Requirement Engineering

TELECOM Nancy 2022-2023

Sources

- Software Engineering 10 - Ian Sommerville - Pearson
- Applying UML and Patterns - Craig Larman
- Software Requirements (Third Edition) - Wiegers and Beatty
- Requirement Engineering – Dick, Hull and Jackson 2017 – Springer
- Software Engineering A Practitioner’s Approach (Ninth Edition) – Pressman - Maxim

Topic covered

- Definition of requirements
- Requirement Engineering
- Requirement Process
  - Elicitation
  - Specification
  - Validation
  - Change

What is a requirement ?

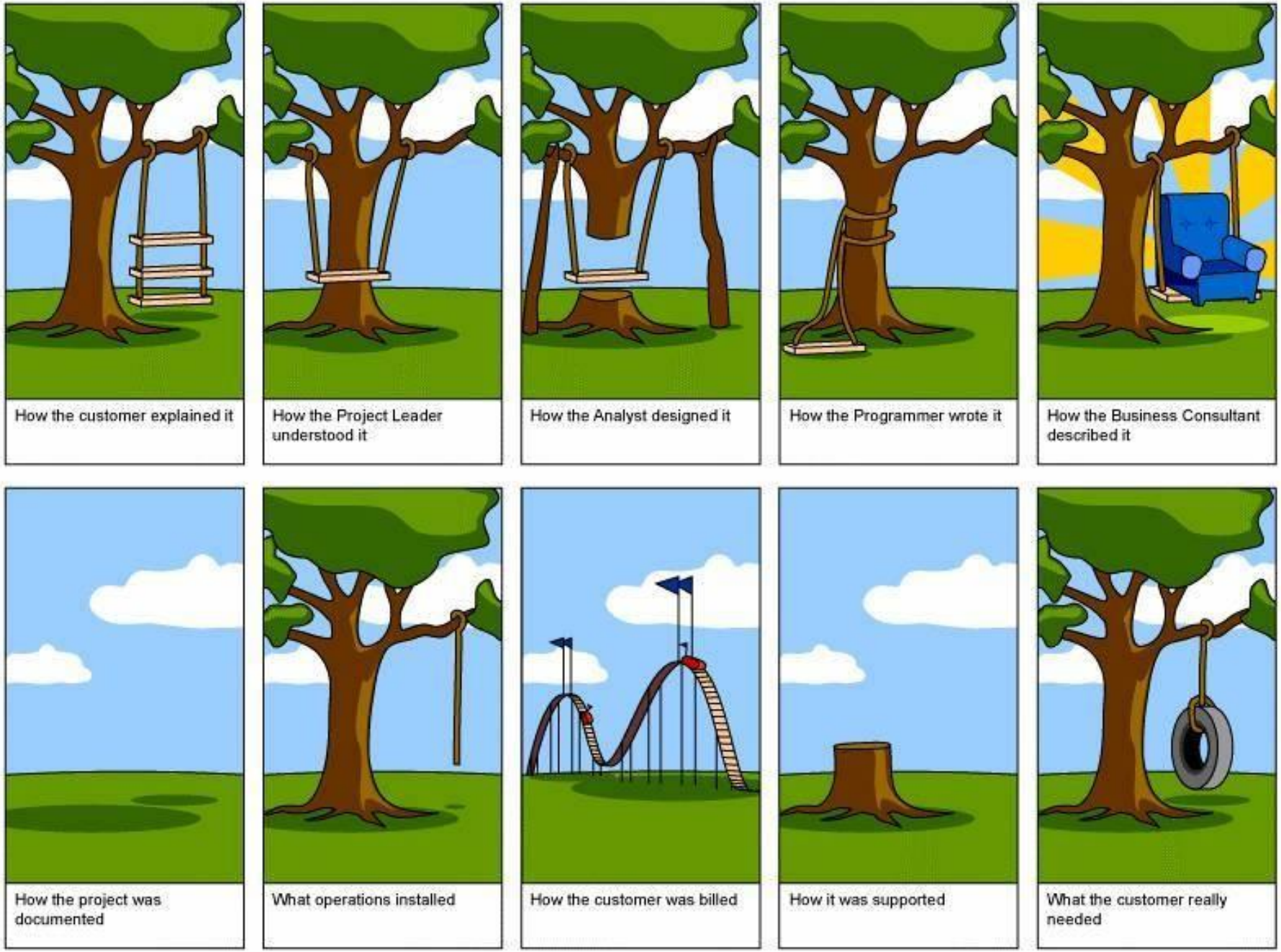
- The IEEE Standard Glossary of Software Engineering Terminology defines a requirement as:[1]
  - A condition or capability needed by a user to solve a problem or achieve an objective.
  - A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document.
  - A documented representation of a condition or capability as in 1 or 2.

IEEE Computer Society (1990). "IEEE Standard Glossary of Software Engineering Terminology". *IEEE Standard*.

Requirements in practice

- A user describes his/her needs, requirements specification to in a specification document
- The software developer take the specification document and produce a piece of software thet meet these needs perfectly

- What can go wrong with this story ?


How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed

Requirement

- Description of the required functionnality
  - Un client peut ajouter un produit dans son panier
- Details of realisation
  - Le client a accès la liste des produits disponibles
  - Quand il ajoute un produit le total de son panier se met à jour
  - Si sa commande dépasse l'avoir de son porte monnaie, le total passe en rouge
- Requirements may have different priorities
  - High, medium, low
  - Shall vs should

Example from 2021

- Requirements

Properties of a requirement

- Unambiguous
- Verifiable
- Clear
- Correct
- Understandable
- Feasible
- Independent
- Atomic
- Necessary
- Implementation-free
- Consistent
- Non Redundant
- Complete

https://www.informit.com/articles/article.aspx?p=1152528&seqNum=4

Unambiguous

- Only one way to interpret the requirement

- *L'application doit permettre au client de modifier leur commande à tout moment*
  - Que signifie modifier une commande
  - Que signifie à tout moment

Verifiable

- It should be possible to test that a requirement is implemented correctly
- Be careful to word that are difficult to test : efficient, simple, user-friendly, flexible, quickly

  - *Le système doit permettre aux enseignants d'ajouter des absences de façon simple.*

Clear *(Simple, Précis, Concis)*

- Requirements must not containt unnecessary information

  - *Le système doit permettre à la scolarité de saisir une justification apportée par un étudiant en fonction de la date d'absence et de l'ajouter aux justifications d'absence pour faire en sorte que l'absence de l'étudiant ne soit pas comptée et que cela ne nuise pas à sa scolarité.*
  - *Le système doit permettre à la scolarité de saisir une justification d'absence apportée par un étudiant*

Correct

- If a requirement contains facts, they must be true
  - *Une TVA de 20% doit être ajoutée au montant de la facture*

Understandable

- Requirements must be grammatically correct and written in a consistent style
  - *Le système doit permettre aux membres bar de faire des modications des comptes clients notamment changer un compte de categorie.*
  - *Il faut pouvoir choisir parmi de nombreux produits*

Feasible

- The requirement must be doable within the constraints of the project (time, money, resources)
  - *Le système doit permettre l'authentification des utilisateurs par reconnaissance faciale*

Independent

- To understand the requirement, there should not be a need to know any other requirement:
  - *Le système permet à un utilisateur d'ajouter un produit à la commande dans une quantité donnée*
  - *Le montant de la commande est calculé au fur et à mesure.*

Atomic

- The requirement should contain a single traceable element: It will be difficult to trace a requirement that includes several requirements

  - *Le système permet au responsible du bar de gérer les stocks*

Necessary

- A requirement is unnecessary if
  - None of the stakeholders needs the requirement or
  - Removing the requirement will not affect the system.

  - *Le système doit permettre à un client du bar d'envoyer des messages*

Implementation-Free

- Requirements should not contain unnecessary design and implementation information:
  - *Le système doit stocker les informations dans un fichier csv*

Consistent

- There should not be any conflicts between the requirements. Conflicts may be direct or indirect. Direct conflicts occur when, in the same situation, different behavior is expected:

  - *Le système doit permettre de changer sa commande à tout moment*
  - *Le système doit permettre au Barman de valider une commande. Une fois validée elle ne peut plus être modifiée*

Non-Redundant

- Each requirement should be expressed only once and should not overlap with another requirement:

  - *Le système doit permettre au trésorier de changer les prix des produits*
  - *Le système permet au trésorier de mettre à jour le prix des produits lors de la reception de commandes*

Complete

- A requirement should be specified for all conditions that can occur:

  - *Le système doit permettre aux responsables du bar de valider une commande servie au fur et a mesure.*

Stakeholders

- Any person or organization who is affected by the system in some way and so who has a legitimate interest, direct or indirect
- Stakeholder types
  - End users
  - System managers
  - System owners
  - External stakeholders

Functional vs non-functional requirements

- Functional Requirements
- Non-Functional Requirements
- Domain requirements

Functional and non-functional requirements

- Functional requirements
  - Statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations.
  - May state what the system should not do.
- Non-functional requirements
  - Constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc.
  - Often apply to the system as a whole rather than individual features or services.
- Domain requirements
  - Constraints on the system from the domain of operation

Functional requirements

- Describe functionality or system services.
- Depend on the type of software, expected users and the type of system where the software is used.
- Functional user requirements may be high-level statements of what the system should do.
- Functional system requirements should describe the system services in detail.

Exemple – prise de rendez vous

Requirement 10

Description : Le système doit permettre à un client de déplacer un rendez-vous. (RC11)
Priorité : **low**
Détails, contraintes et conséquences

1. Le client en consultant ses rendez-vous peut choisir de déplacer un rendez-vous via un bouton de déplacement du rdv.
2. Une fenêtre apparaît proposant d'autres créneaux avec la même prestation, le même salon, le même prix. Si les choix ne satisfont pas le client, celui-ci doit annuler son rdv pour accéder à d'autres propositions de prestations ou de salons.

Requirement 19

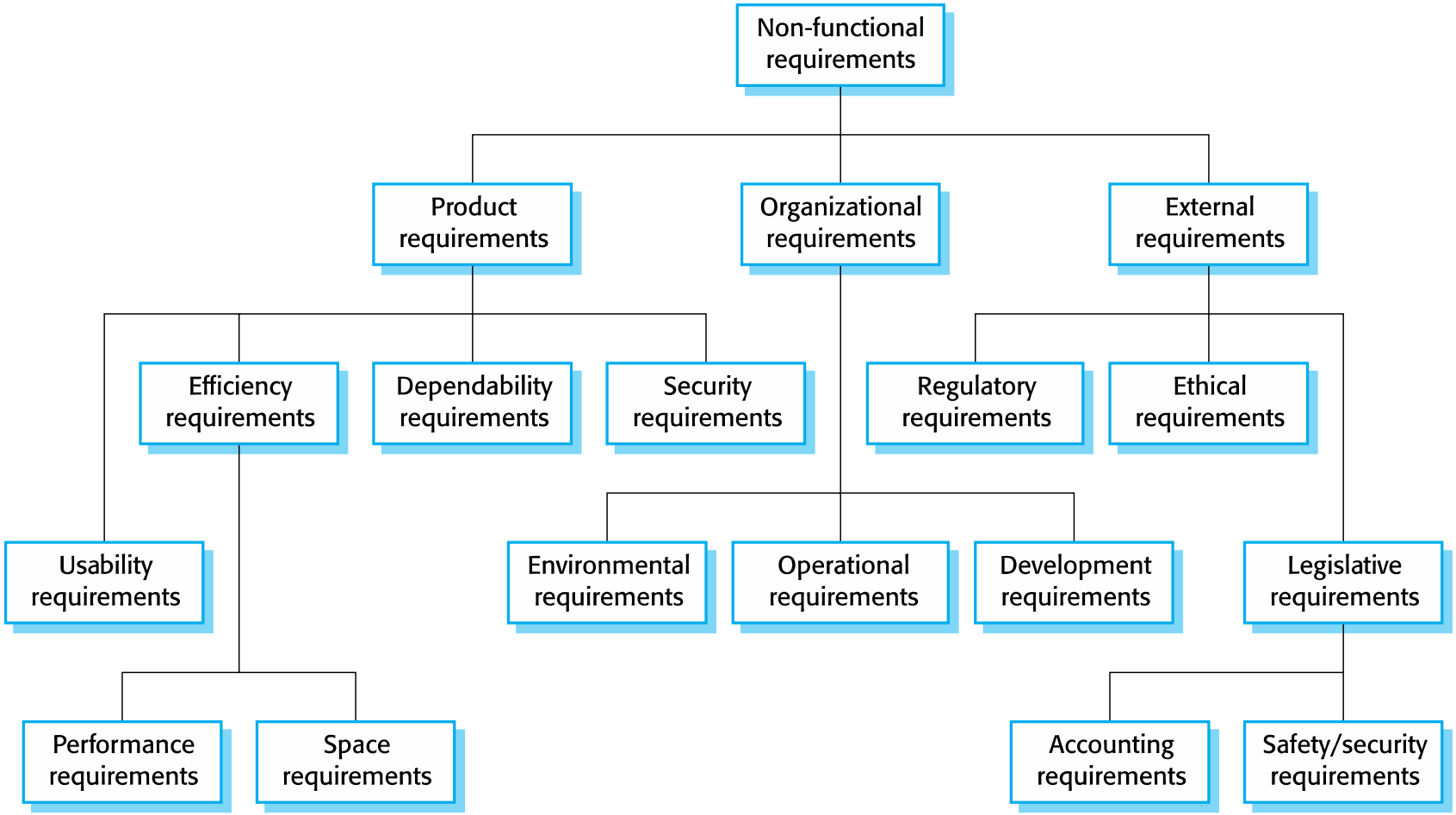
Description : Le système permet au responsable de consulter la liste des rendez-vous. (RR9)
Priorité : **high**
Détails, contraintes et conséquences

1. Une page doit être consacrée à l'affichage de la liste de tous les rendez-vous avec le salon.
2. Cette liste doit pouvoir être triée par un ordre particulier ou par des filtres (chronologie, prestation, prix, employé, matériel, durée).
3. Le responsable peut consulter la liste des rendez-vous sous un format de planning (affichage de la semaine, du mois).

