SSW-540: Fundamentals of Software Engineering

Software Requirements Engineering

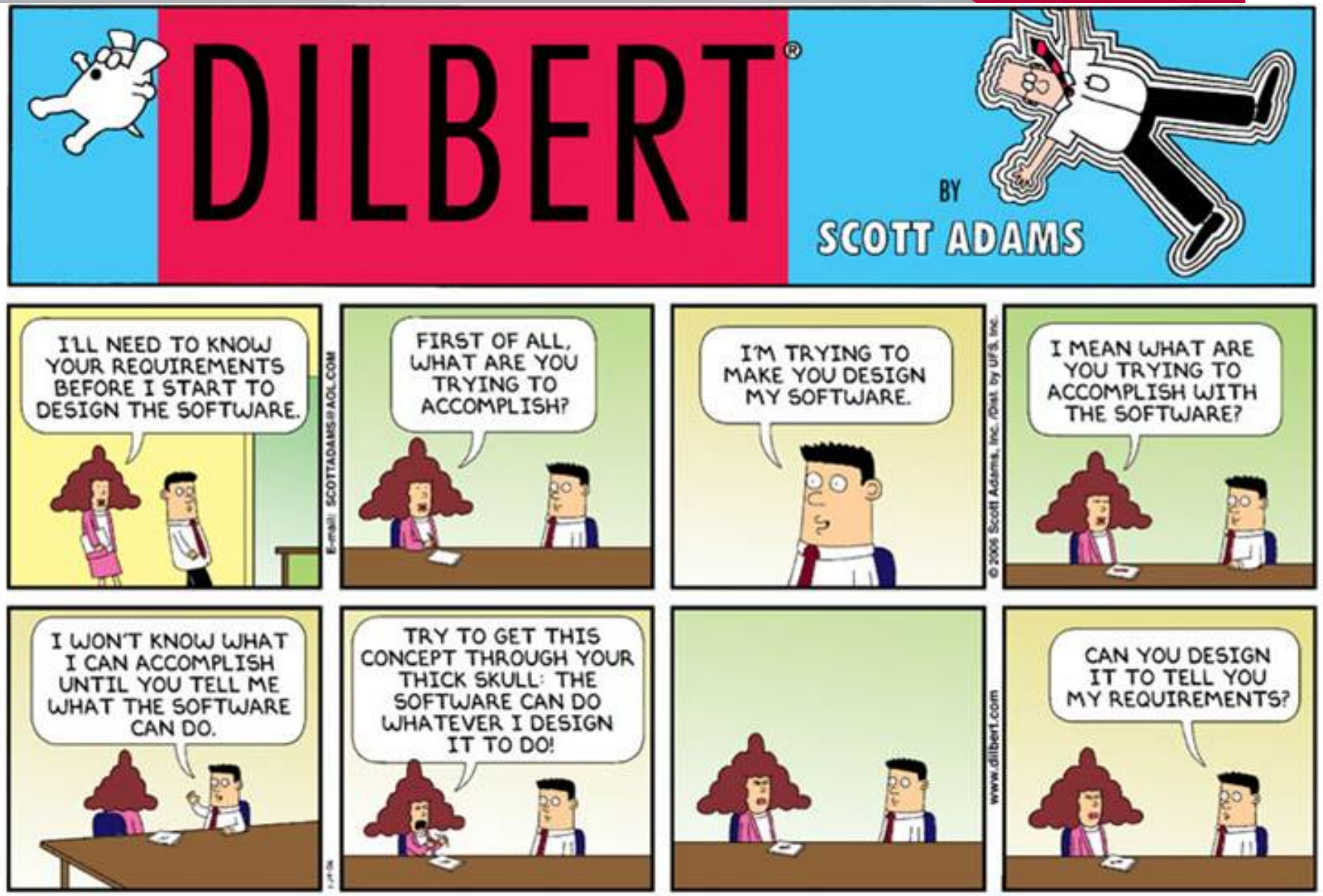
Dr. Richard Ens
School of Systems and Enterprises

