Master of Science
in
Computer Engineering

Software Systems Engineering
a.a. 2021-2022
Gigliola Vaglini

1

Requirements Engineering

Lecture 3

2

2

What is Requirements engineering

- The process of establishing the services that a customer requires from a system and the constraints under which the system operates and is developed.

3

3

What is a requirement

- It may be the basis for a bid for a contract - therefore must be open to interpretation
  - "If a company wishes to let a contract for a large software development project, it must define its needs in a sufficiently abstract way that a solution is not pre-defined. The requirements must be written so that several contractors can bid for the contract, offering, perhaps, different ways of meeting the client organization's needs."
- It may be the basis for the contract itself - therefore must be defined in detail
  - "Once a contract has been awarded, the contractor must write a system definition for the client in more detail so that the client understands and can validate what the software will do. Both of these documents may be called the requirements document for the system."

4

4

Types of requirement

- 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.

5

5

Functional requirements

- User requirements
  - Statements in natural language plus diagrams of the services the system provides and its operational constraints. Written for customers.
- System requirements
  - A structured document setting out detailed descriptions of the system's functions, services and operational constraints. Defines what should be implemented so may be part of a contract between client and contractor.
  - They may state what the system should not do.

6

6

User and system requirements: difference

- The system shall generate monthly reports containing the cost…
- The system shall generate the report for printing after 17.30 on the last working day of the month

7

7

Requirements imprecision

- Problems arise when functional requirements are not precisely stated since they may be interpreted in different ways by developers and users.
- Consider the term 'search' in the following requirement for a clinic information system
  - A user shall be able to search the appointments lists for all doctors using the system.
    - User intention – search for a name across all appointments;
    - Developer interpretation – search for a name in an individual doctor list. User chooses doctor then search.

8

8

Requirements completeness and consistency

- In principle, requirements should be both complete and consistent.
  - Complete
    - They should include descriptions of all facilities required.
  - Consistent
    - There should be no conflict or contradiction in the descriptions of the system facilities.
- In practice, because of system and environmental complexity, it is impossible to produce a complete and consistent requirements document.

9

9

Types of requirement (2)

✧Non-functional requirements

- Constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc.

10

10

Non-functional requirements

- Non-functional requirements may be more critical than functional requirements. If these are not met, the system may be useless.
- Non-functional requirements may affect the overall architecture of a system rather than the individual components.
  - to ensure that performance requirements are met, you may have to organize the system so that communications between components are minimized.
- A single security requirement may generate a number of related functional requirements.
  - It may also generate requirements that restrict existing requirements.

11

11

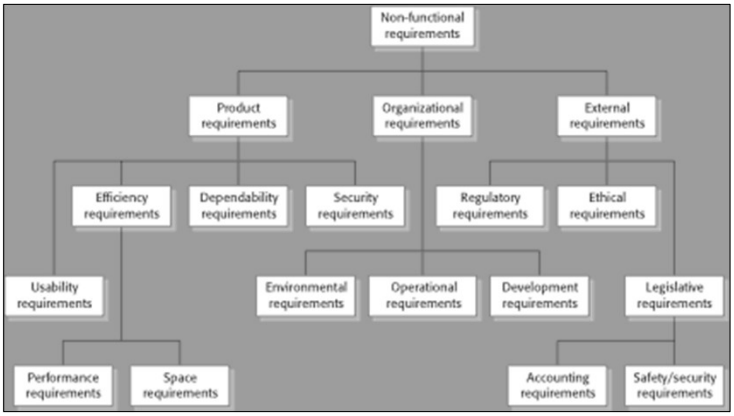
Non-functional requirements classification

- Product requirements
  - Requirements which specify that the delivered product must behave in a particular way e.g. execution speed, reliability, etc.
    - Downtime within working hours shall not exceed 5 seconds in one day
- Organisational requirements
  - Requirements which are a consequence of organisational policies and procedures.
    - Users shall authenticate themselves using their employee identification number
- External requirements
  - Requirements which arise from factors which are external to the system, e.g. interoperability requirements, legislative requirements, etc.
    - The system shall implement privacy provisions as set out in the standard…

12

12

Types of non-functional requirement



13

13

Non-functional requirements specification

- Non-functional requirements may be very difficult to state precisely and imprecise requirements may be difficult to verify: Some measure that can be objectively tested should be associated with each requirement.
  - The system should be easy to use and should be organized in such a way that user errors are minimized. (Not verifiable)
  - The 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)

14

14

Metrics for specifying nonfunctional requirements

Property	Measure
Speed	Processed transactions/second User/event response time Screen refresh time
Size	Mbytes 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

15

15

Types of requirements (3)

- Domain requirements
  - the operational domain imposes constraints on the system.
    - For example, a train control system has to take into account the braking characteristics in different weather conditions.
  - If domain requirements are not satisfied, the system may be unworkable.