Non-functional requirements

- These define system properties and constraints e.g. reliability, response time and storage requirements. Constraints are I/O device capability, system representations, etc.
- Process requirements may also be specified mandating a particular IDE, programming language or development method.
- Non-functional requirements may be more critical than functional requirements. If these are not met, the system may be useless.

Types of nonfunctional requirement



Non-functional requirements implementation

- Non-functional requirements may affect the overall architecture of a system rather than the individual components.
  - For example, to ensure that performance requirements are met, you may have to organize the system to minimize communications between components.
- A single non-functional requirement, such as a security requirement, may generate a number of related functional requirements that define system services that are required.
  - It may also generate requirements that restrict existing requirements.

Non-functional classifications

- Product requirements
  - Requirements which specify that the delivered product must behave in a particular way e.g. execution speed, reliability, etc.
- Organisational requirements
  - Requirements which are a consequence of organisational policies and procedures e.g. process standards used, implementation requirements, etc.
- External requirements
  - Requirements which arise from factors which are external to the system and its development process e.g. interoperability requirements, legislative requirements, etc.

Goals and requirements

- Non-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 non-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.

Usability requirements

- The system should be easy to use by medical staff and should be organized in such a way that user errors are minimized. (Goal)
  - Medical staff shall be able to use all the system functions after four hours of training. After this training, the average number of errors made by experienced users shall not exceed two per hour of system use. (Testable non-functional requirement)

Performance requirement

Requirement 1 Le temps de réponse pour les actions de l'utilisateur doit être inférieur à 100ms dans 99% des cas.
Le système doit être réactif et ne pas ralentir le travail de l'utilisateur. Des tests de performance seront mis en place pour vérifier ce besoin ainsi qu'une solution de monitoring pour vérifier que la performance attendue est bien atteinte.

Non functional requirements

Requirement 3 L'application doit être résistante à des possibles piratages

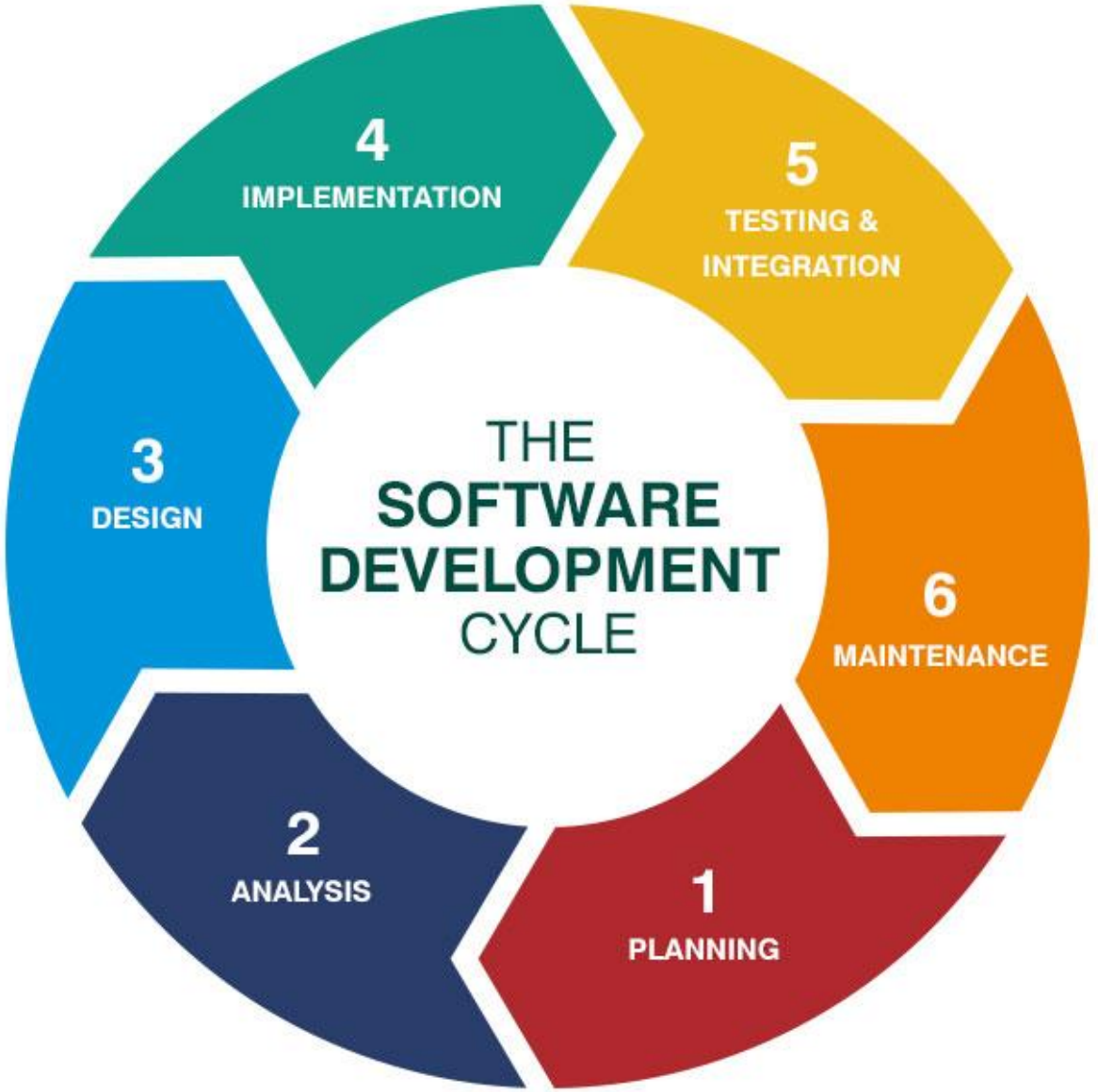
Les gens investissent de l'argent dans l'application, des piratages des serveurs ou compte individuels des utilisateurs pourraient être dangereux et nuire à la réputation du bar ainsi qu'à la confiance des utilisateurs.

Requirement 4 L'application doit protéger l'utilisateur d'une utilisation abusive de son compte en cas de vol

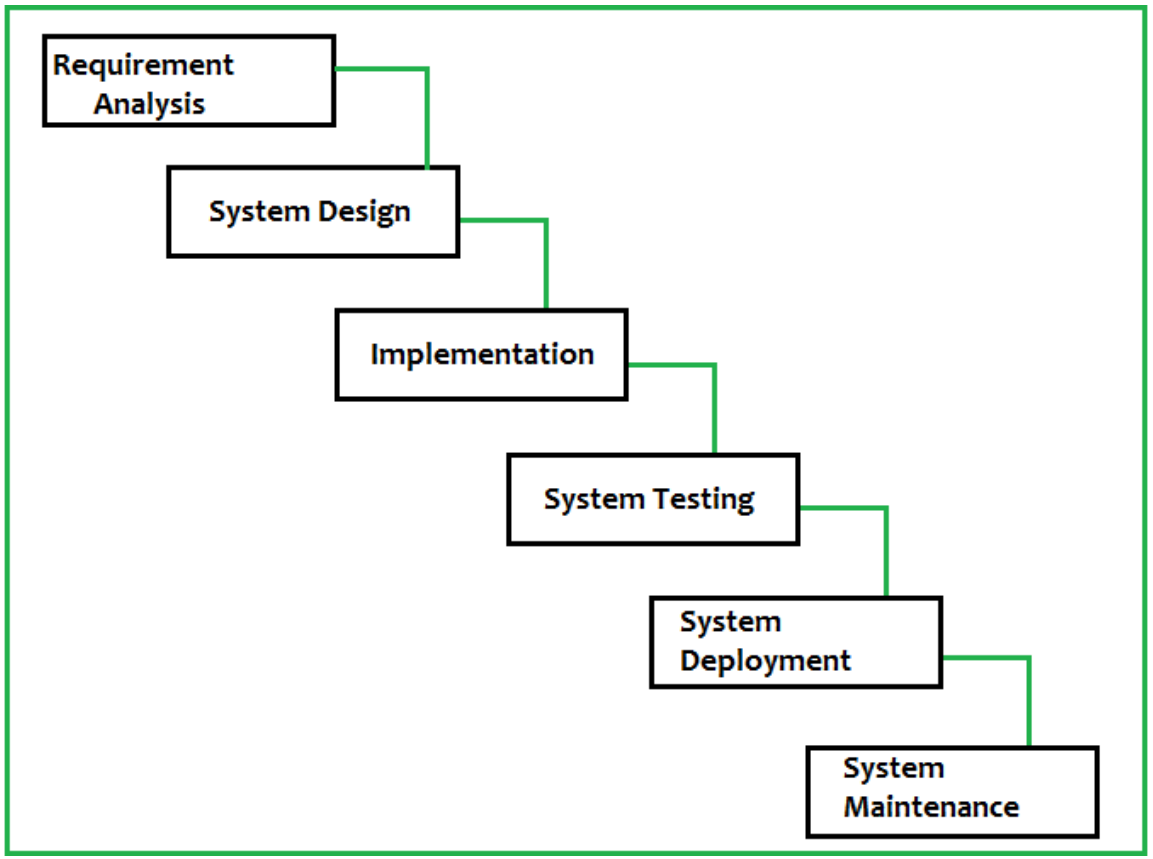
Le système doit pouvoir déconnecter l'utilisateur s'il a été inactif pendant un long moment, de même si un nombre de commande(s) absurde sont prise(s) avec son compte, le système doit pouvoir demander une authentification.