Python Pointers

- Headers
  - Good practice to have some heading material in every Python script that includes your name as author and the purpose of the script. Some IDE's insert a header to which you can add details.
    - A simple comment will suffice: **#** Author: R. Cohen; This script does whatever.
    - Docstrings are used when intro comments extend over several lines:
      """This is a docstring. It can extend over several lines.
      Just end it with 3 quotation marks like you started it."""
- Exiting
  - Different Python interpreters treat the token **exit** in different ways.
  - Best practice is to not use **exit()**. Instead **import sys** library at the start of your script and use **sys.exit()** wherever you would have used **exit()**.
  - (Or, don't exit, just **break** out of whatever loop you are in and continue.)

Requirements engineering

- An engineering process to
  - establish which services the customer requires from a system
  - and the constraints under which it operates and is developed
- The requirements themselves are the descriptions of the system services and constraints that are generated during the requirements engineering process.
- The form of the requirements and the details of the requirements engineering process will vary with the type of software to be built and the type of organization that will build it.



What is a requirement?

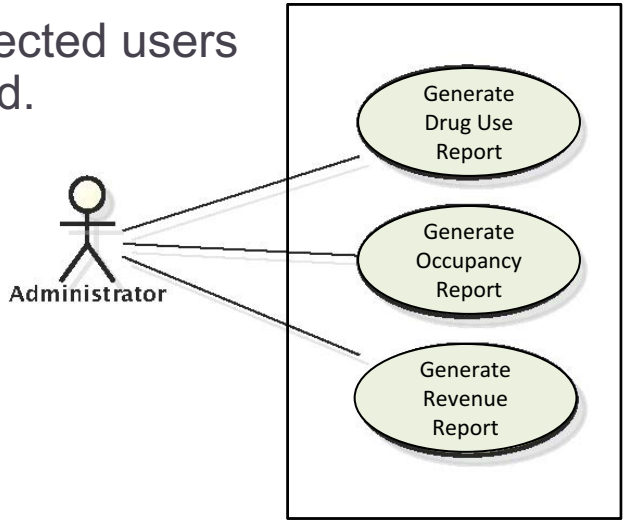
- It may range from a high-level abstract statement of a service or of a system constraint to a detailed mathematical functional specification.
- Requirements may serve a dual function
  - May be the basis for a bid for a contract - therefore must be open to interpretation;
  - May be the basis for the contract itself - therefore must be defined in detail;
  - Both these statements may be called requirements.

Agile methods and requirements

- Many agile methods argue that producing detailed system requirements is a waste of time as requirements change so quickly.
- The requirements document is therefore always out of date.
- Agile methods usually use incremental requirements engineering and may express requirements as ‘user stories’ (as we saw in Sommerville, Chapter 3).
- This is practical for business systems but problematic for systems that require pre-delivery analysis (e.g. critical systems) or systems developed by several teams.

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; what the system is supposed to accomplish
- Functions describe external interactions (inputs, outputs and behaviors). E.g., a use case
- Functional requirements should describe the system services in detail.
  - Detail level depends on the type of software, expected users and the type of system where the software is used.
- Also may state what the system should not do



Requirements imprecision

- Problems arise when requirements are not precisely stated.
- Ambiguous requirements may be interpreted in different ways by developers and users.
- Consider the term ‘search’ in the patient information system requirement:

  “A user shall be able to search the appointments lists for all clinics.”

  Does this mean:

  - A user can search for a patient name across all appointments in all clinics; or
  - A user can search for a patient name in an individual clinic after choosing the clinic from among all clinics?

Requirements completeness and consistency

- In principle, requirements should be both complete and consistent.
  - Complete
    - They should include descriptions of all required capabilities and constraints.
  - Consistent
    - There should be no conflicts or contradictions in the descriptions of the system capabilities and constraints.
- Think about how the requirement can be tested as you write it
- In practice, it is almost impossible to produce a totally complete and consistent requirements document for anything but the smallest software systems.

