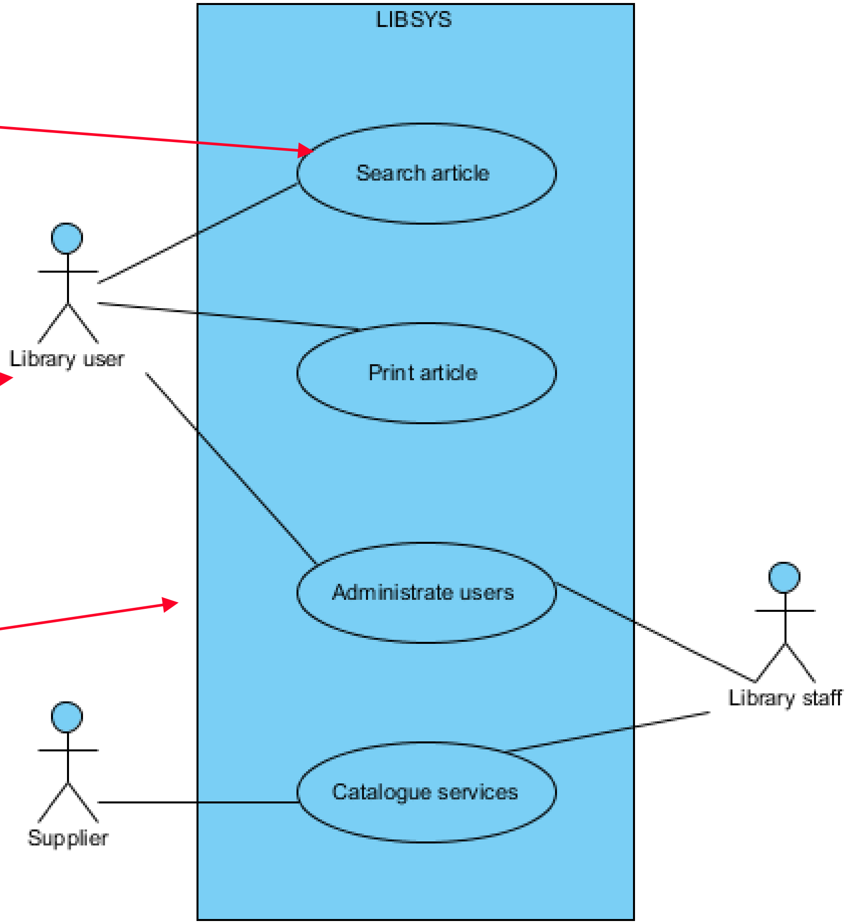
Defines the ***requirements*** of WHAT the system must perform. (functional requirements).

Actor

Any external entity that interacts with the system through an interface.

System boundary

Delineates what is included in the system from what is outside the system and interacts with it.



Adapted after ©Ian Sommerville **Engineering oftware Products** (2019) *and* **Software Engineering, 9th ed.** (2010)**. Chapter 4**

Requirements analysis –
- prioritization

Based on *development constraints* and on *requirements sources*.

Constraints:

- Limited resources
- Limited time
- Limited technical capabilities

Requirements with *higher priority* are *developed and released first*.

Criteria (examples):

- Current customer demands
- Competition and current market condition
- Future and new customer needs
- Immediate sales advantage
- Critical problems in the existing product, etc.

Requirements analysis –
- traceability

Motivation: Ability to test after development and to verify that all requirements have been developed, tested, packed and delivered.

Precondition: requirements are uniquely identified.

Types of traceability:

Backward from –requirement is linked to the source document or to the person which created it.

Forward from – requirement is linked to the design and implementation.

Backward to – design and implementation are linked to the requirement.

Forward from – preceding documents are linked to the requirement.

Requirement relationships matrix – requirements correlation.

Requirements validation

Concerned with demonstrating that the requirements define the system that the customer really wants.

Requirements error costs are high so validation is very important

- Fixing a requirements error after delivery may cost up to 100 times the cost of fixing an implementation error.

Requirement checking

Verifiability. Can the requirements be checked?

Comprehensibility. Is the requirement properly understood?

Traceability. Is the origin of the requirement clearly stated?

Adaptability. Can the requirement be changed without a large impact on other requirements?

Consistency. Are there any requirements conflicts?

Realism. Can the requirements be implemented given available budget and technology?

Validity. Does the system provide the functions which best support the customer’s needs?

Completeness. Are all functions required by the customer included?

Requirements validation techniques

Requirements reviews

- Systematic manual analysis of the requirements.
  - Regular reviews held while the requirements definition is being formulated.
  - Both client and contractor staff should be involved in reviews.
  - Reviews may be formal (with completed documents) or informal.
  - Good communications between developers, customers and users can resolve problems at an early stage.

Prototyping

- Using an executable model of the system to check requirements.

Test-case generation

- Developing tests for requirements to check testability.

Obs. These techniques can be used in conjunction or individually.