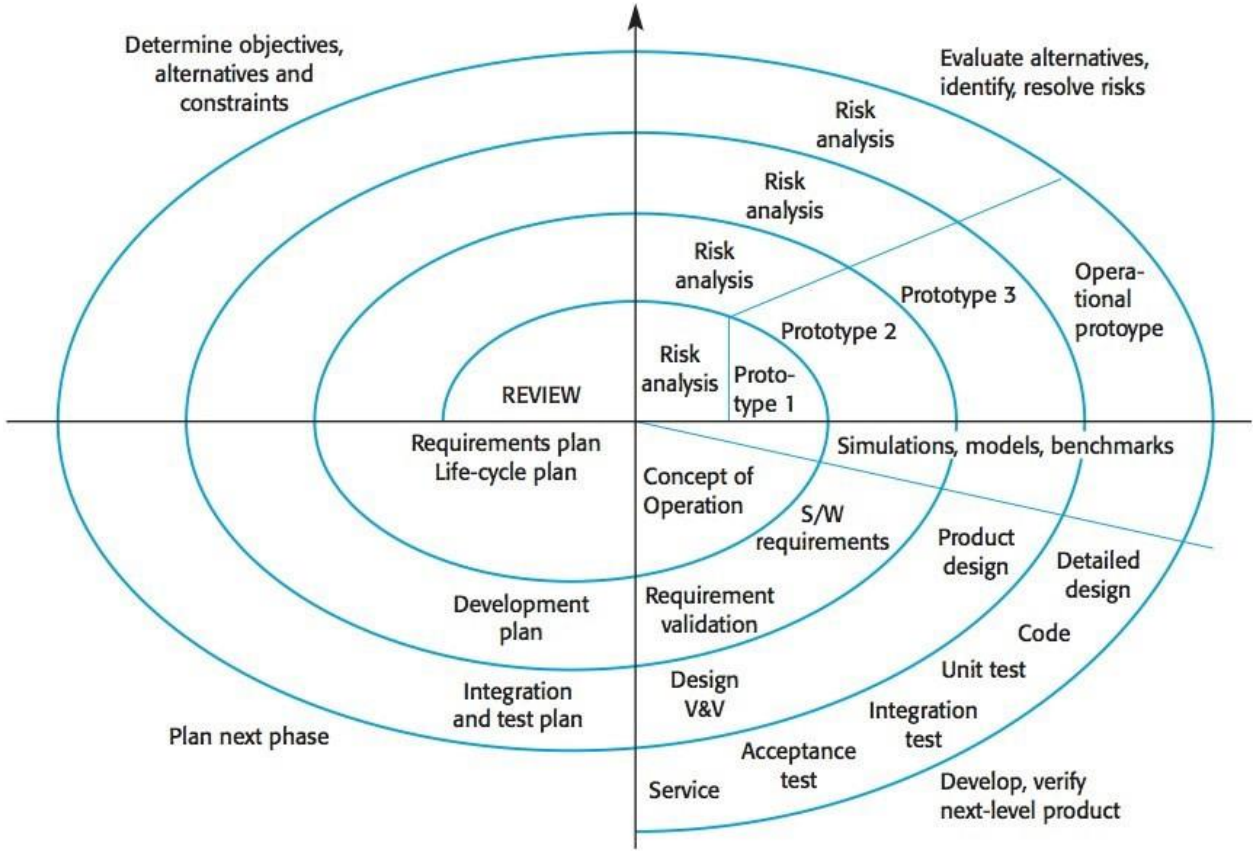
Requirements engineering processes

Reminders


4
IMPLEMENTATION
5
TESTING &
INTEGRATION
3
DESIGN
THE
SOFTWARE
DEVELOPMENT
CYCLE
6
MAINTENANCE
2
ANALYSIS
1
PLANNING

Software process models

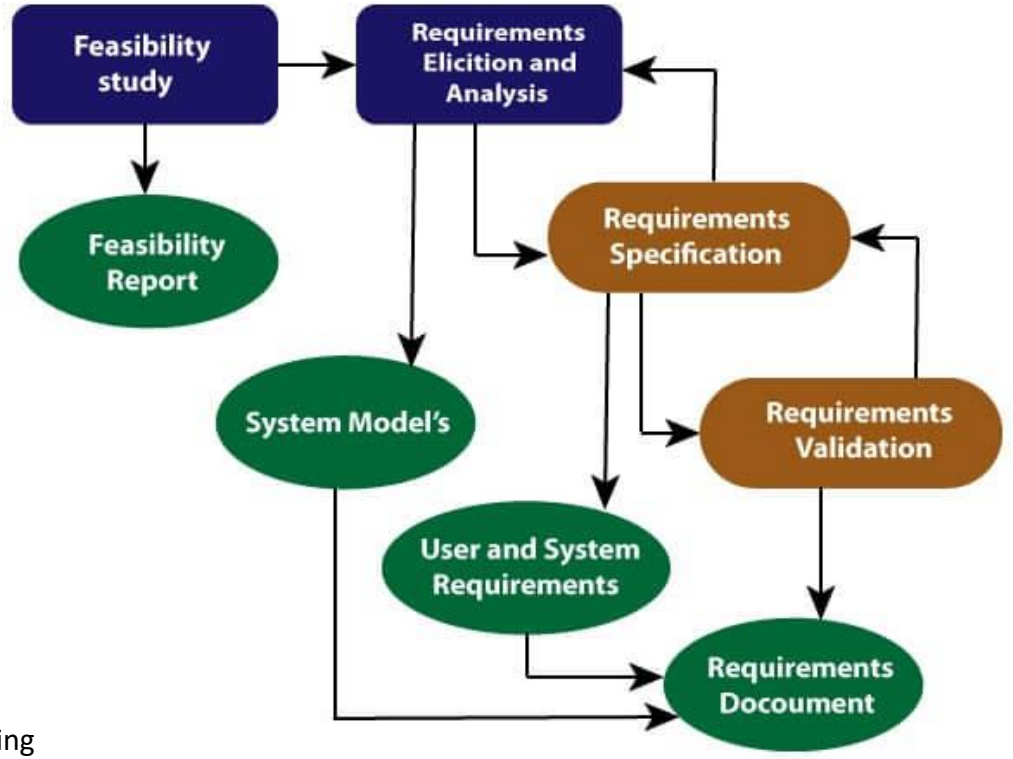




Requirement Engineering process

- Process of defining, documenting and Maintaining requirements

- Generic activities
  - Requirements elicitation;
  - Requirements analysis;
  - Requirement Specification
  - Requirements validation;
  - Requirements management



Requirement Engineering Process

https://www.javatpoint.com/software-engineering-requirement-engineering

Requirements engineering processes

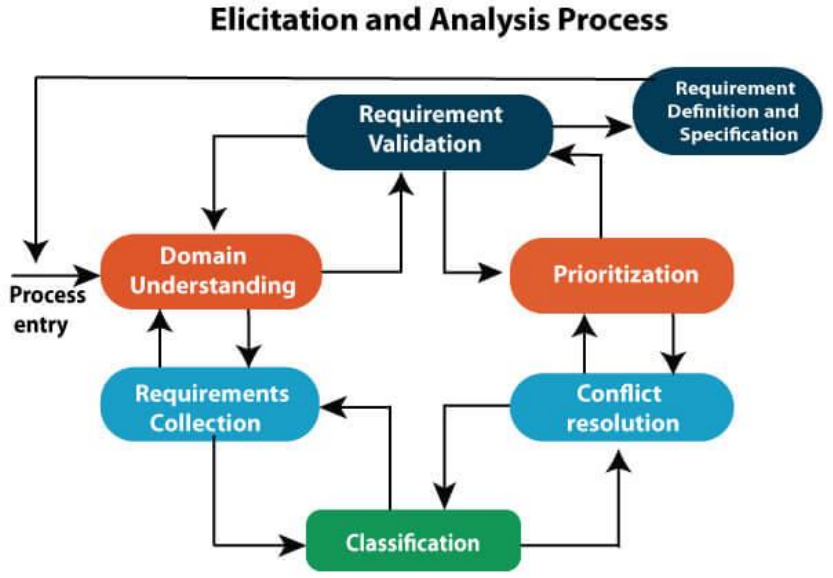
- The processes used for RE vary widely depending on the application domain, the people involved and the organisation developing the requirements.
- However, there are a number of generic activities common to all processes
  - Requirements elicitation;
  - Requirements analysis;
  - Requirements validation;
  - Requirements management.
- In practice, RE is an iterative activity in which these processes are interleaved.

Requirements elicitation and analysis

- Sometimes called requirements elicitation or requirements discovery.
- Involves technical staff working with customers to find out about the application domain, the services that the system should provide and the system's operational constraints.
- May involve end-users, managers, engineers involved in maintenance, domain experts, trade unions, etc. These are called stakeholders.

Requirements elicitation

- Software engineers work with a range of system stakeholders to find out about the application domain, the services that the system should provide, the required system performance, hardware constraints, other systems, etc.
- Stages include:
  - Requirements discovery,
  - Requirements classification and organization,
  - Requirements prioritization and negotiation,
  - Requirements specification.



Problems of requirements elicitation

- Stakeholders don’t know what they really want.
- Stakeholders express requirements in their own terms.
- Different stakeholders may have conflicting requirements.
- Organisational and political factors may influence the system requirements.
- The requirements change during the analysis process. New stakeholders may emerge and the business environment may change.

Process activities

- Requirements discovery
  - Interacting with stakeholders to discover their requirements. Domain requirements are also discovered at this stage.
- Requirements classification and organisation
  - Groups related requirements and organises them into coherent clusters.
- Prioritisation and negotiation
  - Prioritising requirements and resolving requirements conflicts.
- Requirements specification
  - Requirements are documented and input into the next round of the spiral.

Requirements discovery

- The process of gathering information about the required and existing systems and distilling the user and system requirements from this information.
- Interaction is with system stakeholders from managers to external regulators.
- Systems normally have a range of stakeholders.

Interviewing

- Formal or informal interviews with stakeholders are part of most RE processes.
- Types of interview
  - Closed interviews based on pre-determined list of questions
  - Open interviews where various issues are explored with stakeholders.
- Effective interviewing
  - Be open-minded, avoid pre-conceived ideas about the requirements and are willing to listen to stakeholders.
  - Prompt the interviewee to get discussions going using a springboard question, a requirements proposal, or by working together on a prototype system.

Lots of resources

https://www.axia-consulting.co.uk/requirements-gathering.htm

https://www.bridging-the-gap.com/what-questions-do-i-ask-during-requirements-elicitation/

Problems with interviews

- Application specialists may use language to describe their work that isn’t easy for the requirements engineer to understand.
- Interviews are not good for understanding domain requirements
  - Requirements engineers cannot understand specific domain terminology;
  - Some domain knowledge is so familiar that people find it hard to articulate or think that it isn’t worth articulating.

Ethnography

- A social scientist spends a considerable time observing and analysing how people actually work.
- People do not have to explain or articulate their work.
- Social and organisational factors of importance may be observed.
- Ethnographic studies have shown that work is usually richer and more complex than suggested by simple system models.

Scope of ethnography

- Requirements that are derived from the way that people actually work rather than the way I which process definitions suggest that they ought to work.
- Requirements that are derived from cooperation and awareness of other people's activities.
  - Awareness of what other people are doing leads to changes in the ways in which we do things.
- Ethnography is effective for understanding existing processes but cannot identify new features that should be added to a system.

Focused ethnography

- Developed in a project studying the air traffic control process
- Combines ethnography with prototyping
- Prototype development results in unanswered questions which focus the ethnographic analysis.
- The problem with ethnography is that it studies existing practices which may have some historical basis which is no longer relevant.

Stories and scenarios

- Scenarios and user stories are real-life examples of how a system can be used.
- Stories and scenarios are a description of how a system may be used for a particular task.
- Because they are based on a practical situation, stakeholders can relate to them and can comment on their situation with respect to the story.

Examples

- As a teacher, I need to be able to point students that are not attending my lecture so that I can keep track of absentees.
- As a study director, I need to be notified when a student is missing too many lectures

- Stories can lead to the development of features

Requirements specification

Requirements specification

- The process of writing down the user and system requirements in a requirements document.
- Requirements have to be understandable by end-users and customers who do not have a technical background and by developers that need to build the system.
- The requirements may be part of a contract for the system development
  - It is therefore important that these are as complete as possible.

Ways of writing a system requirements specification

Notation	Description
Natural language	The requirements are written using numbered sentences in natural language. Each sentence should express one requirement.
Structured natural language	The requirements are written in natural language on a standard form or template. Each field provides information about an aspect of the requirement.
Design description languages	This approach 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 now rarely used although it can be useful for interface specifications.
Graphical notations	Graphical models, supplemented by text annotations, are used to define the functional requirements for the system; UML use case and sequence diagrams are commonly used.
Mathematical specifications	These notations are based on mathematical concepts such as finite-state machines or sets. Although these unambiguous specifications can reduce the ambiguity in a requirements document, most customers don't understand a formal specification. They cannot check that it represents what they want and are reluctant to accept it as a system contract

Natural language specification

- Requirements are written as natural language sentences supplemented by diagrams and tables.
- Used for writing requirements because it is expressive, intuitive and universal. This means that the requirements can be understood by users and customers.

Guidelines for writing 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.
- Include an explanation (rationale) of why a requirement is necessary.

Problems with natural language

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

Structured specifications

- An approach to writing requirements where the freedom of the requirements writer is limited and requirements are written in a standard way.
- This works well for some types of requirements e.g. requirements for embedded control system but is sometimes too rigid for writing business system requirements.

Form-based specifications

- Definition of the function or entity.
- Description of inputs and where they come from.
- Description of outputs and where they go to.
- Information about the information needed for the computation and other entities used.
- Description of the action to be taken.
- Pre and post conditions (if appropriate).
- The side effects (if any) of the function.

Insulin Pump/Control Software/SRS/3.3.2	
Function	Compute insulin dose: Safe sugar level
Description	Computes the dose of insulin to be delivered when the current measured sugar level is in the safe zone between 3 and 7 units
Inputs	Current sugar reading (r2), the previous two readings (r0 and r1)
Source	Current sugar reading from sensor. Other readings from memory.
Outputs	CompDose – the dose in insulin to be delivered
Destination	Main control loop
Action	CompDose is zero if the sugar level is stable or falling or if the level is increasing but the rate of increase is decreasing. If the level is increasing and the rate of increase is increasing, then CompDose is computed by dividing the difference between the current sugar level and the previous level by 4 and rounding the result. If the result, is rounded to zero then CompDose is set to the minimum dose that can be delivered.
Requires	Two previous readings so that the rate of change of sugar level can be computed.
Pre-condition	The insulin reservoir contains at least the maximum allowed single dose of insulin
Post-condition	r0 is replaced by r1 then r1 is replaced by r2
Side-effects	None

Tabular specification

- Used to supplement natural language.
- Particularly useful when you have to define a number of possible alternative courses of action.
- For example, the insulin pump systems bases its computations on the rate of change of blood sugar level and the tabular specification explains how to calculate the insulin requirement for different scenarios.

Tabular specification of computation for an insulin pump

Condition	Action
Sugar level falling (r2 < r1)	CompDose = 0
Sugar level stable (r2 = r1)	CompDose = 0
Sugar level increasing and rate of increase decreasing ((r2 – r1) < (r1 – r0))	CompDose = 0
Sugar level increasing and rate of increase stable or increasing ((r2 – r1) ≥ (r1 – r0))	CompDose = round ((r2 – r1)/4) If rounded result = 0 then CompDose = MinimumDose

The software requirements document

- The software requirements document is 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.

Requirements document variability

- Information in requirements document depends on type of system and the approach to development used.
- Systems developed incrementally will, typically, have less detail in the requirements document.
- Requirements documents standards have been designed e.g. IEEE standard. These are mostly applicable to the requirements for large systems engineering projects.

Un modèle de document (celui du projet)

Cahier des charges

Le projet de MOCI

Version 0.1 draft

Préparé par <authors>

<Organization>

28 mai 2021

Introduction

1 Introduction

Ce cahier des charges est réalisé dans le cadre du cours de TELECOM Nancy et fait référence aux livres suivants [3, 1, 2].

1.1 Objectif

<Identifiez le produit dont le cahier des charges va être décrit dans ce document. Indiquez ce que couvre le produit, en particulier si le cahier des charges ne concerne qu'une partie d'un système.>

L'objectif du système est de permettre la gestion des absences à TELECOM Nancy. Il doit permettre la saisie des absences par les enseignants, la saisie de leur justification et le décompte de ces absences.

1.2 Audience et sections importantes

<Décrivez les lecteurs potentiels (développeurs, testeurs, marketing, etc) et indiquer quels sont les parties du document qui leur sont destinés.> Les lecteurs potentiels de ce documents sont la direction des études qui doit pouvoir contrôler que le système fonctionnerai bien selon ses attentes et les développeurs.

1.3 Portée du projet

<Donnez une description rapide du logiciel spécifié, son but, les objectifs métiers, les gains attendus et comment il contribue au objectifs et stratégies de l'organisation.> Le système est seulement en charge de la gestion des absences. Il doit en simplifier la saisie pour permettre à plus d'enseignants de le relever. Il doit surtout permettre à la direction des études et aux étudiants un suivi plus précis des absences pour permettre une réaction plus rapide en cas de problème.