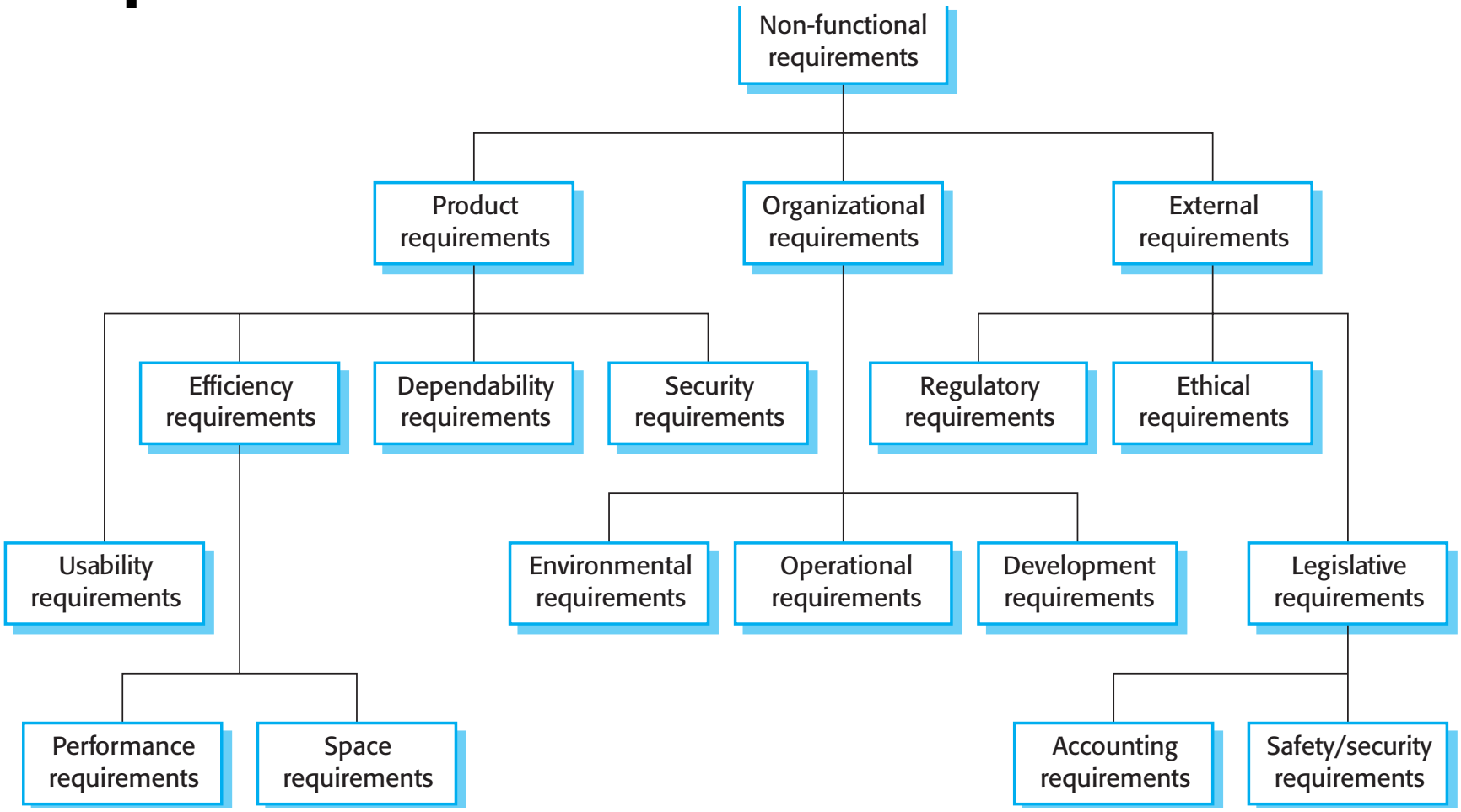
Non-functional requirements

- Constraints on the services or functions offered by the system such as timing constraints, and constraints on the development process, standards, etc.
- Often apply to the system as a whole rather than individual features or services
- NFR's define how the system operates, not what a system does
- Parameterized qualities of system including
  - Usability
  - Performance
  - Scalability
  - Reliability
  - Maintainability, et al.
- If the system does not meet its NFRs, the system may be useless!