Requirements management

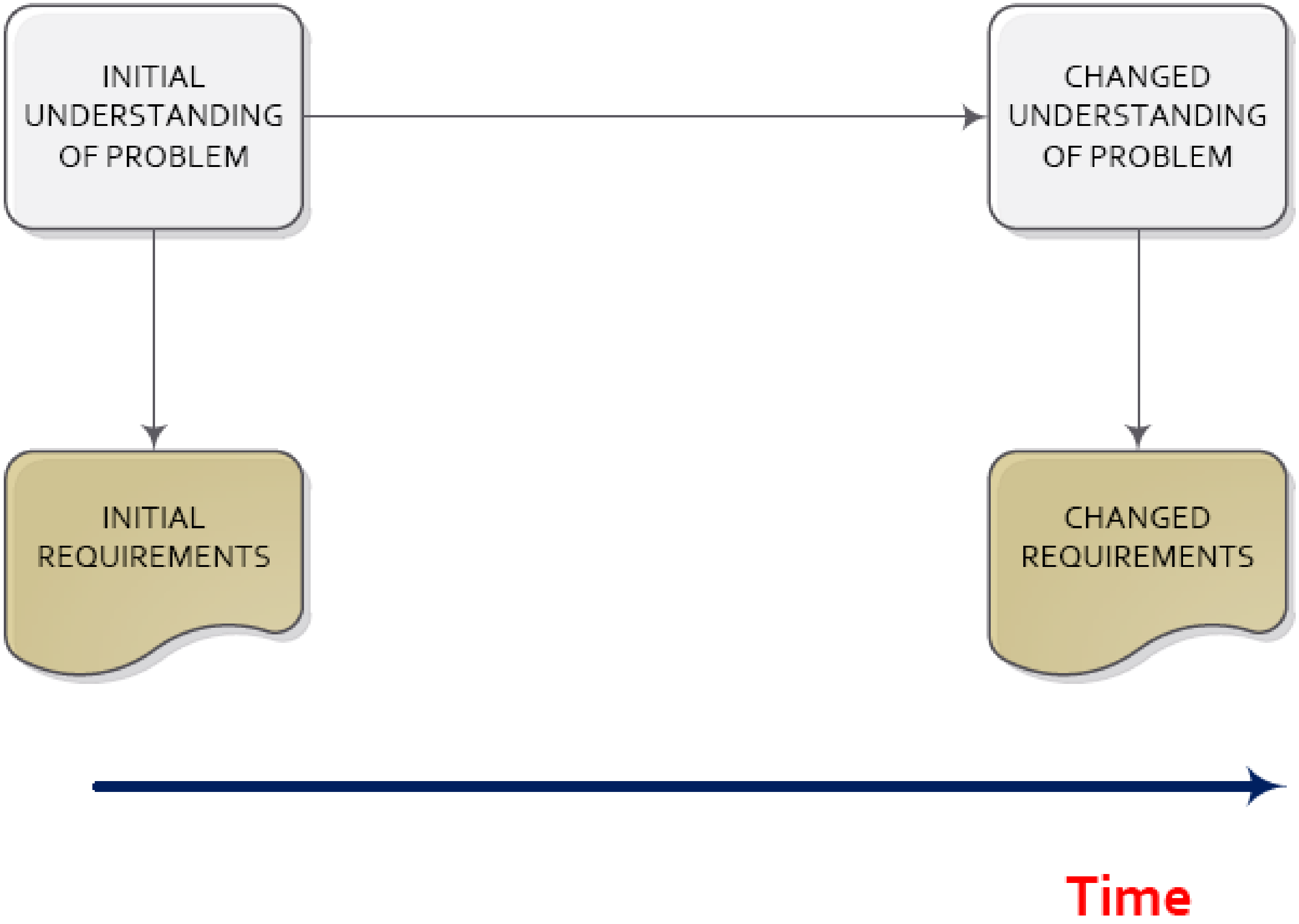
Def. Requirements management is the process of managing ***changing*** requirements during the requirements engineering process and system development.

Examples of causes for change in project-based software:

- The priority of requirements from different viewpoints changes during the development process.
- System customers may specify requirements from a business perspective that conflict with end-user requirements.
- The business and technical environment of the system changes during its development.

- Feasibility study
- Requirements elicitation and analysis
- Requirements validation
- Requirements management

Requirements evolution


INITIAL UNDERSTANDING OF PROBLEM
CHANGED UNDERSTANDING OF PROBLEM
INITIAL REQUIREMENTS
CHANGED REQUIREMENTS
Time

Enduring and volatile requirements

Enduring requirements. Stable requirements derived from the core activity of the customer organisation.

May be derived from domain models.

Example: a hospital will always have doctors, nurses, etc.

Volatile requirements. Requirements which change during development or when the system is in use.

Example: In a hospital, the requirements derived from health-care policy.

Requirements change management

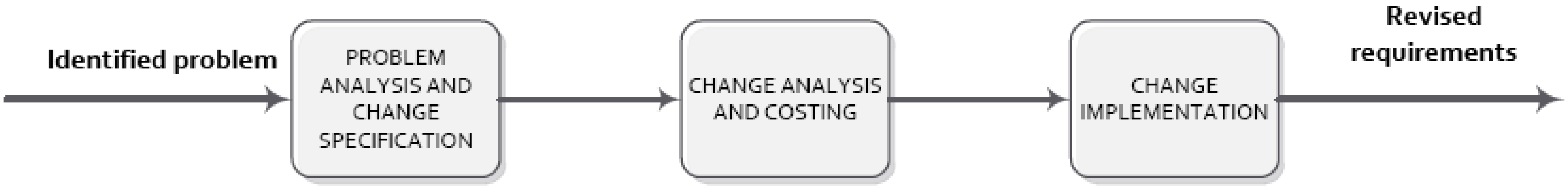
Should apply to all proposed changes to the requirements.

Main activities:

Problem analysis. Discuss requirements problem, check its validity and propose change;

Change analysis and costing. Assess effects of change on other requirements;

Change implementation. Modify requirements document and other documents to reflect change. Recommandation: minimize external references and make the document sections as modular as possible.