1.4 Conventions

<Décrivez les conventions de nommage et de structure utilisés au long du document et comment ils peuvent aider le lecteur.>

Besoins fonctionnels

3 Besoins fonctionnels

<Pour chaque besoin fonctionnel, donnez une description détaillée de son fonctionnement permettant de comprendre plus précisément son fonctionnement. La numérotation doit correspondre aux besoins utilisateurs.>

Requirement 1

Description : Le système doit permettre de saisir l'absence d'un étudiant à un cours.
Priorité : **high**
Détails, contraintes et conséquences

1. L'absence est ajoutée aux absences de l'étudiant

Requirement 2

Description : Le système doit permettre à un enseignant et à la direction des études d'éditer la liste des absences pour un étudiant.
Priorité : **high**
Détails, contraintes et conséquences

1. L'utilisateur peut rechercher un étudiant à partir de son nom ou de son numéro INE.
2. La recherche est facilitée par un mécanisme d'auto-complétion.
3. La liste des absences s'affiche à l'écran : date, motif, justification.
4. Le total des absences justifiées et non justifiées est également affiché.

Requirement 3

Description : Le système doit notifier l'étudiant lorsqu'une nouvelle absence le concernant est ajoutée.
Priorité : **low**
Détails, contraintes et conséquences

1. Chaque fois qu'une absence est ajoutée par un enseignant ou par la scolarité, l'étudiant reçoit un message le prévenant
2. Le message contient la date et l'heure de l'absence ainsi que le nombre d'absences déjà enregistrée
3. Un étudiant peut choisir de ne plus recevoir de notifications.

Besoins des interfaces externes

4 Besoin des interfaces externes

4.1 Interfaces Utilisateurs

<Décrivez les interfaces utilisateur principales en incluant éventuellement des guides de conception, des contraintes et quelques exemples d'écrans si nécessaire>

4.2 Interfaces Matérielles

<Décrivez les produits et les caractéristiques des appareils utilisés pour l'application ainsi que les interfaces de communication>

4.3 Interfaces Logicielles

< Décrivez les logiciels ou applications, les composants ainsi que leur version, les bases de données et les outils qui seront utilisés avec l'application, quels sont les protocoles utilisés et les données échangées. >

4.4 Interfaces de communication

< Décrivez interfaces et les standards de communication utilisés ainsi que les standards de sécurité utilisés pour les échanges de données et le chiffrage si nécessaire >

Besoins non fonctionnels

5 Besoins non fonctionnels

5.1 Performance

Requirement 1
Description : Le temps de réponse pour les actions de l'utilisateur doit être inférieur à 100ms dans 99% des cas.
Le système doit être réactif et ne pas ralentir le travail de l'utilisateur. Des tests de performance seront mis en place pour vérifier ce besoin ainsi qu'une solution de monitoring pour vérifier que la performance attendue est bien atteinte.

5.2 Sureté

Requirement 2
Description : Le système doit pouvoir être mis à jour sans être interrompu
Les mises à jour logicielles doivent pouvoir être effectuées sans interruption du service

5.3 Sécurité

Requirement 3
Description : Le système doit fournir un moyen d'authentification à 2 facteurs

5.4 Qualité logicielle

5.5 Besoins liés au domaine

Appendices

6 Appendices

6.1 Appendice A : Glossaire

6.2 Appendice B : Modèles d'analyse

<Ajoutez ici les diagrammes et modèles qui peuvent servir à la compréhension du cahier des charges.>

Modèle de données

<Diagramme entité association ou diagramme de classe correspondant aux informations nécessaires à l'application.>

Requirements validation

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.

Requirements checking

- Validity. Does the system provide the functions which best support the customer's needs?
- Consistency. Are there any requirements conflicts?
- Completeness. Are all functions required by the customer included?
- Realism. Can the requirements be implemented given available budget and technology
- Verifiability. Can the requirements be checked?

Requirements validation techniques

- Requirements reviews
  - Systematic manual analysis of the requirements.
- Prototyping
  - Using an executable model of the system to check requirements. Covered in Chapter 2.
- Test-case generation
  - Developing tests for requirements to check testability.

Requirements reviews

- Regular reviews should be 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.

Review checks

- Verifiability
  - Is the requirement realistically testable?
- 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?

Requirements management

- Requirements management is the process of managing changing requirements during the requirements engineering process and system development.
- New requirements emerge as a system is being developed and after it has gone into use.
- You need to keep track of individual requirements and maintain links between dependent requirements so that you can assess the impact of requirements changes. You need to establish a formal process for making change proposals and linking these to system requirements.

Key points

- Requirements for a software system set out what the system should do and define constraints on its operation and implementation.
- Functional requirements are statements of the services that the system must provide or are descriptions of how some computations must be carried out.
- Non-functional requirements often constrain the system being developed and the development process being used.
- They often relate to the emergent properties of the system and therefore apply to the system as a whole.

Key points

- The requirements engineering process is an iterative process that includes requirements elicitation, specification and validation.
- Requirements elicitation is an iterative process that can be represented as a spiral of activities – requirements discovery, requirements classification and organization, requirements negotiation and requirements documentation.
- You can use a range of techniques for requirements elicitation including interviews and ethnography. User stories and scenarios may be used to facilitate discussions.

Key points

- Requirements specification is the process of formally documenting the user and system requirements and creating a software requirements document.
- The software requirements document is an agreed statement of the system requirements. It should be organized so that both system customers and software developers can use it.

Key points

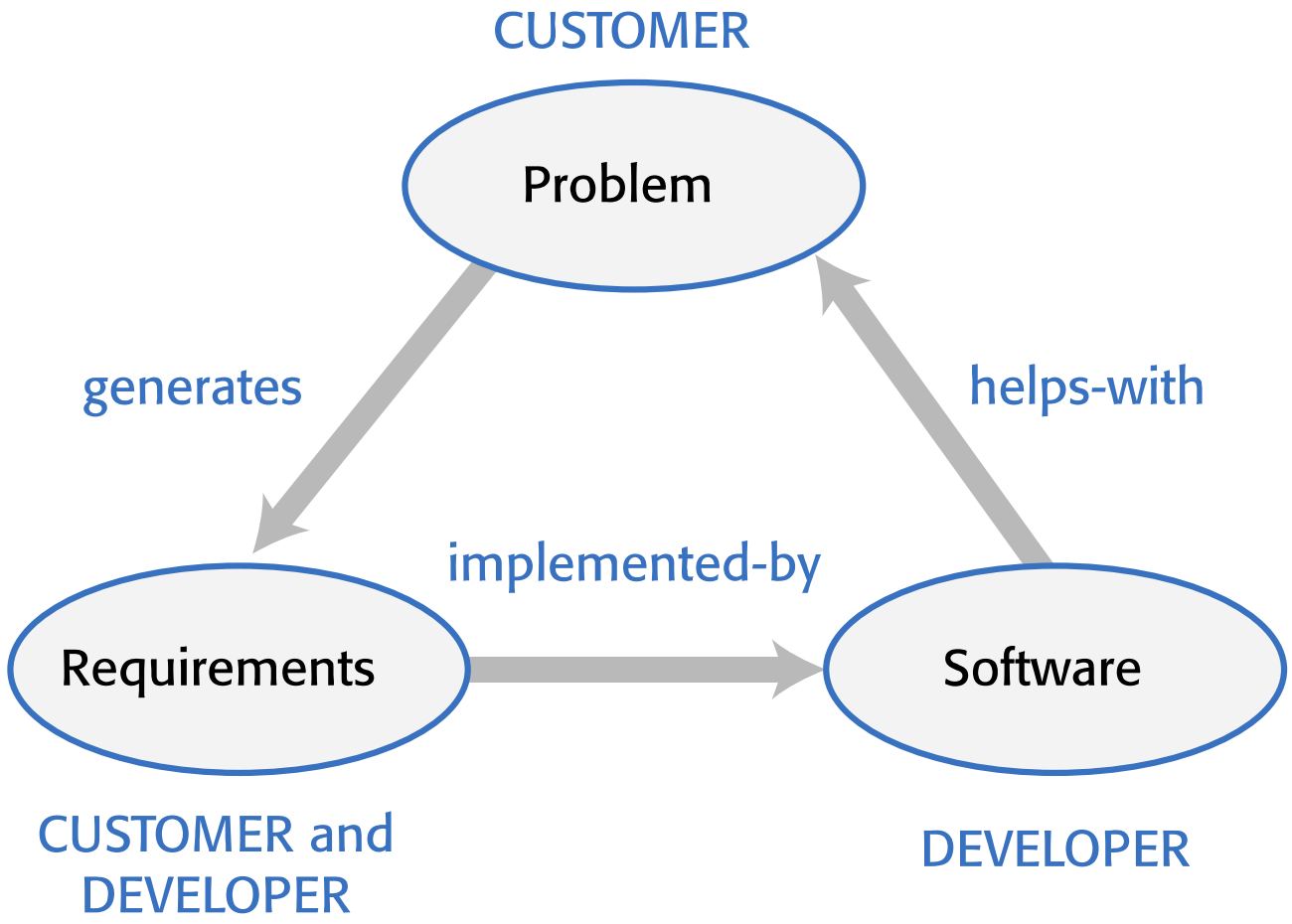
- Requirements validation is the process of checking the requirements for validity, consistency, completeness, realism and verifiability.
- Business, organizational and technical changes inevitably lead to changes to the requirements for a software system. Requirements management is the process of managing and controlling these changes.

Requirements and Software Products

Software products

- Software products are generic software systems that provide functionality that is useful to a range of customers.
- Many different types of products are available from large-scale business systems (e.g. MS Excel) through personal products (e.g. Evernote) to simple mobile phone apps and games (e.g. Suduko).
- Software product engineering methods and techniques have evolved from software engineering techniques that support the development of one-off, custom software systems.
- Custom software systems are still important for large businesses, government and public bodies. They are developed in dedicated software projects.

Project-based software engineering



Product software engineering

- The starting point for product development is a business opportunity that is identified by individuals or a company. They develop a software product to take advantage of this opportunity and sell this to customers.
- The company who identified the opportunity design and implement a set of software features that realize the opportunity and that will be useful to customers.
- The software development company are responsible for deciding on the development timescale, what features to include and when the product should change.
- Rapid delivery of software products is essential to capture the market for that type of product.

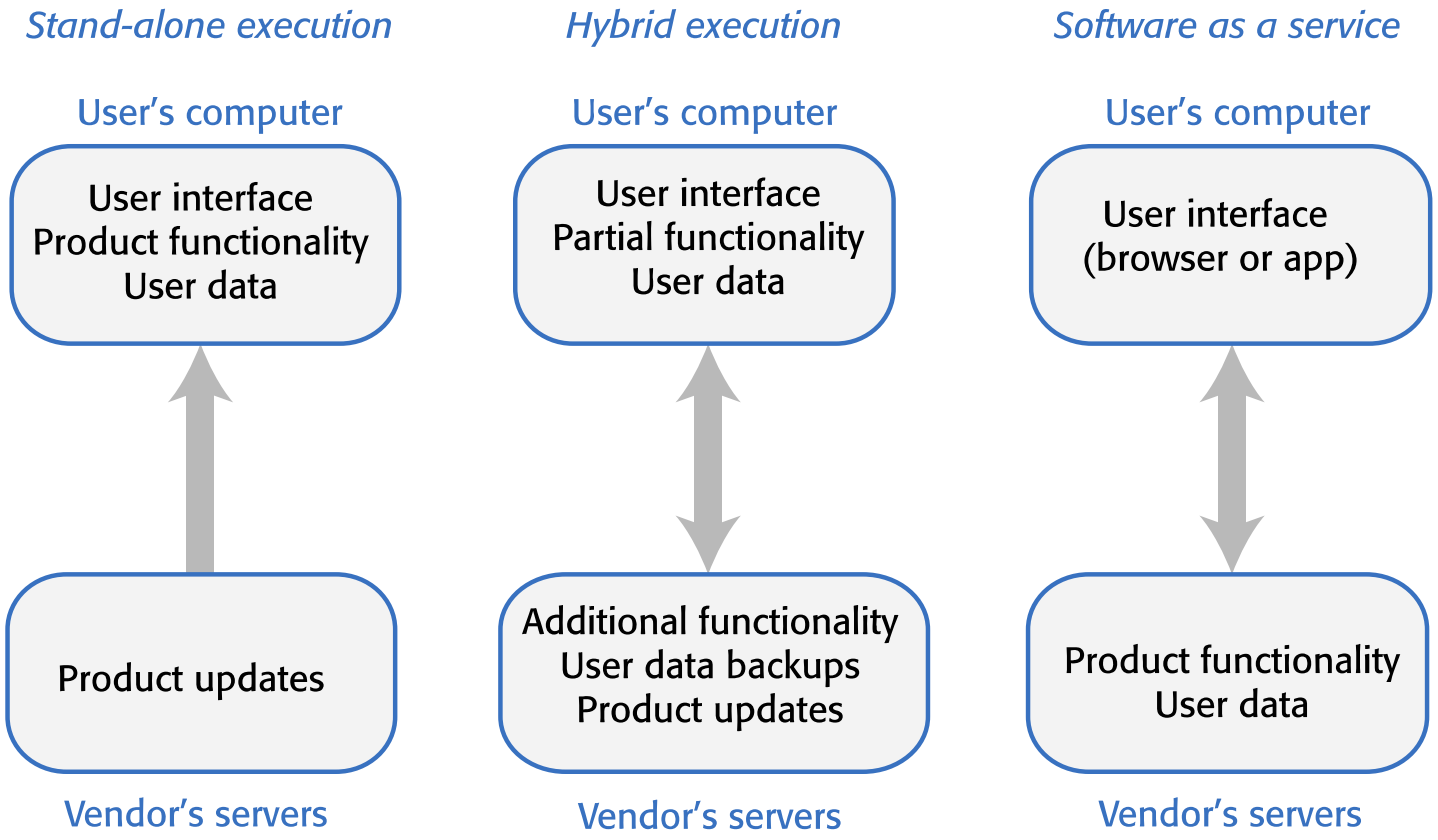
Software product lines and platforms

- Software product line
  A set of software products that share a common core. Each member of the product line includes customer-specific adaptations and additions. Software product lines may be used to implement a custom system for a customer with specific needs that can't be met by a generic product.
- Platform
  A software (or software+hardware) product that includes functionality so that new applications can be built on it. An example of a platform that you probably use is Facebook. It provides an extensive set of product functionality but also provides support for creating 'Facebook apps'. These add new features that may be used by a business or a Facebook interest group.