16

16

Domain requirements problems

- It is difficult for a non-specialist to understand the implications of a domain requirement and how it interacts with other requirements.
- Understandability
  - Requirements are expressed in the language of the application domain.
- Implicitness
  - Domain specialists understand the area so well that they do not think of making the domain requirements explicit.

17

17

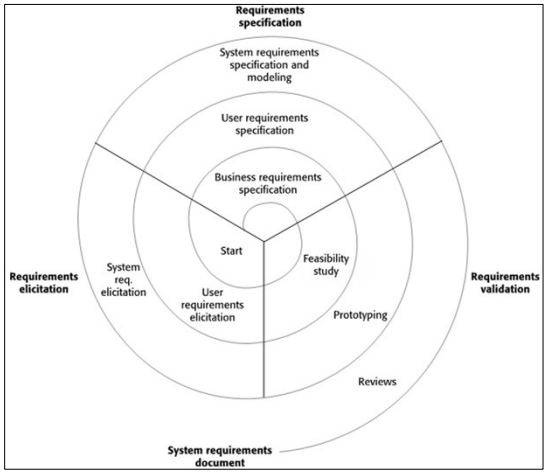
Requirements engineering process

- 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
  1. Requirements elicitation;
  2. Requirements analysis;
  3. Requirements validation;
  4. Requirements management.
- In practice, RE is an iterative activity in which these processes are interleaved.

18

18

A spiral view



19

19

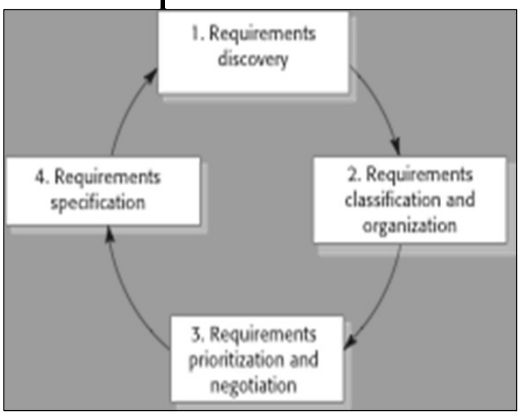
Requirements elicitation and analysis (activities 1,2)

- This phase involves technical staff working with customers, and in general stakeholders, to find out about the application domain, the services that the system should provide and the system's operational constraints.
- Graphical means (as UML Use cases, for example) were developed originally to support requirements elicitation.

20

20

The requirements elicitation and analysis phase is again a cyclic process



21

21

Process activities

- ✧ Requirements discovery
  - Interacting with stakeholders to discover their requirements. Domain requirements are also discovered at this stage.
- ✧ Requirements classification and organisation
  - related requirements are grouped and organised 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 loop.

22

22

Problems of requirements elicitation and analysis

- Stakeholders don't know what they really want.
- Stakeholders express requirements in their own terms.
- Different stakeholders may have conflicting requirements.
- New stakeholders may emerge and the business environment change.

23

23

Interviewing stakeholders

- 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.
  - Normally a mix of closed and open interviewing.
- Results
  - Interviews are good for getting an overall understanding of what stakeholders expect from and how they might interact with the system.
  - Interviews are not good for understanding domain requirements because
    - engineers cannot understand specific domain terminology

24

24

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.

25

25

Requirements validation (activity 3)

- Concerned with demonstrating that the requirements define the system that the customer really wants. To be checked:
  - 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?

26

26

Requirements management (activity 4)

- Requirements management is the process of managing changing requirements.
- You need to 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.

27

27

Change management activity is a cyclic process too

✧ Deciding if a change should be accepted

- *Problem analysis and change specification*
  - During this stage, the problem or the change proposal is analyzed to check that it is valid. This analysis is fed back to the change requestor to be considered.
- *Change analysis and costing*
  - The effect of the proposed change is assessed using traceability information and general knowledge of the system requirements. Once this analysis is completed, a decision is made whether or not to proceed with the requirements change.
- *Change implementation*
  - The requirements document and, where necessary, the system design and implementation, are modified.