Non-functional requirements implementation

- Non-functional requirements often 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.

Myriad of non-functional requirements



Distinguishing the functional from the non-functional

- Functional requirements describe what a software system will do; non-functional requirements specify how the system will be.
- Functional requirements often describe an action taken by a software system; non-functional requirements often describe an overall property of a software system.
- A functional requirement describes a behavior of a software system; non-functional requirements elaborate a characteristic of the software system as a whole.

Why one must know the difference

- Modularization
  - Functionality is usually provided within distinct modules (components) so that functions can be removed or added easily
  - Non-functional constraints or qualities are often implemented via architectural properties including relationships and connectors. Often, you don't want to lose a non-functional constraint or quality by the removal of a component
- Accommodation
  - Clients/customers (stakeholders) can tell you what a software system must do, but they often don't know much about the properties that system must have.
  - Explicit NFR's assure that those properties are addressed.

Clues to tell the difference

- Is it to be a choice on a menu or button? (F)
- Does the requirement focus on the experience of the user? (NF)
- Is the primary characterization a verb or an activity (F)
- Is the primary characterization an adjective or a quality? (NF)
- Does the requirement specify a feature of the system or the work that is done? (F)
- Does the requirement specify a property of the system or the character of the work? (NF)

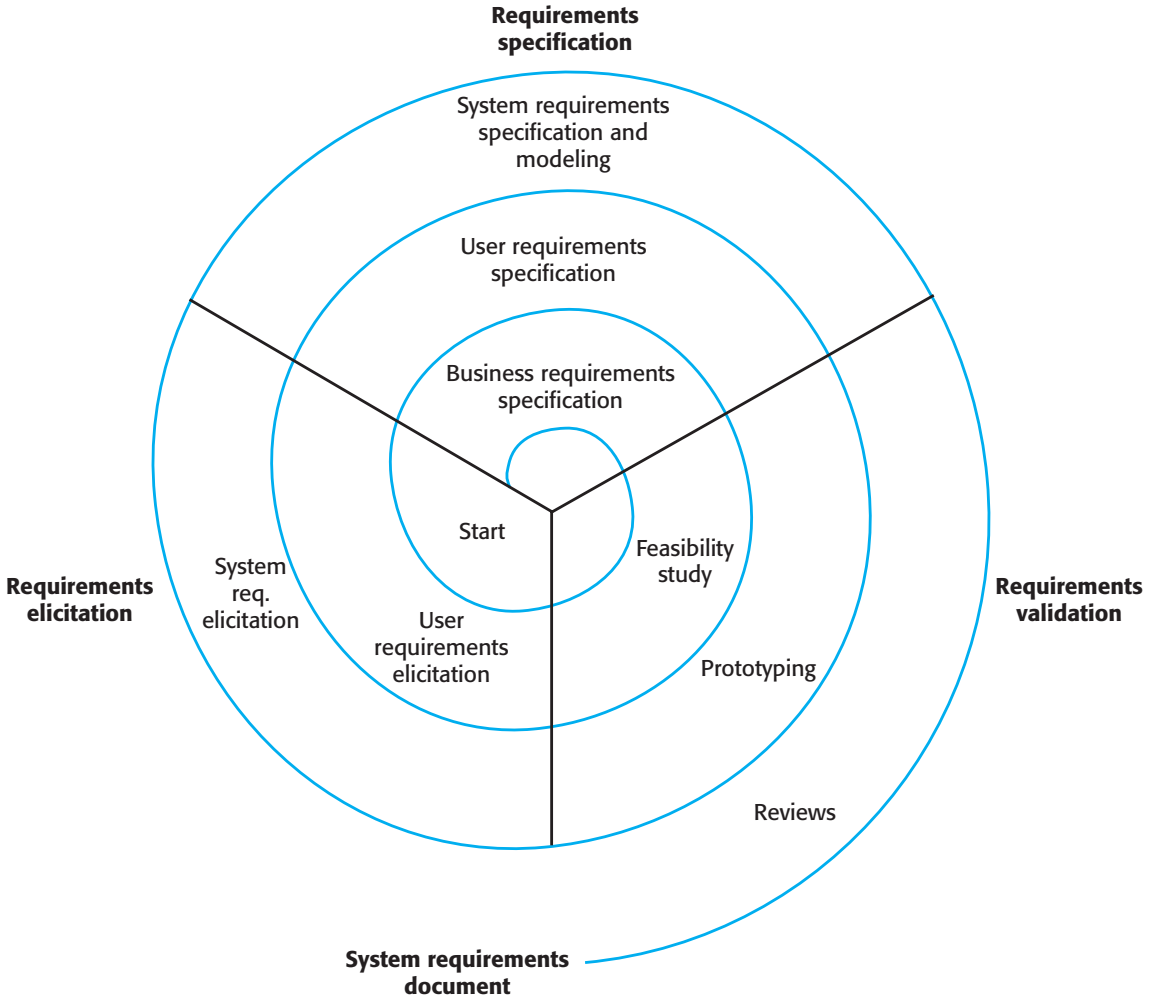
Goals versus requirements

- Non-functional requirements may be very difficult to state precisely but imprecise requirements may be difficult to verify
- Goal
  - A general intention of the user such as ease of use
- Goals are helpful to developers as they convey the intentions of the system users—but they are not, generally verifiable
- Verifiable non-functional requirement
  - A statement using some measure that can be objectively tested
  - (Your textbook includes some examples of NFR metrics)

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.

A spiral view of the requirements engineering process



Requirements elicitation and analysis

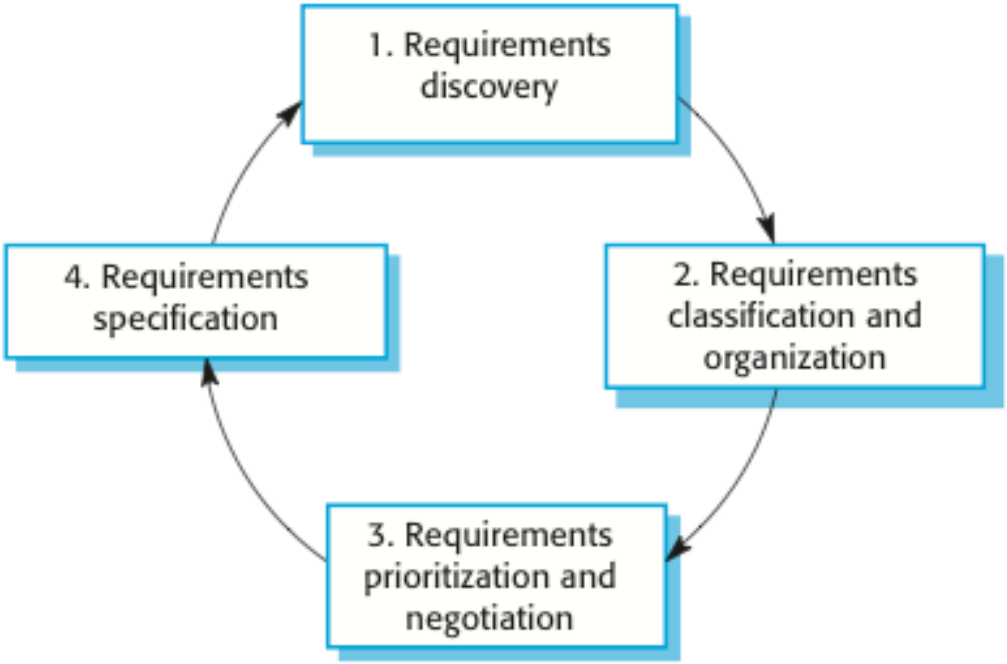
- Sometimes called requirements elicitation or requirements discovery.
- Involves technical staff working with customers to find out about the application domain, the services that the system should provide and the system's operational constraints.
- May involve end-users, managers, engineers involved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

Problems in requirements elicitation

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

Requirements elicitation and analysis stages

- Requirements discovery,
- Requirements classification and organization,
- Requirements prioritization and negotiation,
- Requirements specification.