Software execution models

- Stand-alone The software executes entirely on the customer's computers.
- Hybrid Part of the software's functionality is implemented on the customer's computer but some features are implemented on the product developer's servers.
- Software service All of the product's features are implemented on the developer's servers and the customer accesses these through a browser or a mobile app.

Software execution models



Comparable software development

- The key feature of product development is that there is no external customer that generates requirements and pays for the software. This is also true for other types of software development:
  - Student projects Individuals or student groups develop software as part of their course. Given an assignment, they decide what features to include in the software.
  - Research software Researchers develop software to help them answer questions that are relevant to their research.
  - Internal tool development Software developers may develop tools to support their work - in essence, these are internal products that are not intended for customer release.

Features, scenarios and stories

Software products

- There are three factors that drive the design of software products
  - 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
- In the early stage of product development, you are trying to understand, what product features would be useful to users, and what they like and dislike about the products that they use.

Software features

- A feature is a fragment of functionality such as a ‘print’ feature, a ‘change background feature’, a ‘new document’ feature and so on.
- Before you start programming a product, you should aim to create a list of features to be included in your product.
- The feature list should be your starting point for product design and development.

User understanding

- It makes sense in any product development to spend time trying to understand the potential users and customers of your product.
- A range of techniques have been developed for understanding the ways that people work and use software.
  - These include user interviews, surveys, ethnography and task analysis.
  - Some of these techniques are expensive and unrealistic for small companies.
- Informal user analysis and discussions, which simply involve asking users about their work, the software that they use, and its strengths and weaknesses are inexpensive and very valuable.

Figure 3.1 From personas to features

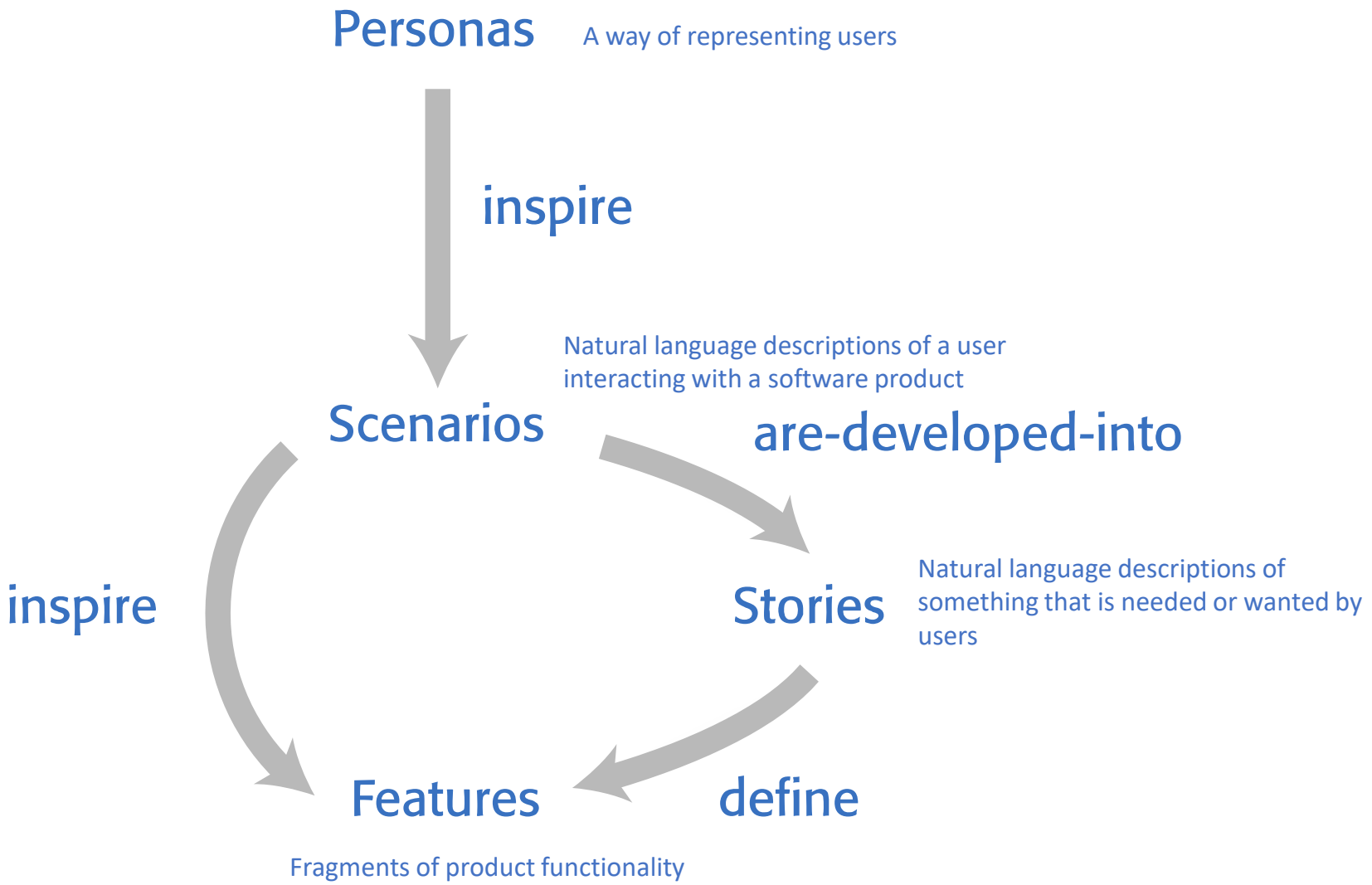


Figure 3.2 Feature description

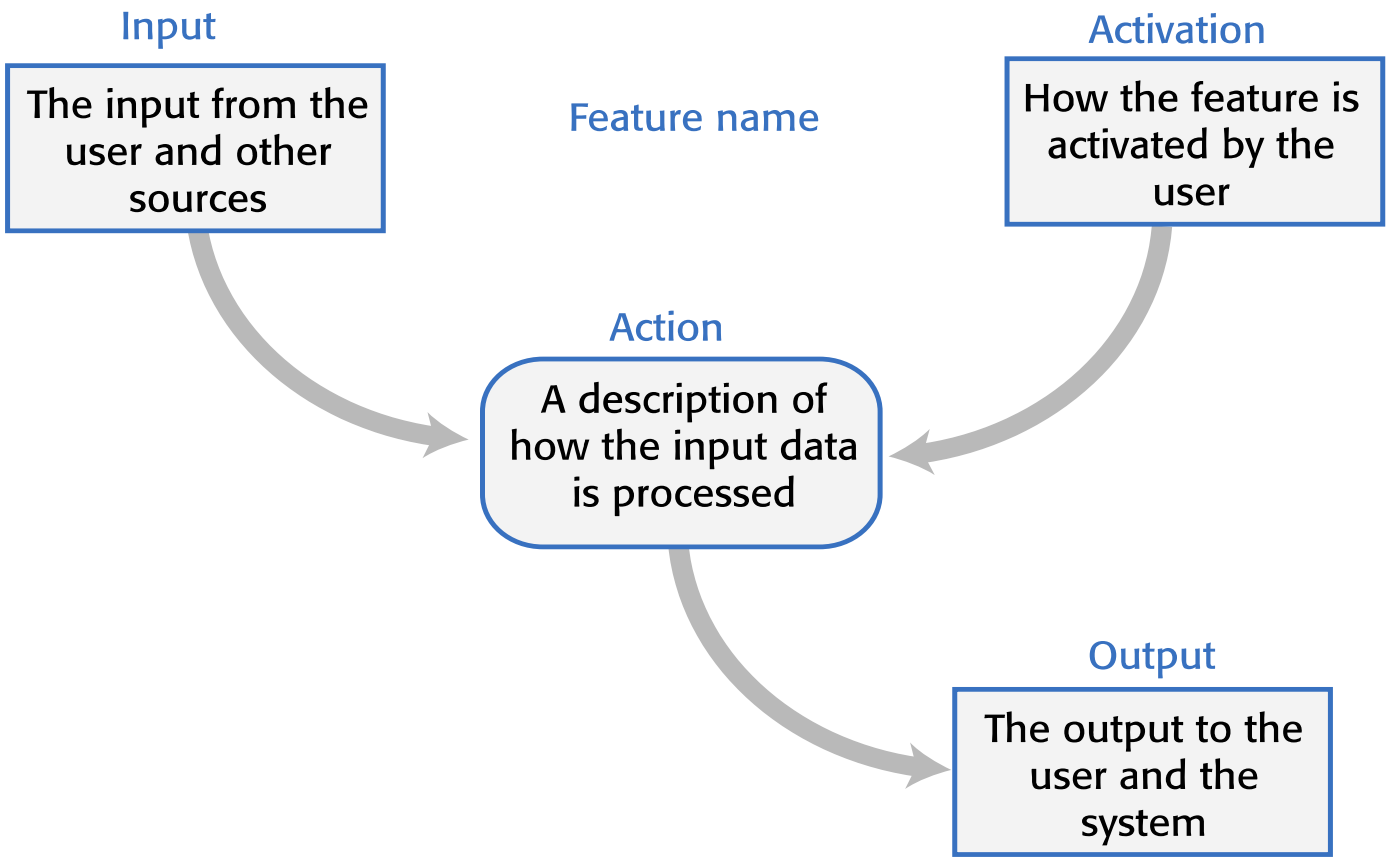
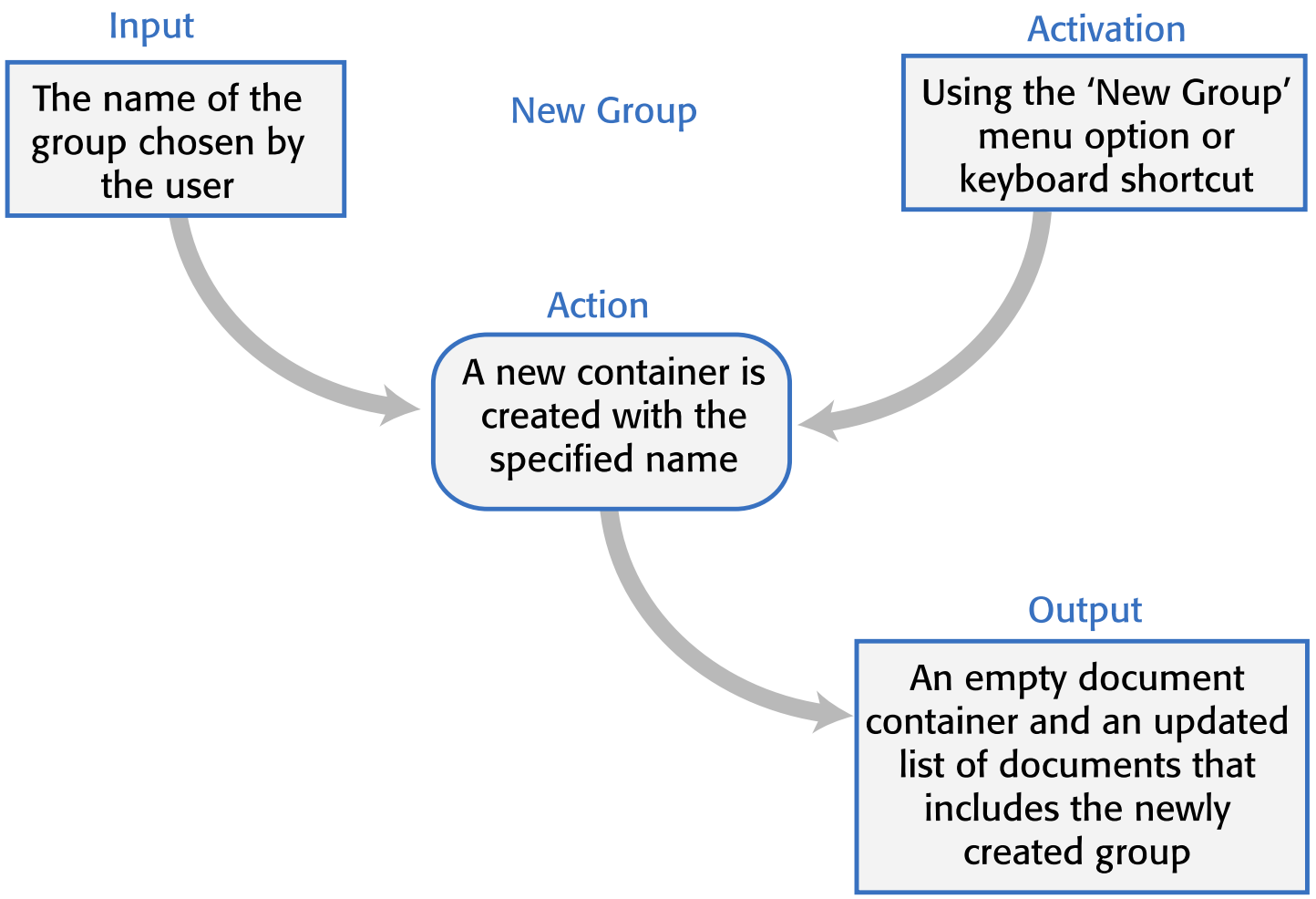


Figure 3.3 The ‘New Group’ feature description



Personas

- You need to have an understanding of your potential users to design features that they are likely to find useful and to design a user interface that is suited to them.
- Personas are ‘imagined users’ where you create a character portrait of a type of user that you think might use your product.
  - For example, if your product is aimed at managing appointments for dentists, you might create a dentist persona, a receptionist persona and a patient persona.
- Personas of different types of user help you imagine what these users may want to do with your software and how it might be used. They help you envisage difficulties that they might have in understanding and using product features.