28

28

Requirements document

- The document specifying the requirements, i.e. the services that the customer requires and the constraints under which the system shall operate, should exist.
  1. It reduces the development effort.
     - The preparation of a document forces the customers to consider rigorously all of the requirements before design begins and reduces later redesign, recoding, and retesting;
  2. It provides a basis for estimating costs and schedules;
  3. It facilitates transfer.
     - To new users and new machines

29

29

Agile methods and requirements documents

- ✧ Many agile methods argue that producing a requirements document is a waste of time as requirements change so quickly. The document is therefore always out of date.
- ✧ Methods such as XP use incremental requirements engineering. Systems developed incrementally will, typically, have less detail in the requirements document.
- ✧ Such behavior is practical for business systems but problematic for systems that require a lot of pre-delivery analysis (e.g. critical systems) or systems developed by several teams.
- ✧ Requirements document standards are mostly applicable to the requirements for large systems engineering projects.

30

30

Requirements and design

- In principle, requirements should state what the system should do and the design should describe how it does this.
- In practice, requirements and design are inseparable
  - A system architecture may be designed to structure the requirements
  - The use of a specific architecture to satisfy non-functional requirements may be a domain requirement in consequence of a regulatory requirement

31

31

Ways of writing a requirement 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

32

32

Guidelines for writing requirements in a structured manner

- Invent a standard format and use it for all requirements.
- Use language in a consistent way. Use shall for mandatory requirements.
- Avoid conjunctions (and, or) that make a complex requisite.
- Avoid the use of computer jargon.
- Include an explanation (rationale) of why a requirement is necessary.

33

Some format rules

- Ubiquitous: The <system name> shall <system response>
- Event-driven: WHEN <trigger> <optional precondition> the <system name> shall <system response>
- State-driven: WHILE <system state>, the <system name> shall <system response>
- Unwanted behavior: IF <unwanted condition or event>, THEN the <system name> shall <system response>
- Optional: WHERE <feature is included>, the <system name> shall <system response>

34

34

Example requirement in natural language for an insulin pump

3.2.The system shall measure the blood sugar and deliver an insulin dose, when the sugar level is increasing and the increasing rate is increasing, every 10 minutes. *(Changes in blood sugar are relatively slow so more frequent measurement is unnecessary; less frequent measurement could lead to unnecessarily high sugar levels.)*

35

35

Using the structured natural language

- The Insulin Pump shall measure the blood sugar every 10 minutes
- WHEN the measure of the sugar level is increasing and the increasing rate is increasing the Insulin Pump shall deliver the computed dose (CompDose)

36

36

When writing a complex (or multiple) requirement

- *The navigator shall be able to view the aircrafts position relative to the route's radio beacons or as estimated by the inertial guidance.*
  - The form above is potentially dangerous because it is not clear whether the "or" indicates that the navigator can choose which method to use for navigation or that the developers can decide which to implement.
- Better to write
  - *The navigator shall be able to view the aircraft's position relative to the route's radio beacons.*
  - *The navigator shall be able to view the aircraft's position as estimated by inertial guidance.*

37

37

Using a form

- 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.

38

38

Another structured specification of the requirement for the insulin pump

Insulin Pump/Control Software	
Function	Compute insulin dose: CompDose.
Description	
Computes the dose of insulin to be delivered when the current measured sugar level is increasing and increasing rate is increasing.	
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.

39

39

3.2. revised

Action
CompDose is computed.
Requirements
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.

40

40

A supplement: tabular specification

- 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 needed in different scenarios.

41

41

Tabular specification for computing CompDose

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

42

42

Structured specifications at the end

- This type of specifications work well for requirements of embedded control system, but is sometimes too rigid for writing business system requirements.

43

43

Use cases and Sequence diagrams

- Use-cases are a kind of scenario included in the UML.
- Use cases identify the actors in an interaction and describe the interaction itself.
- They are supplemented by more detailed tabular description.
- UML sequence diagrams may be used to add detail to use-cases by showing the sequence of events processed in the system.

44

44

The complete software requirements document

- The software requirements document is the official statement of what is required of the system developers.
- Should include both the definition of the user requirements and of the system requirements.
- It is NOT a design document.