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 a range of system stakeholders from managers to external regulators.
- Possible techniques:
  - Interviewing, open-ended or with pre-determined questions
  - Ethnography, possibly combined with prototyping
  - Stories and (more structured) scenarios, UML use cases

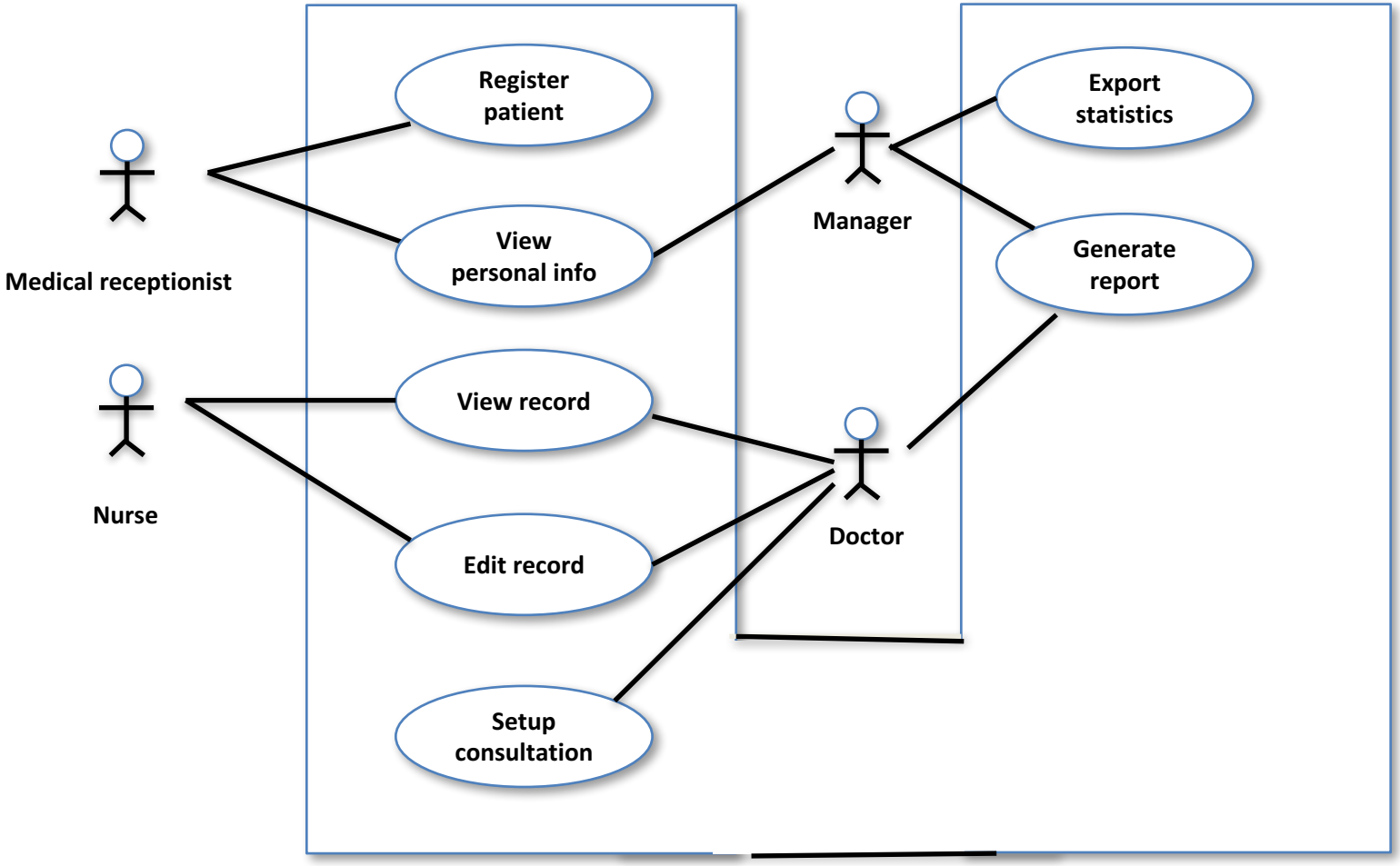
Requirements specification

- The process of writing down the user and system requirements
  - Format will vary (e.g., Agile stories vs. requirements documents)
- User requirements have to be understandable by stakeholders who do not have a technical background.
- System requirements are more detailed requirements and will include more technical information.
- The requirements may be part of a contract for the system development (so they must be as complete as possible)
- Ways of documenting software requirements:
  - Natural language, perhaps structured with forms or templates
  - Graphical notations like UML
  - Formal language, mathematics, finite-state machines

UML Use cases

- Use cases are a scenario based technique in the UML which identify the actors in an interaction and describe the interaction itself.
- A full set of use cases should describe all possible interactions with the system.
- High-level graphical model supplemented by more detailed tabular description.
- Sequence diagrams may be used to add detail to use cases by showing the sequence of event processing in the system.

Some use cases for the patient management system



Simplified Use Case Template

Item	Content
ID	[*unique identifier*]
Name	[*the use case name*]
Actors	[*human and/or systems interacting*]
Description	[*the steps needed to fulfill the use case*]
Data	[*if any*]
Stimulus	[*triggers for this use case*]
Response	[*what success looks like*]