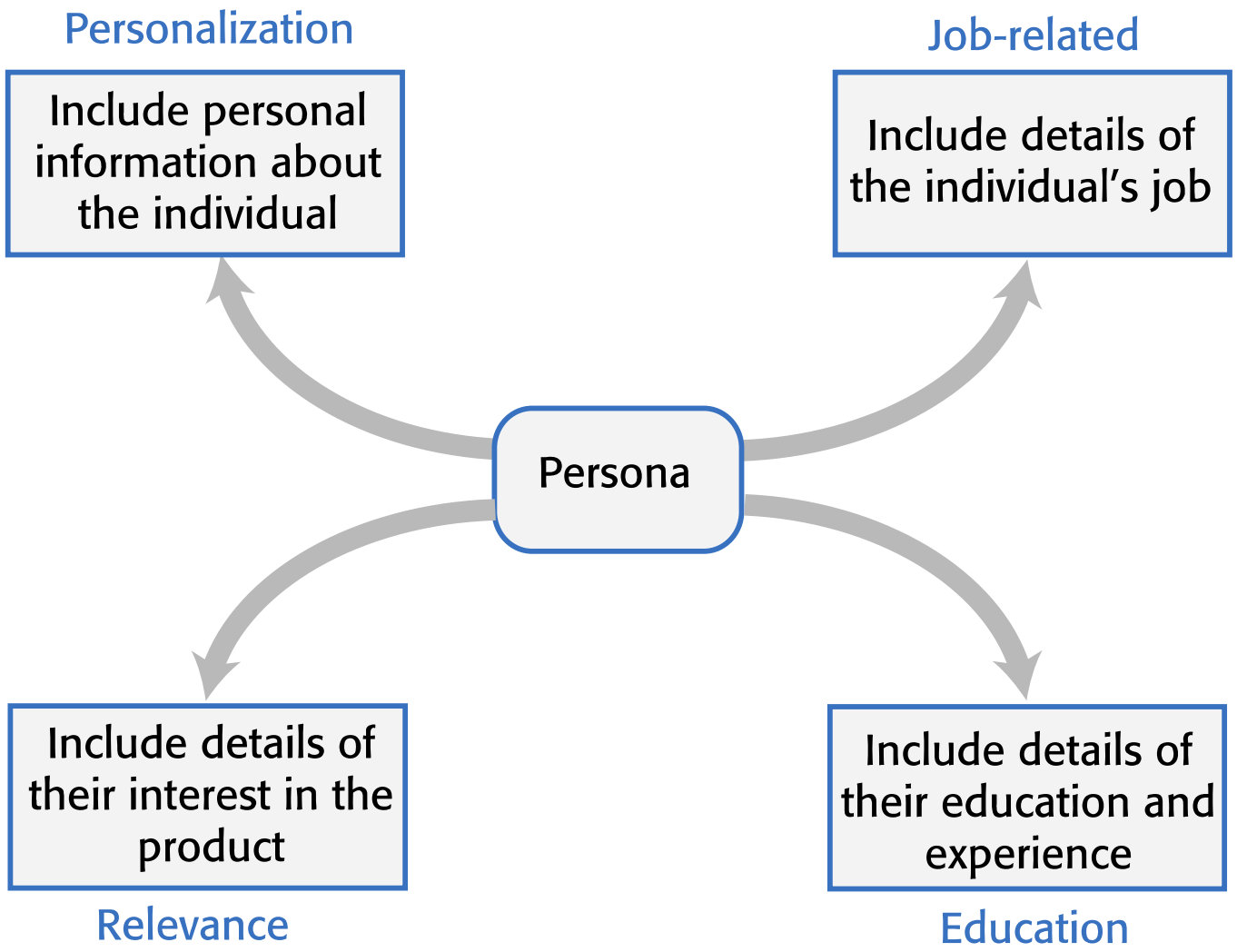
Jack, a primary school teacher

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.

Persona descriptions

- A persona should ‘paint a picture’ of a type of product user. They should be relatively short and easy-to-read.
- You should describe their background and why they might want to use your product.
- You should also say something about their educational background and technical skills.
- These help you assess whether or not a software feature is likely to be useful, understandable and usable by typical product users.



Personalization
You should give them a name and say something about their personal circumstances. This is important because you shouldn't think of a persona as a role but as an individual. It is sometimes helpful to use an appropriate stock photograph to represent the person in the persona. Some studies suggest that this helps project teams use personas more effectively.

Job-related
If your product is targeted at business, you should say something about their job and (if necessary) what that job involves. For some jobs, such as a teacher where readers are likely to be familiar with the job, this may not be necessary.

Education
You should describe their educational background and their level of technical skills and experience. This is important, especially for interface design.

Relevance
If you can, you should say why they might be interested in using the product and what they might want to do with it.

Emma, a history teacher

Emma, age 41, is a history teacher in a secondary school (high school) in Edinburgh. She teaches students from ages 12 to 18. She was born in Cardiff in Wales where both her father and her mother were teachers. After completing a degree in history from Newcastle University, she moved to Edinburgh to be with her partner and trained as a teacher. She has two children, aged 6 and 8, who both attend the local primary school. She likes to get home as early as she can to see her children, so often does lesson preparation, administration and marking from home.

Emma uses social media and the usual productivity applications to prepare her lessons, but is not particularly interested in digital technologies. She hates the virtual learning environment that is currently used in her school and avoids using it if she can. She believes that face-to-face teaching is most effective. She might use the iLearn system for administration and access to historic films and documents. However, she is not interested in a blended digital/face-to-face approach to teaching.

Elena, a school IT technician

Elena, age 28, is a senior IT technician in a large secondary school (high school) in Glasgow with over 2000 students. Originally from Poland, she has a diploma in electronics from Potsdam University. She moved to Scotland in 2011 after being unemployed for a year after graduation. She has a Scottish partner, no children, and hopes to develop her career in Scotland. She was originally appointed as a junior technician but was promoted, in 2014, to a senior post responsible for all the school computers.

Although not involved directly in teaching, Elena is often called on to help in computer science classes. She is a competent Python programmer and is a ‘power user’ of digital technologies. She has a long-term career goal of becoming a technical expert in digital learning technologies and being involved in their development. She wants to become an expert in the iLearn system and sees it as an experimental platform for supporting new uses for digital learning.

Persona benefits

- The main benefit of personas is that they help you and other development team members empathize with potential users of the software.
- Personas help because they are a tool that allows developers to ‘step into the user’s shoes’.
  - Instead of thinking about what you would do in a particular situation, you can imagine how a persona would behave and react.
- Personas can help you check your ideas to make sure that you are not including product features that aren’t really needed.
- They help you to avoid making unwarranted assumptions, based on your own knowledge, and designing an over-complicated or irrelevant product.

Deriving personas

- Personas should be based on an understanding of the potential product users, their jobs, their background and their aspirations.
- You should study and survey potential users to understand what they want and how they might use the product.
- From this data, you can then abstract the essential information about the different types of product user and use this as a basis for creating personas.
- Personas that are developed on the basis of limited user information are called proto-personas.
  - Proto-personas may be created as a collective team exercise using whatever information is available about potential product users. They can never be as accurate as personas developed from detailed user studies, but they are better than nothing.

Scenarios

- A scenario is a narrative that describes how a user, or a group of users, might use your system.
- There is no need to include everything in a scenario – the scenario isn’t a system specification.
- It is simply a description of a situation where a user is using your product’s features to do something that they want to do.
- Scenario descriptions may vary in length from two to three paragraphs up to a page of text.

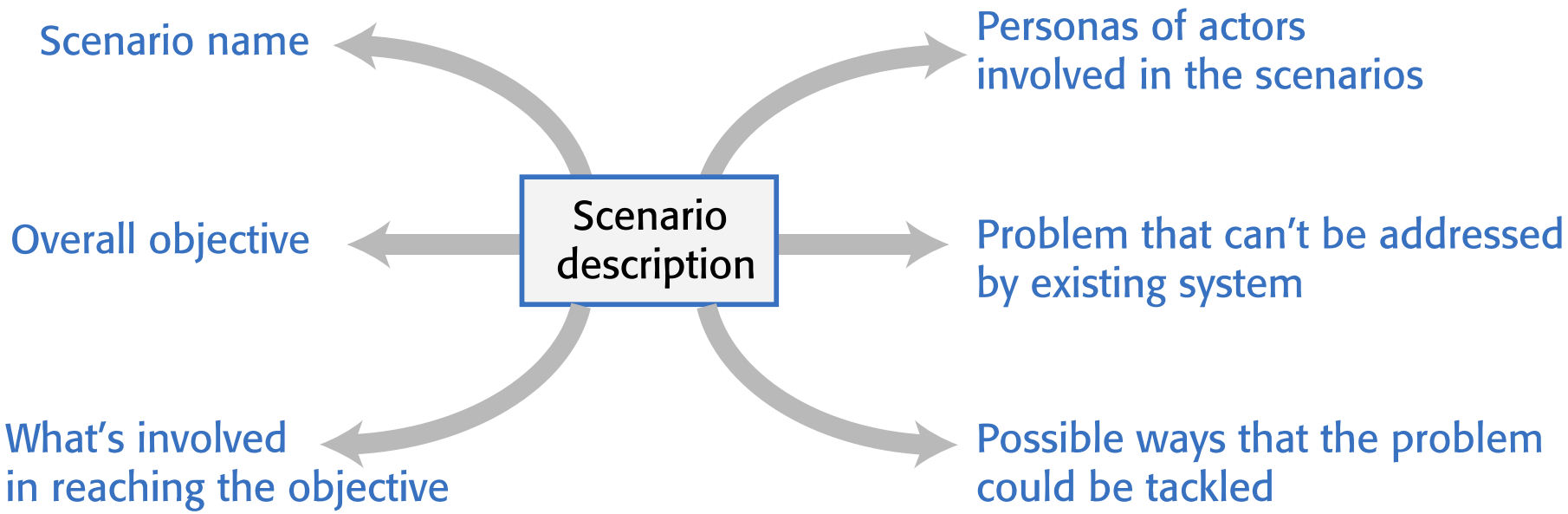
Fishing in Ullapool

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.



Scenario elements

- A brief statement of the overall objective.
  - In Jack's scenario, this is to support a class project on the fishing industry.
- References to the personas involved (Jack) so that you can get information about the capabilities and motivation of that user.
- Information about what is involved in doing the activity. For example, in Jack's scenario this involves gathering reminiscences from relatives, accessing newspaper archives, etc.
- An explanation of problems that can't be readily addressed using the existing system.
  - Young children don't understand issues such as copyright and privacy, so photo sharing requires a site that a teacher can moderate to make sure that published images are legal and acceptable.
- A description of one way that the identified problem might be addressed.
  - In Jack's scenario, the preferred approach is to use an external tool designed for school students.

Emma’s scenario

- Emma’s scenario is different from Jack’s scenario in that it describes a common and well-understood process rather than something new.
- Emma is an e-learning sceptic and she is not interested in innovative applications. She wants a system that will make her life easier and reduce the amount of routine administration that she has to do.
- The scenario discusses how parts of the process (setting up an email group and web page) are automated by the iLearn system.

Emma is teaching the history of the First World War to a class of 14 year olds (S3). A group of S3 students are visiting the historic World War One battlefields in northern France. She want to set up a ‘battlefields group’ where the students who are attending the trip can share their research about the places they are visiting as well as their pictures and thoughts about the visit.

From home, she logs onto the iLearn system system using her Google account credentials. Emma has two iLearn accounts – her teacher account and a parent account associated with the local primary school. The system recognises that she is a multiple account owner and asks her to select the account to be used. She chooses the teacher account and the system generates her personal welcome screen. As well as her selected applications, this also shows management apps that help teachers create and manage student groups.

Emma selects the ‘group management’ app, which recognizes her role and school from her identity information and creates a new group. The system prompts for the class year (S3) and subject (history) and automatically populates the new group with all S3 students who are studying history. She selects those students going on the trip and adds her teacher colleagues, Jamie and Claire, to the group.

She names the group and confirms that it should be created. The app sets up an icon on her iLearn screen to represent the group, creates an email alias for the group and asks Emma if she wishes to share the group. She shares access with everyone in the group, which means that they also see the icon on their screen. To avoid getting too many emails from students, restricts sharing of the email alias to Jamie and Claire.

The group management app then asksEmma if she wishes to set up a group web page, wiki and blog. Emma confirms that a web page should be created and she types some text to be included on that page.