45

45

The structure of a requirements document

Chapter	Description
Preface	This should define the expected readership of the document and describe its version history, including a rationale for the creation of a new version and a summary of the changes made in each version.
Introduction	This should describe the need for the system. It should briefly describe the system's functions and explain how it will work with other systems. It should also describe how the system fits into the overall business or strategic objectives of the organization commissioning the software.
Glossary	This should define the technical terms used in the document. You should not make assumptions about the experience or expertise of the reader.
User requirements definition	Here, you describe the services provided for the user. The nonfunctional system requirements should also be described in this section. This description may use natural language, diagrams, or other notations that are understandable to customers. Product and process standards that must be followed should be specified.
System architecture	This chapter should present a high-level overview of the anticipated system architecture, showing the distribution of functions across system modules. Architectural components that are reused should be highlighted.

46

46

The structure of a requirements document

Chapter	Description
System requirements specification	This should describe the functional and nonfunctional requirements in more detail. If necessary, further detail may also be added to the nonfunctional requirements. Interfaces to other systems may be defined.
System models	This might include graphical system models showing the relationships between the system components and the system and its environment. Examples of possible models are object models, data-flow models, or semantic data models.
System evolution	This should describe the fundamental assumptions on which the system is based, and any anticipated changes due to hardware evolution, changing user needs, and so on. This section is useful for system designers as it may help them avoid design decisions that would constrain likely future changes to the system.
Appendices	These should provide detailed, specific information that is related to the application being developed; for example, hardware and database descriptions. Hardware requirements define the minimal and optimal configurations for the system. Database requirements define the logical organization of the data used by the system and the relationships between data.
Index	Several indexes to the document may be included. As well as a normal alphabetic index, there may be an index of diagrams, an index of functions, and so on.

47

47

Requirements document standards

- Requirements documents standards have been designed e.g. IEEE standard. These are mostly applicable to the requirements for large systems engineering projects.

48

48

IEEE Std 830-1998

- IEEE Std 830-1998 is the IEEE Recommended Practice for Software Requirements Specifications and describes the recommended approaches to this task.
- It is based on a model in which the result of the software requirements specification process is a specification document.
- Here documentation is the main point

49

49

Software Requirements Specification (SRS)

- The specification document is called Software Requirements Specification (SRS), which must be complete, correct, consistent, unambiguous and understandable both by customers and suppliers.

50

50

Considerations for producing a good SRS

- Some information should be considered when writing an SRS.
  1. Nature of the SRS;
  2. Environment of the SRS;
  3. Characteristics of a good SRS;
  4. Joint preparation of the SRS;
  5. SRS evolution;
  6. Validation;
  7. Embedding design in the SRS.

51

51

1. Nature of the SRS

The basic issues that the SRS writer(s) shall address are the following:

- *Functionality.*
  - What is the software supposed to do?
- *External interfaces.*
  - How does the software interact with people, the system's hardware, other hardware and other software?
- *Performance.*
  - What is the speed, availability, response time, recovery time of various software functions, etc.?
- *Attributes.*
  - What are the portability, correctness, maintainability, security, etc. considerations?
- *Design constraints imposed on an implementation.*
  - Are there any required standards in effect, implementation language, policies for database integrity, resource limits, operating environment(s) etc.?

52

52

2. Environment of the SRS

- A software requirement may exist because of the nature of the task to be solved or because of a special characteristic of the project.
- A properly written SRS limits the range of valid designs, but does not specify any particular design and not impose additional constraints on the software.

53

53

3. SRS quality attributes

- **A good SRS must be**

a) UNAMBIGOUS
b) CORRECT
c) COMPLETE
d) VERIFIABLE
e) CONSISTENT
f) MODIFIABLE
g) TRACEABLE
h) RANKED (for importance and/or stability)

54

54

3.a Unambigous

An SRS is UNAMBIGOUS if, and only if, every requirement stated therein has only one interpretation. In particular,

1) As a minimum, this requires that each characteristic of the final product is described using a single unique term.

2) In cases where a term used in a particular context could have multiple meanings, the term must be included in a glossary where its meaning is made more specific.

55

55

3.b Correct

- An SRS is correct if, and only if, every requirement stated therein is one that the software shall meet.

56

56

3.c Complete

An SRS is COMPLETE if it posses the following qualities:

1) Inclusion of all significant requirements, whether relating to functionality, perfomance, design constraints, attributes or external interfaces.

2) Definition of the responses of the software to all reliable classes of input data in all realizable classes of situations. Note that it is important to specify the responses to valid and invalid input values.

3) Full labelling and referencing of all figures, tables, and diagrams in the SRS and definition of all terms and units of the measure.

57

57

3.c Complete (cont.)