Identified problem
PROBLEM ANALYSIS AND CHANGE SPECIFICATION
CHANGE ANALYSIS AND COSTING
CHANGE IMPLEMENTATION
Revised requirements

Formative evaluation

1. Check the activities implied in requirements elicitation and analysis.

https://forms.gle/jDpfz54XV1a3oiDp6

Topics covered

- Software requirements
- Software requirements document
- Requirements engineering process for project-based software
- Requirements engineering for software products

Product feasibility

Feasibility decision based on :

product vision:

- WHAT is the product to be developed?
- WHO are the target customers and users?
- WHY should customers buy this product?

Example : ***iLearn*** system

FOR teachers and educators

WHO need a way to help students use web-based learning resources and applications,

THE iLearn system is an open learning environment

THAT allows the set of resources used by classes and students to be easily configured for these students and classes by teachers themselves.

UNLIKE Virtual Learning Environments, such as Moodle, the focus of iLearn is the learning process rather than the administration and management of materials, assessments and coursework.

OUR product enables teachers to create subject and age-specific environments for their students using any web-based resources, such as videos, simulations and written materials that are appropriate.

Features (requirements) discovery

Involves identifying:

- product FEATURES* useful for target users (mostly functional requirements)
- what users like and dislike about the products that they use (related mostly to extra-functional requirements)

Participants : technical staff (DEVELOPER)

Methods :

- informal user analysis and discussions about users work, the software they use, its strengths and weaknesses
- scenario analysis with users to clarify scenarios and make them more realistic
- teamwork (to identify features that crosscut more scenarios)

Tools :

Scenarios and user stories, used to stimulate thinking about the product

*FEATURE = fragment of functionality of a software product, that implements a requirement.

Features (requirements) discovery

Feature = fragment of *functionality* of a software product, that *implements a requirement* (something that user or system needs).

- Examples : ‘print’ feature, ‘change background feature’, ‘new document’ feature

List of features = starting point for product design and development

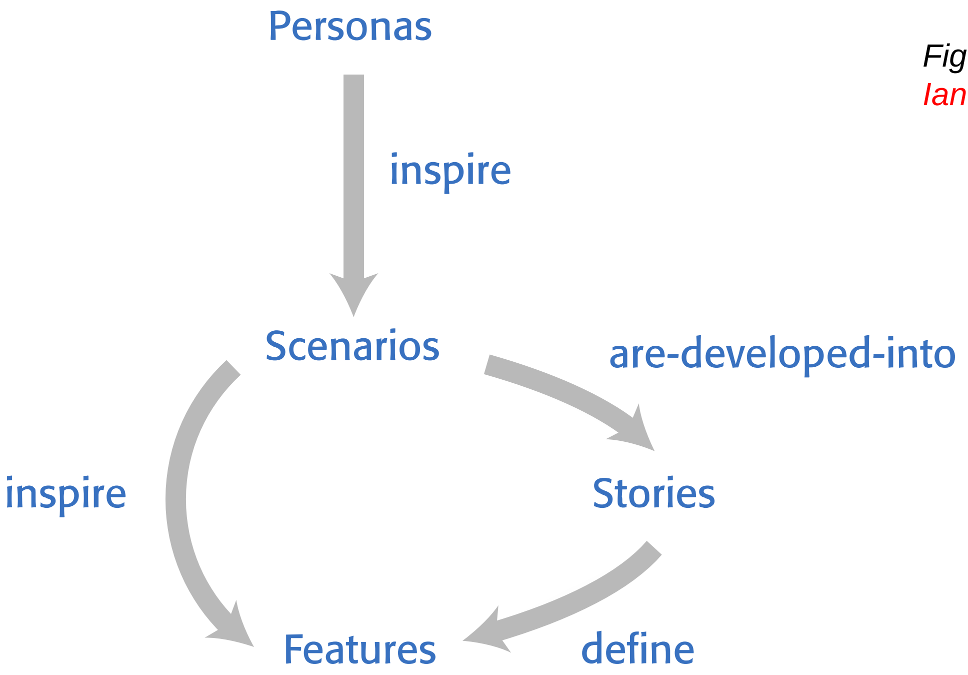


Figure 3.1 From Personas to Features
Ian Sommerville – Engineering Software Products

Personas

(from Spanish: individuals implied in communication)

Aim : get an understanding of potential users of the software product and envisage difficulties that they might have in understanding and using product features.

Persona (type of user) = '*imagined user*' for which character portrait is created.

Examples : Personas **in iLearn** system :

Jack, a primary school teacher

Jack, age 32, is a primary school (elementary school) teacher in Ullapool, a large coastal village in the Scottish Highlands. He teaches children from ages 9-12. He was born in a fishing community north of Ullapool, where his father runs a marine fuels supply business and his mother is a community nurse. He has a degree in English from Glasgow University and retrained as a teacher after several years working as a web content author for a large leisure group.