Comments	[*any preconditions or special considerations*]

Simplified Use Case Template

Item	Content	Sample (simple) content
ID	[*unique identifier*]	UC001
Name	[*the use case name*]	Generate Patient Report
Actors	[*human and/or systems interacting*]	Manager
Description	[*the steps needed to fulfill the use case*]	1. User logs in 2. User accesses DB 3. User requests report
Data	[*if any*]	Patient DB
Stimulus	[*triggers for this use case*]	none
Response	[*what success looks like*]	List of current patients is generated
Comments	[*any preconditions or special considerations*]	User must be (pre)authorized to access these data and the contents of this report must be pre-specified within the system

Requirements Analysis

- Requirements analysis is performed throughout the requirements documentation and specification activities and consists of
  - Requirements classification and organization
    - Coherent organization of requirements reduces redundancies and promotes comprehension
  - Requirements prioritization and negotiation
    - Multiple stakeholders will hold conflicting requirements
    - Resolution, often via compromise, is essential to move the project forward

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
- We check requirements for
  - 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 tested within the implementation?

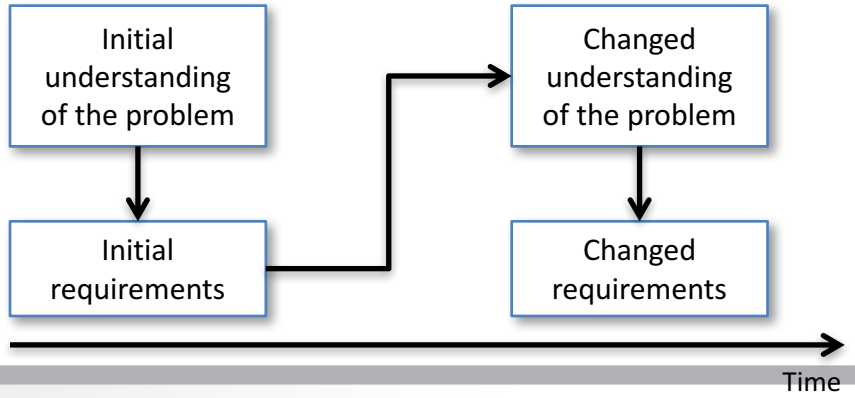
Requirements validation techniques

- Requirements reviews
  - Systematic manual analysis of the requirements checking for
    - Verifiability
    - Comprehensibility
    - Traceability
    - Adaptability
- Prototyping
  - Using an executable model of the system to check requirements.
- Test-case generation
  - Developing tests for requirements to check testability before implementation begins