Any SRS that uses the phrase

... TBD (To Be Determined) ...

is not a complete SRS. If it is ncessary, it should be accompanied by:

1) A description of the conditions causing the TBD (for example, why an answer is not known) so that the situation can be solved.

2) A description of what must be done to eliminate the TBD.

58

58

3.d Verifiable

An SRS is VERIFIABLE if and only if every requirement stated therein is verifiable. A requirement is verifiable if and only if there exists some finite cost-effective process with which a person or machine can check that the software product meets the requirement.

If a method cannot be devised to determine whether the software meets a particular requirement then that requirement be removed or revised.

59

59

3.e Consistent

Consistency refers to internal consistency.

An SRS is CONSISTENT if and only if no set of individual requirements described in it conflict. There are three types of likely conflicts in an SRS:

1) two or more requirements might describe the same real world object but use different terms for that objects.

2) the specified characteristic of the real world object might conflict.

3) there might be a logical or temporal conflict between two specified actions.

60

60

3.f Modifiable

An SRS is MODIFIABLE if and only if its structure and style are such that any necessary changes to the requirements can be made easily, completely and consistently. Modifiability generally requires an SRS to:

- have a coerent and easy-to-use organization, with a table of contents, an index, and explicit cross-referencing;
- not to be redundant;
  - whenever redundancy is necessary, the SRS should include explicit cross-references to make it modifiable.
- Express each requirement separately, rather than intermixed with other requirements.

61

61

3.g Traceable

An SRS is TRACEABLE if the origin of each of its requirements is clear and if it facilitates the referencing of the requirement in future development or enhancement documentation. Two types of traceability are recommented:

1) Backward Traceability: depends upon each requirement explicitly referencing its source in previous documents.
   - requisito collegabile a qualche elemento del progetto o del codice
2) Forward Traceability: depends upon each requirement in the SRS having a unique name or reference number.

62

62

3.h Ranked

An SRS is RANKED for importance and/or stability if each requirement in it has an identifier to indicate either the importance or stability of that particular requirement.

Typically, all of the requirements that relate to a software product are not equally important. Some requirements may be essential, while others may be desiderable.

Each requirement in the SRS should be identified to make these differences clear and explicit

63

63

4. Joint preparation of the SRS

- Supplier and customer should agree on what the completed software must do. This agreement has the form of an SRS.
- Therefore, the customer and the supplier should work together to produce a well-written and completely understood SRS.

64

64

5. SRS evolution

- The SRS may need to evolve as the development of the software product progresses. It may be impossible to specify some details at the time the project is initiated. Two major considerations in this process are the following:
  - Requirements should be specified as completely and thoroughly as is known at the time, even if evolutionary revisions can be foreseen as inevitable.
  - The fact that they are incomplete should be noted.
- A formal change process should be initiated to identify, control, track, and report projected changes.
- Approved changes in requirements should be incorporated in the SRS in such a way as to
  - Provide an accurate and complete audit trail of changes;
  - Permit the review of superseded portions of the SRS.

65

65

6. Requirements validation

- Concerned with demonstrating that the requirements define the system that the customer really wants, before delivery.
  - Fixing a requirement error after delivery may cost up to 100 times the cost of fixing an implementation error.
- The most relevant technique for validating requirements is prototyping, i.e. requirements are checked by means of an executable model
- Another technique is
  - Review, i.e. a systematic manual analysis of the requirements.

66

66

6.1 Prototyping

- Many tools exist that allow a prototype, exhibiting some characteristics of a system, to be created very quickly and easily. Prototypes are useful for the following reasons:
  - The customer may be more likely to view the prototype and react to it than to read the SRS and react to it. Thus, the prototype provides quick feedback.
  - The prototype displays unanticipated aspects of the systems behavior. Thus, it produces not only answers but also new questions.
  - An SRS based on a prototype tends to undergo less change during development, thus shortening development time.

67

67

6.2 Review

- The software product is presented to project personnel, managers, users, customers, user representatives, or other interested parties for comment or approval.
- A systematic evaluation of the software is performed to examine the suitability of the product for its intended use and identify discrepancies from specifications. Technical reviews may also provide examination and recommendations of alternatives.
- The review has a low cost and reveals at least 60% of errors [Boehm]
- Example
  - Requirements analysis of CTC (Centralised Traffic Controller) – North American Railway – 1990
    - 10 groups of concurrent analists, each group discovered 25 anomalies
    - 77 anomalies are discovered during SRS inspection, 15 are discovered during the next phases.

68

68

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.