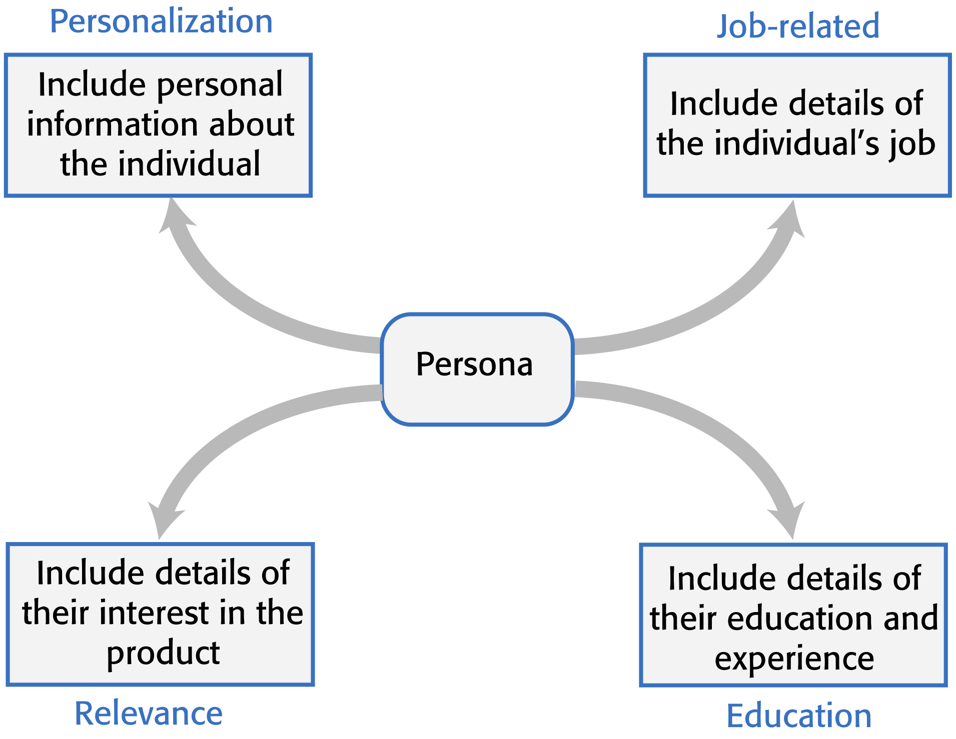
Jack's experience as a web developer means that he is confident in all aspects of digital technology. He passionately believes that the effective use of digital technologies, blended with face to face teaching, can enhance the learning experience for children. He is particularly interested in using the iLearn system for project-based teaching, where students work together across subject areas on a challenging topic.

- ***Emma, a history teacher***
- ***Elena, a school IT technician***

Persona description

Name
Personal circumstances (think the persona as an individual).
Photograph (optional)

Figure 3.4 Persona Descriptions
Ian Sommerville – Engineering Software Products


Personalization
Include personal information about the individual
Job-related
Include details of the individual's job
Persona
Include details of their interest in the product
Relevance
Include details of their education and experience
Education

Information about their job and (if necessary) what that job involves.

The motivation for being interested in using the product.
What they might want to do with it.

Describe educational background and level of technical skills and experience.
(important, especially for interface design).

Persona benefits

Helps :

- development team members empathize with potential users of the software.
- as a tool that allows developers to ‘step into the user’s shoes’.
  - imagine how a persona, not a developer, would behave and react
- check ideas and make sure not to include features that aren’t really needed.
- avoid making unwarranted assumptions, based on developers’ own knowledge, and designing an over-complicated or irrelevant product.

Deriving personas

Based on an understanding of the potential product users, their jobs, their background and their aspirations.

1. *Study and survey potential users* to understand what they want and how they might use the product.

2. *Abstract the essential information* about the different types of product user and use this as a basis for creating personas.

A light variant :

Proto-personas : developed on the basis of limited user information

- may be created as a collective team exercise using whatever information is available about potential product users.
- not as accurate as personas developed from detailed user studies.

Scenarios

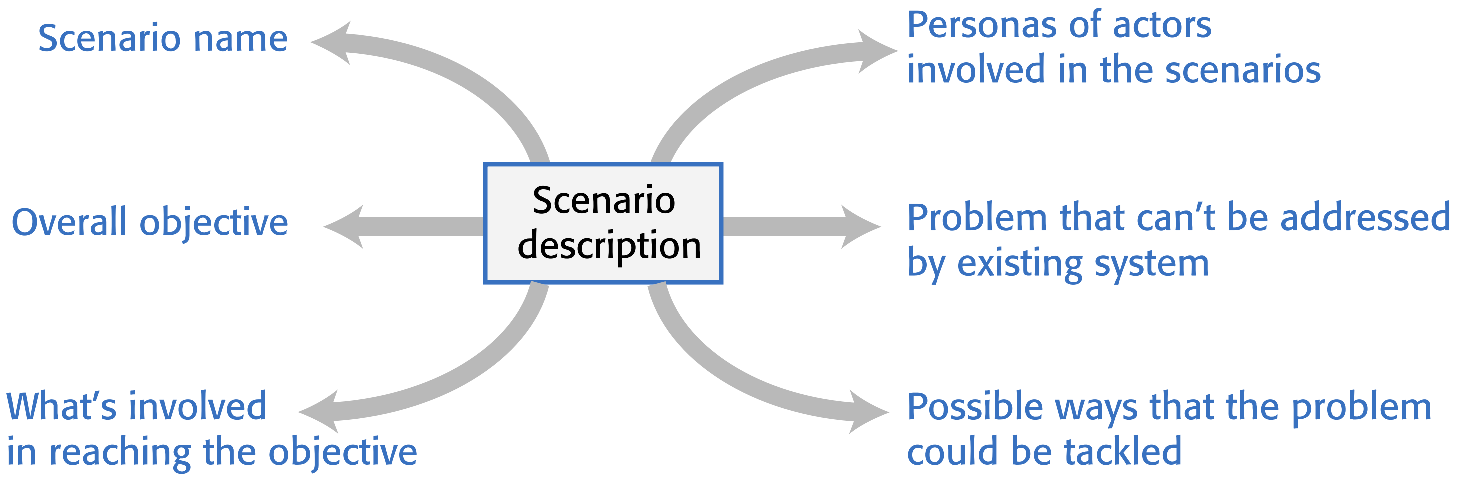
Scenario = narrative that describes a situation where a user, or a group of users, use product’s features to do something that they want to do.