Requirements Management

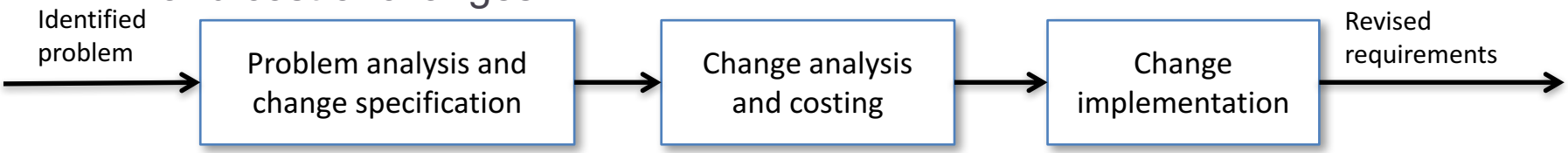
Requirements management is the process of managing changing requirements during the requirements engineering process, system development and deployment.

- New requirements emerge as a system is being developed and after it has gone into use.
- One must keep track of individual requirements and maintain links between dependent requirements so that the impact of requirements changes can be accurately assessed.
- In large software systems, a formal process is needed for making change proposals and linking these to software requirements.



Requirements management planning

- Establishes the level of requirements management detail that is required.
- Requirements management decisions:
  - *Requirements identification* -Each requirement must be uniquely identified so that it can be cross-referenced with other requirements.
  - *A change management process* -The set of activities that assess the impact and cost of changes.



  - *Traceability policies* -These policies define the relationships between each requirement and between the requirements and the system design that should be recorded.
  - *Tool support* -Tools that may be used ranging from specialist requirements management systems to spreadsheets and simple database systems.

Tools for Requirements Management

- http://makingofsoftware.com/resources/list-of-rm-tools lists 101 tools for requirements management, 25 of which are "noteworthy" (and listed below)

✧ Some interface with integrated development environments (IDEs)

✧ "Selected" tools:

  - Agile Manager (HP), Blueprint, CA Agile Central, Caliber, codeBeamer Requirements Management, Cognition Cockpit, Enterprise Architect, HPE ALM/Quality Center, IBM Rational **DOORS** (& DOORS Next Generation), inSTEP BLUE, Innovator for Business Analysts, inteGREAT, **Jama**, **Jira** Software, Kovair ALM Studio, Mingle, objectiF RPM (was inSTEP RED), Polarion Requirements, Serena Dimensions RM, TestTrack RM, TopTeam Analyst, VersionOne, Visure Requirements

Eliciting requirements is not always easy

Watch the 7½ minutes video on gathering requirements

- On Canvas in Module Week 3
- On YouTube at http://www.youtube.com/watch?v=lXNu0VBVCUc

Classroom Activity – within your EMSS project teams

Activity:

Within your team, draft the primary functional requirements for the EMSS as UML Use Cases, identifying the actors and the probable "sunny-day" scenario/steps for each use case. Note that actors are the humans and systems that will interact with your Energy Management Software System.

Readout to the class should be your use case diagram identifying the use cases you will be working with, insofar as you have defined them during this class period.

Submission to complete the Canvas Assignment EMSS Week 3:

Document each of your use cases using the simplified use case template presented in this lecture and submit those with your Use Case diagram as the functional requirements for your system.

Python *Pearl*

Python offers the ability to exit a loop completely when an external condition is triggered or you want to skip part of the loop and start the next execution. Python uses **break** and **continue** statements to do this.

The **break** statement terminates the current loop and resumes execution at the next statement. It is used in both *while* and *for* loops.

The **continue** statement returns the control to the beginning of the while loop.

[findGrade is a function returning a grade]

```
while x!="q":
        try:
                score = raw_input ("Enter a score from 0 to 100, or enter q to quit: ")
                number = float(score)
                grade = findGrade (number)
                print grade
        except:
                if score=="q":
                        break
                print "This script requires a number."
```