She then accesses flickr using the icon on her screen, logs in and creates a private group to share trip photos that students and teachers have taken. She uploads some of her own photos from previous trips and emails an invitation to join the photo-sharing group to the Battlefield email list. Emma uploads material from her own laptop that she has written about the trip to iLearn and shares this with the ‘Battlefields Group’. This action adds her documents to the web page and generates an alert to group members that new material is available.

Writing scenarios

- Scenarios should always be written from the user's perspective and based on identified personas or real users.
- Your starting point for scenario writing should be the personas that you have created. You should normally try to imagine several scenarios from each persona.
- Ideally, scenarios should be general and should not include implementation information.
  - However, describing an implementation is often the easiest way to explain how a task is done.
- It is important to ensure that you have coverage of all of the potential user roles when describing a system.

Elena has been asked by David, the head of the art department in her school, to help set up an iLearn environment for his department. David wants an environment that includes tools for making and sharing art, access to external websites to study artworks, and ‘exhibition’ facilities so that the students’ work can be displayed.

Elena’s starts by talking to art teachers to discover the tools that they recommend and the art sites that they use for studies. She also discovers that the tools they use and the sites they access vary according to the age of their students. Consequently, different student groups should be presented with a toolset that is appropriate for their age and experience.

Once she has established what is required, Elena logs into the iLearn system as an administrator and starts configuring the art environment using the iLearn setup service. She creates sub-environments for three age groups plus a shared environment that includes tools and sites that may be used by all students.

She drags and drops tools that are available locally and the URLs of external websites into each of these environments. For each of the sub-environments, she assigns an art teacher as its administrator so that they can refine the tool and web site selection that has been set up. She publishes the environments in ‘review mode’ and makes them available to the teachers in the art department.

After discussing the environments with the teachers, Elena shows them how to refine and extend the environments. Once they have agreed that the art environment is useful, it is released to all students in the school.

User involvement

- It is easy for anyone to read and understand scenarios, so it is possible to get users involved in their development.
- The best approach is to develop an imaginary scenario based on our understanding of how the system might be used then ask users to explain what you have got wrong.
- They might ask about things they did not understand and suggest how the scenario could be extended and made more realistic.
- Our experience was that users are not good at writing scenarios.
  - The scenarios that they created were based on how they worked at the moment. They were far too detailed and the users couldn’t easily generalize their experience.

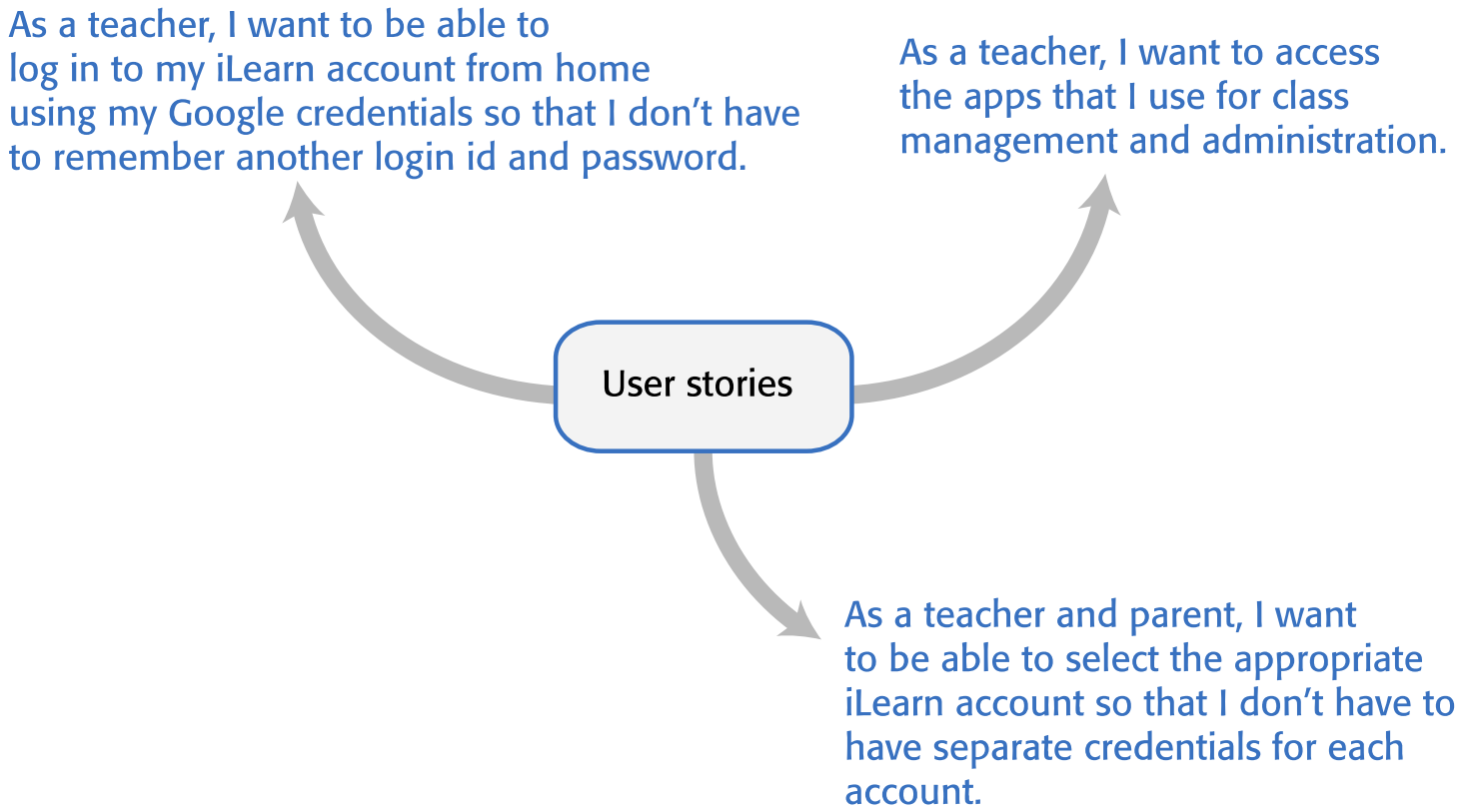
User stories

- Scenarios are high-level stories of system use. They should describe a sequence of interactions with the system but should not include details of these interactions.
- User stories are finer-grain narratives that set out in a more detailed and structured way a single thing that a user wants from a software system.
  - As an author, I need a way to organize the book that I'm writing into chapters and sections.
- This story reflects what has become 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
- A variant of this 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

- An important use of user stories is in planning.
  - Many users of the Scrum method represent the product backlog as a set of user stories.
- User stories should focus on a clearly defined system feature or aspect of a feature that can be implemented within a single sprint.
- If the story is about a more complex feature that might take several sprints to implement, then it is called an epic.
  - As a system manager, I need a way to backup the system and restore either individual applications, files, directories or the whole system.
  - There is a lot of functionality associated with this user story. For implementation, it should be broken down into simpler stories with each story focusing on a single aspect of the backup system.

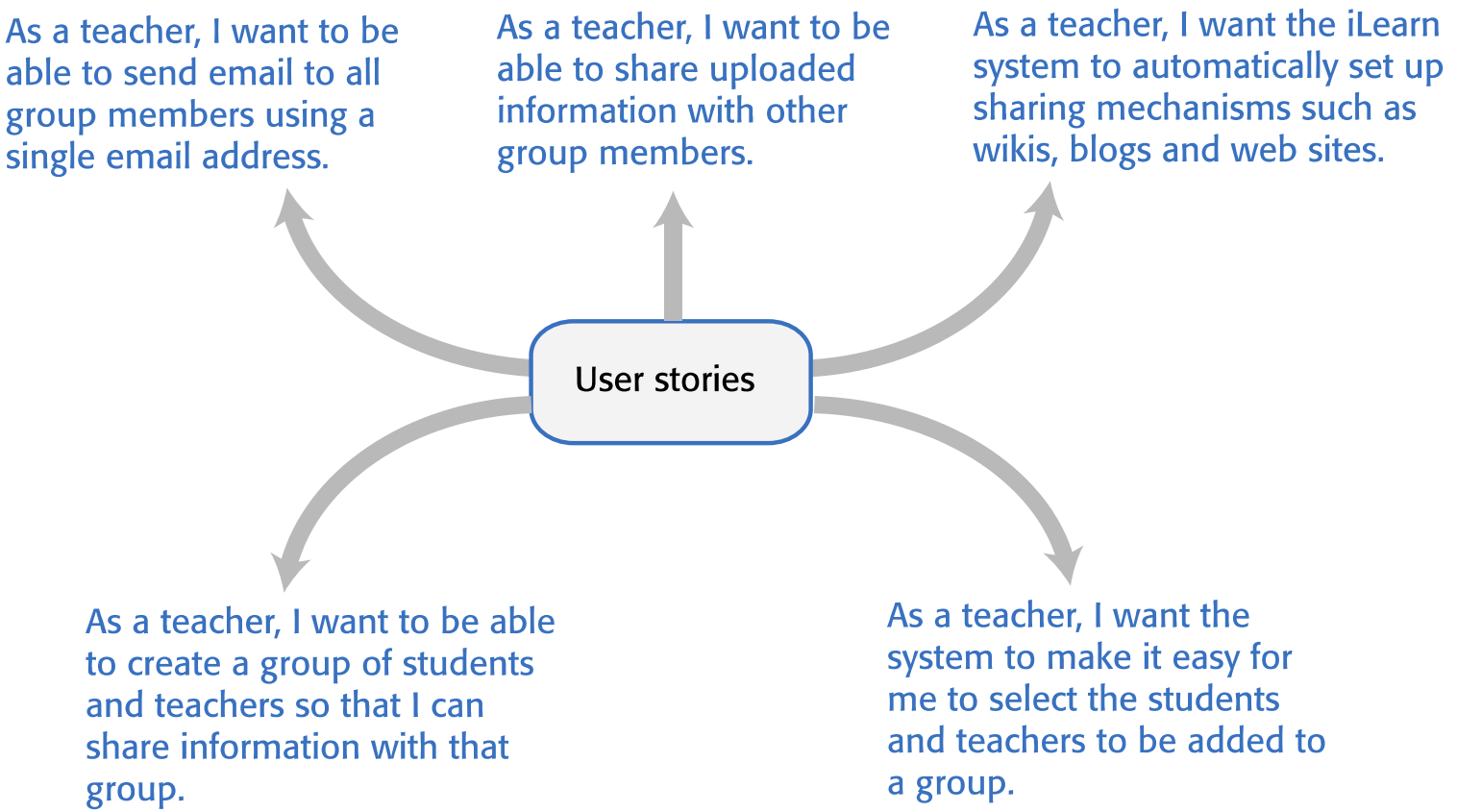
User stories from Emma's scenario



Feature description using user stories

- Stories can be used to describe features in your product that should be implemented.
- Each feature can have a set of associated stories that describe how that feature is used.

User stories describing the Groups feature

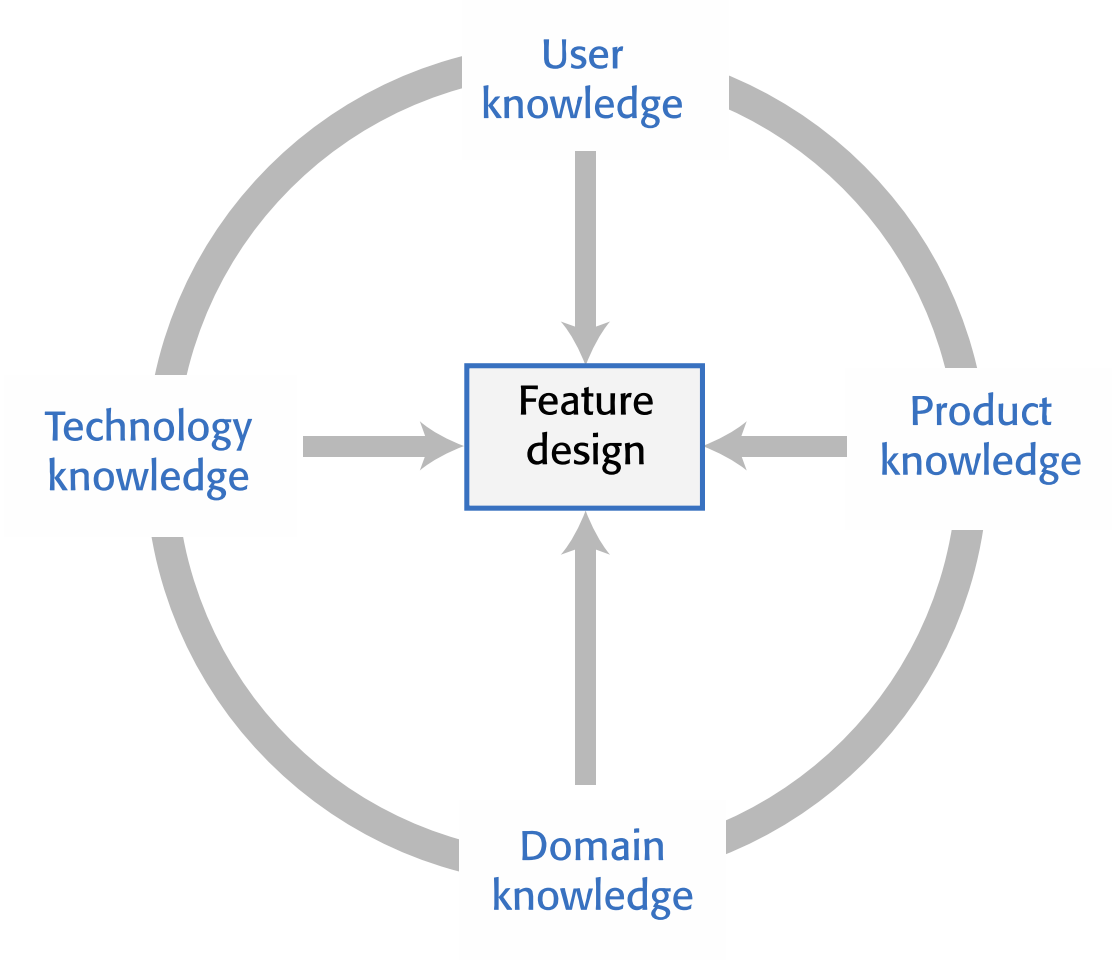


Stories and scenarios

- As you can express all of the functionality described in a scenario as user stories, do you really need scenarios?’
- Scenarios are more natural and are helpful for the following reasons:
  - Scenarios read more naturally because they describe what a user of a system is actually doing with that system. People often find it easier to relate to this specific information rather than the statement of wants or needs set out in a set of user stories.
  - If you are interviewing real users or are checking a scenario with real users, they don’t talk in the stylized way that is used in user stories. People relate better to the more natural narrative in scenarios.
  - Scenarios often provide more context - information about what the user is trying to do and their normal ways of working. You can do this in user stories, but it means that they are no longer simple statements about the use of a system feature.