- Tool (utility) of facilitating communication and stimulating design creativity.
- Effective in communication - understandable and accessible to users and to people responsible for funding and buying the system.
- Help developers to agree on a shared understanding of the system that they are creating.
- Are NOT specifications.
- Lack detail, may be incomplete, may not represent all types of user interactions

Scenarios

Figure 3.5 Elements of a scenario description

Ian Sommerville – Engineering Software Products



Scenario name
Personas of actors involved in the scenarios
Overall objective
Scenario description
Problem that can't be addressed by existing system
What's involved in reaching the objective
Possible ways that the problem could be tackled

Writing scenarios

- Describe :
  - A user problem
  - A imagined way to solve it

- From the user's perspective and based on identified personas or on real users.

- Starting point : the personas created
  - imagine several scenarios from each persona.

- Ideally, general and without implementation information.

(However, describing an ideea of implementation is often the easiest way to explain how a task is done).

- Make sure to cover all the potential user roles when describing a system.

Scenarios

Example : ***Fishing in Ullapool***

Jack is a primary school teacher in Ullapool, teaching P6 pupils. He has decided that a class project should be focused around the fishing industry in the area, looking at the history, development and economic impact of fishing.

As part of this, students are asked to gather and share reminiscences from relatives, use newspaper archives and collect old photographs related to fishing and fishing communities in the area. Pupils use an iLearn wiki to gather together fishing stories and SCRAN (a history archive site) to access newspaper archives and photographs. However, Jack also needs a photo-sharing site as he wants students to take and comment on each others' photos and to upload scans of old photographs that they may have in their families. He needs to be able to moderate posts with photos before they are shared, because pre-teen children can't understand copyright and privacy issues.

Jack sends an email to a primary school teachers' group to see if anyone can recommend an appropriate system. Two teachers reply and both suggest that he uses KidsTakePics, a photo-sharing site that allows teachers to check and moderate content. As KidsTakePics is not integrated with the iLearn authentication service, he sets up a teacher and a class account with KidsTakePics.

He uses the the iLearn setup service to add KidsTakePics to the services seen by the students in his class so that, when they log in, they can immediately use the system to upload photos from their phones and class computers.

User stories

Scenarios : high-level stories of system use; describe an undetailed sequence of interactions with the system.

User stories = finer-grain narratives that set out in a more detailed and structured way a single thing that a user wants from a software system.

The standard format of a user story:

As a <role>, I <want | need> **to** <do something>

As a teacher, I want **to** tell all members of my group when new information is available

Variant of the standard format (adds a justification for the action):

As a <role> I <want | need> **to** <do something> **so that** <reason>

As a teacher, I need to be able **to** report who is attending a class trip **so that** the school maintains the required health and safety records.

User stories in planning

One important use - in planning.

Ex. product backlog : set of user stories.

- *User story* - Clearly defined system feature or aspect of a feature that can be implemented within a single development time unit (ex. sprint).

Ex. **As a** teacher, I need **to** be able to report who is attending a class trip **so that** the school maintains the required health and safety records.

- *Epic* – is more complex and should be broken down into simpler stories.

Ex. **As a** system manager, I need a way **to** backup the system and restore either individual applications, files, directories or the whole system.

Example of templates:

https://www.aha.io/roadmapping/guide/requirements-management/what-is-a-good-feature-or-user-story-template

Features

Feature *properties:*

Independence

- independ on how other system features are implemented
- not affected by the order of activation of other features.

Coherence

- linked to a single item of functionality.
- does only one thing
- does not have side-effects.

Relevance

- reflect the way that users normally carry out some task.
- does not provide obscure functionality that is hardly ever required.

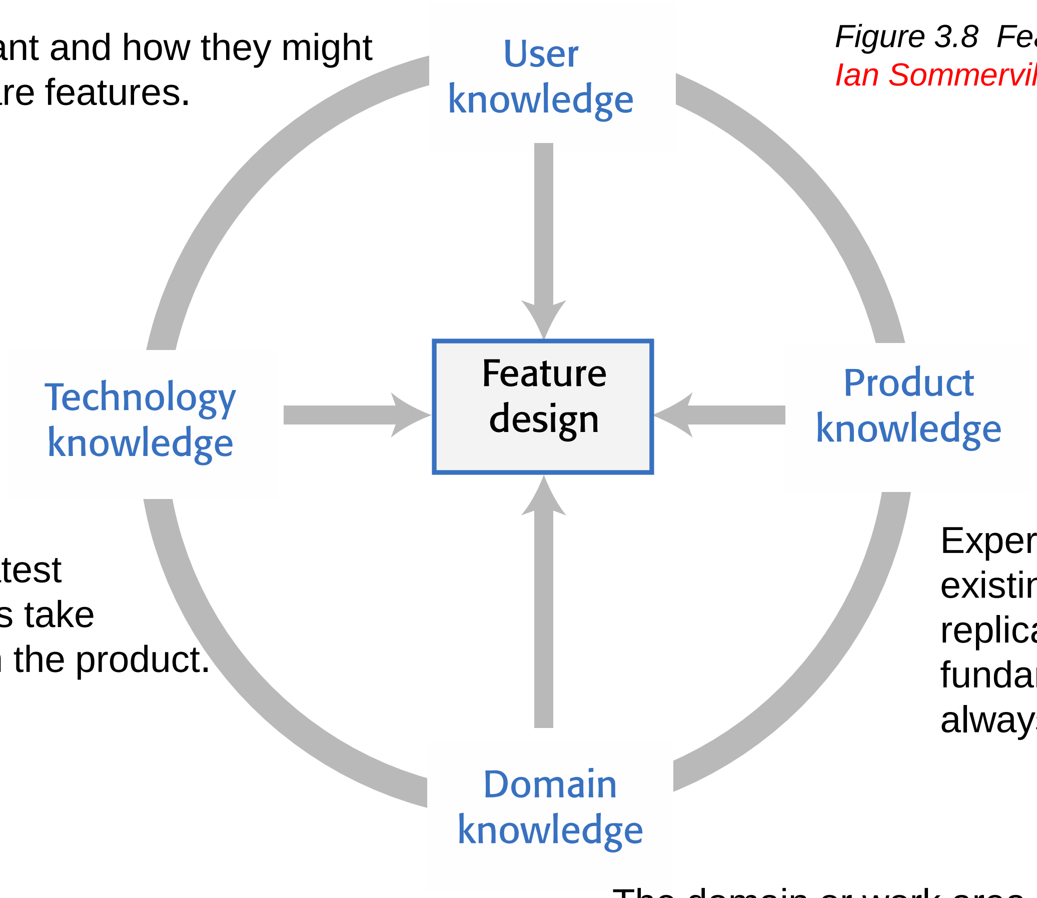
List of features

- Defines the overall functionality of the software
- Is the output of the feature identification process
- Used for designing and implementing the product.
- Details added when implementing the feature.
- Described with a standard input-action-output template or by a structured narrative that may include a set of user stories.

Knowledge required for feature design

What users want and how they might use the software features.

Figure 3.8 Features design
Ian Sommerville – Engineering Software Products



Understanding latest technology allows take advantage of it in the product.

Experience or research of the existing product to identify and replicate features that provide fundamental functionality that is always required.

The domain or work area that product aims to support.

Factors in feature design

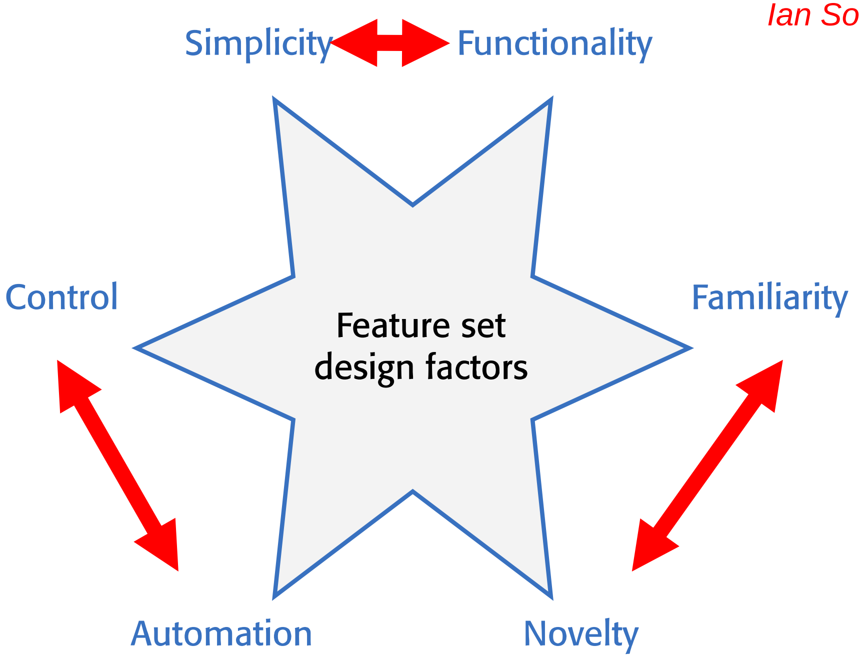


Figure 3.9 Factors in feature design

Ian Sommerville – Engineering Software Products

⟷ Trade-off – between elements that can not be simultaneously maximized.

Feature derivation

Sources

- Product vision
- Scenarios
- User stories

Method:

Highlight phrases in narrative description thinking about the features needed to support *user actions*, identified by *active verbs* (ex. use, choose, send, update, etc.).

Team activity - discuss features and generate ideas about :

- new, related features,
- generalizing features.

Feature derivation

Example : from product vision

FOR teachers and educators WHO need a way *to help students use web-based learning resources and applications*, THE iLearn system is an open learning environment THAT *allows the set of resources used by classes and students to be easily configured for these students and classes by teachers themselves*.

UNLIKE Virtual Learning Environments, such as Moodle, the focus of iLearn is the learning process itself, rather than the administration and management of materials, assessments and coursework. OUR product *enables teachers to create subject and age-specific environments for their students* using any web-based resources, such as videos, simulations and written materials that are appropriate.

1. A feature that allows users to access and use existing web-based resources;
2. A feature that allows the system to exist in multiple different instantiations;
3. A feature that allows user configuration of the system to create a specific instantiation.

Feature derivation

Example : from a scenario

Jack is a primary school teacher in Ullapool, teaching P6 pupils. He has decided that a class project should be focused around the fishing industry in the area, looking at the history, development and economic impact

As part of this, students are asked to gather and share reminiscences from relatives, use newspaper archives and collect old photographs related to fishing and fishing communities in the area. *Students use an iLearn wiki* to gather together fishing stories and *SCRAN (a history archive) to access newspaper archives and photographs*. However, Jack also needs a photo-sharing site as he wants *pupils to take and comment on each others' photos* and to *upload scans of old photographs* that they may have in their families. He needs to be able to moderate posts with photos before they are shared, because pre-teen children can't understand copyright and privacy issues.