Feature identification

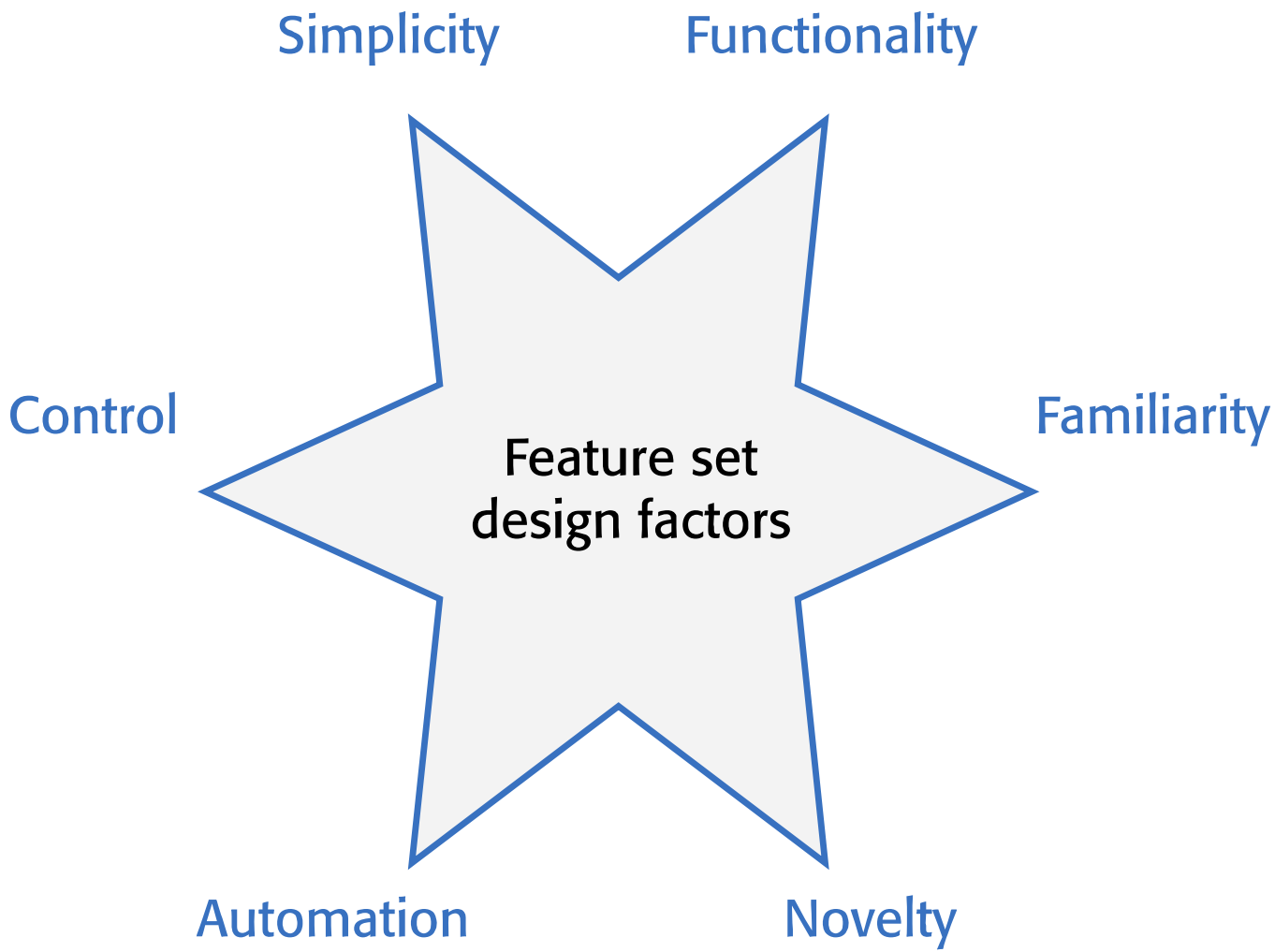
- Your aim in the initial stage of product design should be to create a list of features that define your product.
- A feature is a way of allowing users to access and use your product's functionality so the feature list defines the overall functionality of the system.
- Features should be independent, coherent and relevant:
  - Independence
    Features should not depend on how other system features are implemented and should not be affected by the order of activation of other features.
  - Coherence
    Features should be linked to a single item of functionality. They should not do more than one thing and they should never have side-effects.
  - Relevance
    Features should reflect the way that users normally carry out some task. They should not provide obscure functionality that is hardly ever required.



Knowledge required for feature design

- User knowledge
  You can use user scenarios and user stories to inform the team of what users want and how they might use it the software features.
- Product knowledge
  You may have experience of existing products or decide to research what these products do as part of your development process. Sometimes, your features have to replicate existing features in these products because they provide fundamental functionality that is always required.
- Domain knowledge
  This is knowledge of the domain or work area(e.g. finance, event booking) that your product aims to support. By understanding the domain, you can think of new innovative ways of helping users do what they want to do.
- Technology knowledge
  New products often emerge to take advantage of technological developments since their competitors were launched. If you understand the latest technology, you can design features to make use of it.

Factors in feature set design

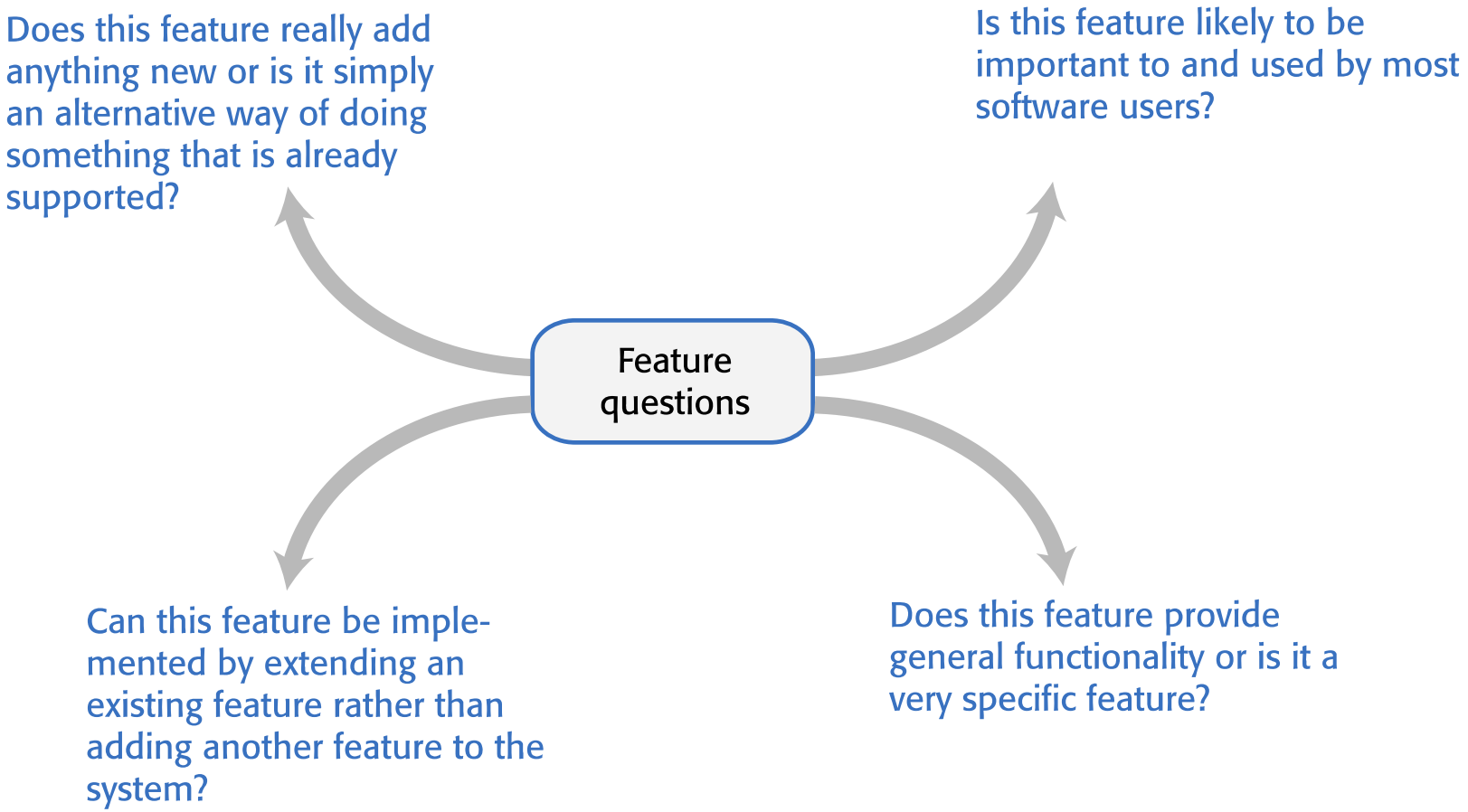


Feature trade-offs

- Simplicity and functionality
  - You need to find a balance between providing a simple, easy-to-use system and including enough functionality to attract users with a variety of needs.
- Familiarity and novelty
  - Users prefer that new software should support the familiar everyday tasks that are part of their work or life. To encourage them to adopt your system, you need to find a balance between familiar features and new features that convince users that your product can do more than its competitors.
- Automation and control
  - Some users like automation, where the software does things for them. Others prefer to have control. You have to think carefully about what can be automated, how it is automated and how users can configure the automation so that the system can be tailored to their preferences.

Feature creep

- Feature creep occurs when new features are added in response to user requests without considering whether or not these features are generally useful or whether they can be implemented in some other way.
- Too many features make products hard to use and understand
- There are three reasons why feature creep occurs:
  - Product managers are reluctant to say ‘no’ when users ask for specific features.
  - Developers try to match features in competing products.
  - The product includes features to support both inexperienced and experienced users.



Feature derivation

- Features can be identified directly from the product vision or from scenarios.
- You can highlight phrases in narrative description to identify features to be included in the software.
  - You should think about the features needed to support user actions, identified by active verbs, such as use and choose.

The iLearn system 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

Features from the product vision

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

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. *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.

Features from Jack’s scenario

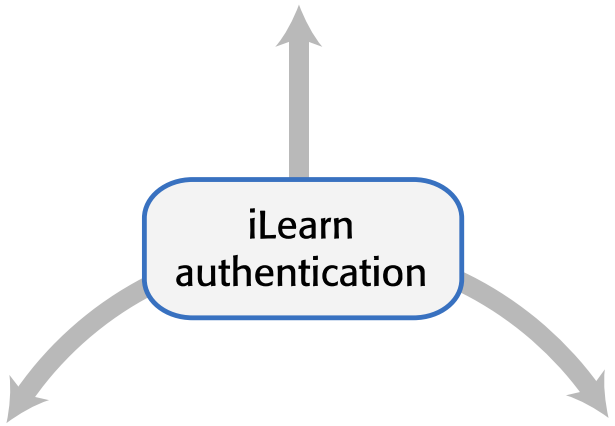
- A wiki for group writing.
- Access to the SCRAN history archive. This is a shared national resource that provides access to historical newspaper and magazine articles for schools and universities.
- Features to set up and access an email group.
- A feature to integrate applications with the iLearn authentication service.

The feature list

- The output of the feature identification process should be a list of features that you use for designing and implementing your product.
- There is no need to go into a lot of detail about the features at this stage. You add detail when you are implementing the feature.
- You can describe features using a standard input-action-output template by using structured narrative descriptions or by a set of user stories.

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.

Feature description using user stories

- Description
  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

Innovation and feature identification

- Scenarios and user stories should always be your starting point for identifying product features.
  - Scenarios tell you how users work at the moment. They don’t show how they might change their way of working if they had the right software to support them.
  - Stories and scenarios are ‘tools for thinking’ and they help you gain an understanding of how your software might be used. You can identify a feature set from stories and scenarios.
- User research, on its own, rarely helps you innovate and invent new ways of working.
- You should also think creatively about alternative or additional features that help users to work more efficiently or to do things differently.

Key points 1

- A software product feature is a fragment of functionality that implements something that a user may need or want when using the product.
- The first stage of product development is to identify the list of product features in which you identify each feature and give a brief description of its functionality.
- Personas are ‘imagined users’ where you create a character portrait of a type of user that you think might use your 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 your 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 2

- Scenarios should always be written from the user's perspective and should be based on identified personas or 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.
- You can identify features from scenarios and stories by highlighting user actions in these narratives and thinking about the features that you need to support these actions.