Jack *sends an email to a primary school teachers' group*, which he is a member of to see if anyone can recommend an appropriate system. Two teachers reply and both suggest that he uses KidsTakePics, a photo-sharing site that allows teachers to check and moderate content. As KidsTakePics *is not integrated with the iLearn authentication service*, he sets up a teacher and a class account with KidsTakePics.

He uses the the iLearn setup service to add KidsTakePics to the services seen by the students in his class so that when they log in, they can immediately use the system to upload photos from their phones and class computers.

1. A wiki for group writing.
2. Access to the SCRAN history archive.
3. Features to set up and access an e-mail group.
4. A feature to integrate applications with the iLearn authentication service.

Configuration feature has already been identified from product vision.

New related features and feature generalization

Example :

Initial feature :

1. A wiki for group writing.

New **related** features :

Age-appropriate ways for collaborative writing

1.1 Collaborative writing, where several people can work simultaneously on the same document.

1.2 Blogs and web pages as a way of sharing information

Example :

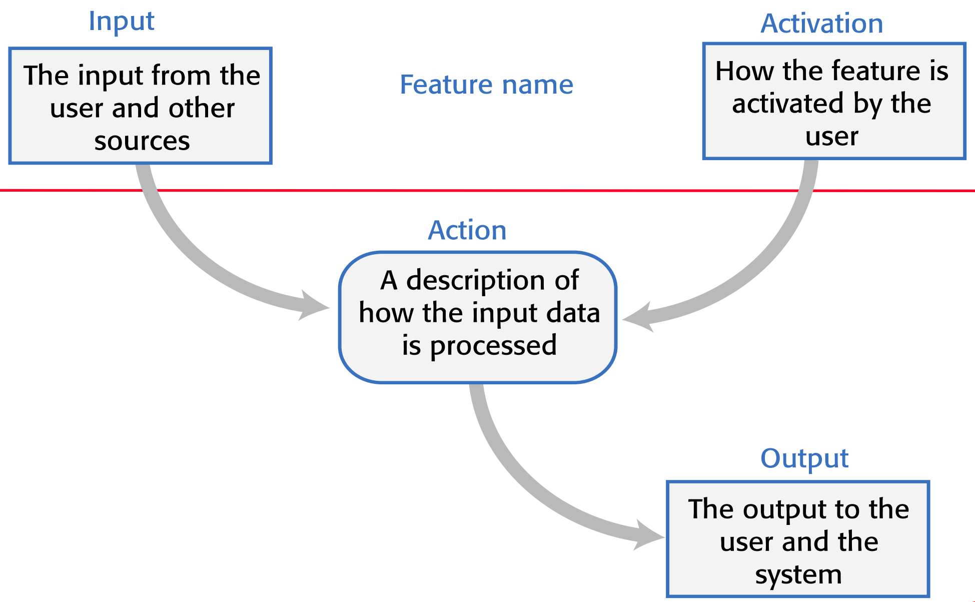
Initial feature :

2. Access to the SCRAN history archive.

Generalized feature :

2. Access to the ***any external*** (history) archive.

Features (requirements) description : INPUT-ACTION-OUTPUT



Input

The input from the user and other sources

Feature name

Activation

How the feature is activated by the user

Action

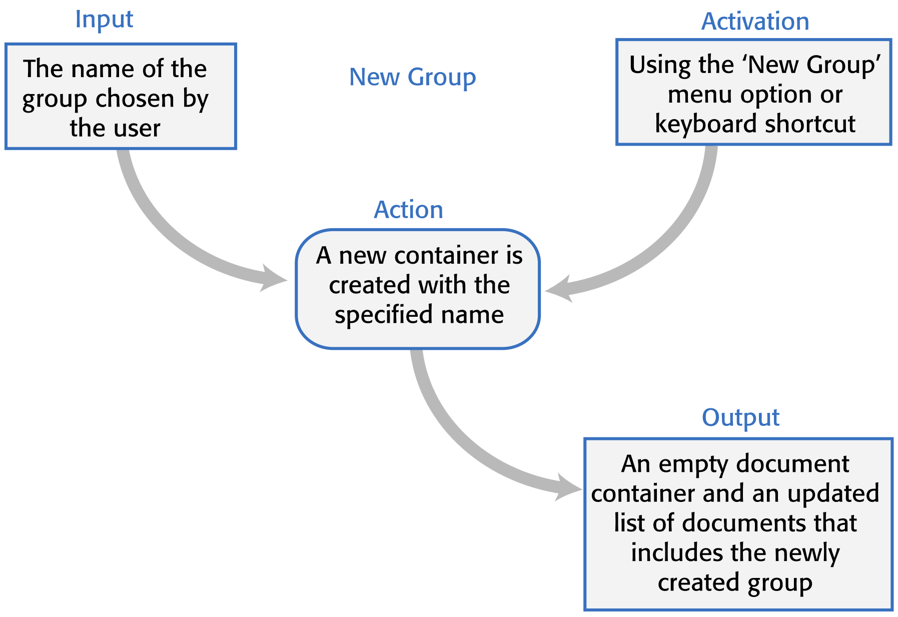
A description of how the input data is processed

Output

The output to the user and the system

Figure 3.2 Feature description

Ian Sommerville – Engineering Software Products



Input

The name of the group chosen by the user

New Group

Activation

Using the 'New Group' menu option or keyboard shortcut

Action

A new container is created with the specified name

Output

An empty document container and an updated list of documents that includes the newly created group

Figure 3.3 The 'New Group' feature description

Ian Sommerville – Engineering Software Products

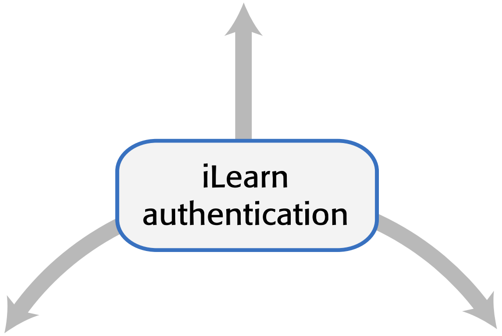
Adapted after ©Ian Sommerville **Engineering oftware Products** (2019) *and* **Software Engineering, 9th ed.** (2010)**. Chapter 4**

Features (requirements) description : STRUCTURED NARATIVE DESCRIPTION Example

Figure 3.11 The iLearn authentication feature

Ian Sommerville – Engineering Software Products

Description
Authentication is used to identify users to the system and is currently based on a login id/password system. Users may authenticate themselves using their national user id and a personal password or may use their Google or Facebook credentials.



Constraints
All users must have a national user id and system password that they use for initial system authentication. They may then link their account with their Google/Facebook account for future authentication sessions.

Comments
Future authentication mechanisms may be based on biometrics and this should be considered in the design of the system.

Features (requirements) description : STRUCTURED NARATIVE DESCRIPTION with USER STORIES
Example

For relatively long feature description.

feature

Description with more user stories

As a system manager, I want **to** create and configure an iLearn environment by adding and removing services to/from that environment **so that** I can create environments for specific purposes.

As a system manager, I want **to** set up sub-environments that include a subset of services that are included in another environment.

As a system manager, I want **to** assign administrators to created environments.

As a system manager, I want **to** limit the rights of environment administrators **so that** they cannot accidentally or deliberately disrupt the operation of key services.

As a teacher, I want **to** be able to add services that are not integrated with the iLearn authentication system.

Constraints

The use of some tools may be limited for license reasons so there may be a need to access license management tools during configuration.

Comments

Based on Elena’s and Jack’s scenarios

Features (requirements) description : NARATIVE DESCRIPTION STRUCTURING and USER STORIES Structuring examples

<qualifier(s)> <verb> <qualifier(s)> <noun> <condition(s)>

Ex: Automatically cancel accommodation reservation if the guest is a no-show.

Its representation as user story :

As a front desk clerk, **I want** the system to automatically cancel an accommodation reservation if the guest is a no-show **so that** the rooms are immediately made available for future reservations and walk-in guests**.**

OBS. Nouns must be defined in the glossary
Accommodation: A temporary place to stay. May be provided by a room, suite, chalet or dormitory.
Guest: A person who occupies accommodation at a hotel.
Reservation: The arrangement by which accommodation is secured in advance of a stay.

Verb-noun style
https://www.linkedin.com/pulse/describing-software-features-phil-robinson

<action> **the** *<result>* **by|for|of|to** *<object>*

Ex.: Create **the** container **for** documents or groups.

FDD (Feature Driven Development) style
Ian Sommerville – Engineering Software Products

Features demonstration

Objective :

- Demonstrate (new) identified features
- Discover problems, omissions, inconsistencies in the list of features

Method :

- Create a prototype system or extend an existing prototype.
- Focus on novel and critical feature of the system

Result :

- Updated feature list

Creative innovation

A feature set is identified from scenarios and user stories.

Scenarios and user stories based on the research on users result in an understanding of how the software might be used.

The development team may obtain future improvement by creatively think about alternative or additional features that help users to work more efficiently or to do things differently.

Formative evaluation

1. Why is important to create Personas ?

2. Which are the sources of information for deriving the product features?

3. Consider this simple feature description : “Grade quiz”. Extend it with a possible example of qualifications and condition (verb-noun style). Then describe it with a user story (represented in standard format), where the user is “teacher”.

https://forms.gle/wePLGxnozFHMfHaA7

Key points

Software requirements specify *services* and *qualities* that the customer requires from a system, and *constraints* under which system *operates* and *is developed*.

There are two basic orthogonal classifications for software requirements : functional / extra-functional and user / system.

Requirements engineering process is an iterative process including requirements *elicitation*, *analysis and specification*, *validation*, and *management*.

Requirements engineering process has the main objective to create and maintain a *document* containing all *requirements specifications*.

Key points

A *software product feature* is a fragment of functionality that implements a requirement.

The first stage of product development is to *identify the list of product features* in which each feature is identified and gets a brief description of its functionality.

Personas are *‘imagined users*’ for which is created a character portrait of a type of user that is supposed to use the product.

A *persona description* should ‘paint a picture’ of a typical product user. It should describe their *educational background, technology experience* and why they might *want to use* the product.

A *scenario* is a narrative that describes a situation where a user is accessing product features to do something that they want to do.

Key points

Scenarios should always be *written from the user's perspective* and should be based on identified personas or on real users.

User stories are finer-grain narratives that set out, in a *structured way*, something that a user wants from a software system.

User stories may be used as a way of *extending and adding detail* to a scenario or as part of the description of system features.

The key influences in feature identification and design are *user research, domain knowledge, product knowledge,* and *technology knowledge*.

Features can be identified from scenarios and stories by highlighting *user actions* in these narratives and thinking about the features that are needed to *support these actions*.