69

69

IEEE Std 1028

- **IEEE Standard for Software Reviews**
- This standard describes how to carry out a review.
- Software reviews can be used also in support of the final verification and validation of the software.

70

70

7. Embedding design in the SRS

- A requirement specifies an externally visible function or attribute of a system. The SRS writer(s) should clearly distinguish between identifying required design constraints and projecting a specific design. Note that every requirement in the SRS limits design alternatives. This does not mean, though, that every requirement is design.
- The SRS should specify what functions are to be performed on what data to produce what results at what location for whom. The SRS should not normally specify design items such as the following:
  - Partitioning the software into modules;
  - Allocating functions to the modules;
  - Describing the flow of information or control between modules;
  - Choosing data structures.

71

71

7.1 Exceptions

- In special cases some requirements may severely restrict the design. For example, security or safety requirements may reflect directly into design since they require that
  - certain functions are kept in separate modules;
  - only limited communication between some areas of the program are permitted.
- Other examples of valid design constraints are physical requirements, performance requirements, software development standards, and software quality assurance standards.

72

72

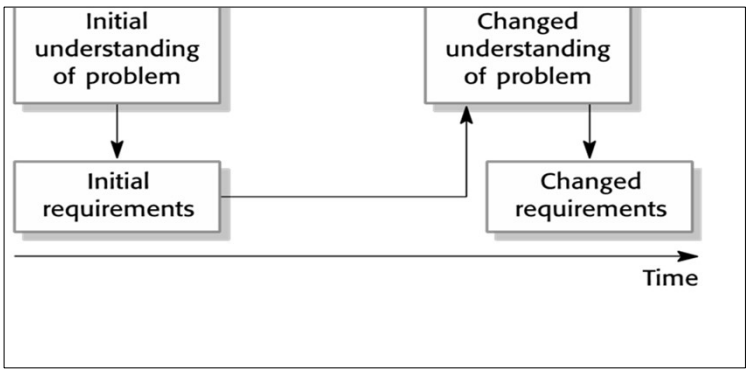
Changing requirements

- The business and technical environment of the system always changes after installation.
  - New hardware may be introduced, it may be necessary to interface the system with other systems, business priorities may change, and new legislation and regulations may be introduced that the system must necessarily abide by.
- The people who pay for a system and the users of that system are rarely the same people.
  - System customers impose requirements because of organizational and budgetary constraints. These may conflict with end-user requirements and, after delivery, new features may have to be added for user support if the system is to meet its goals.

73

73

Requirements evolution



74

74

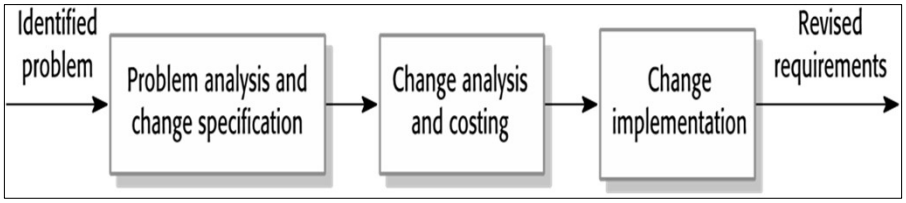
Deciding if a requirements change should be accepted

- *Problem analysis and change specification*
  - the change proposal is analyzed to check that it is valid. This analysis may produce a more specific requirements change proposal, or the withdrawal of the request.
- *Change analysis and costing*
  - The effect of the proposed change is assessed using traceability information and general knowledge of the system requirements. This analysis produces the decision whether or not to proceed with the change.
- *Change implementation*
  - The requirements document, the system design and implementation are modified. Ideally, the document should be organized so that changes can be easily implemented.

75

75

Requirements change management



76

76

System specifications

Lecture 4

77

System models

- Models of the existing system are used during requirements engineering. They help to clarify what the existing system does and can be used as a basis for discussing its strengths and weaknesses.
- Models of the new system are used during requirements engineering to help explain the proposed requirements to other system stakeholders. Engineers use these models to discuss design proposals and to document the system for implementation.
- In a model-driven engineering process, it is possible to automatically generate a complete or partial system implementation from the system model.

78

78

Software systems specification

- Different types of systems have different specification needs and it is necessary to choose the
- suitable specification language

79

79

Software systems specification (2)

- Systems are often specified by means of some kind of graphical notation, which is now almost always based on notations in the Unified Modeling Language (UML).
- Actually we define different abstract models of the system, with each model presenting a different view or perspective.

80

80

Models give different perspectives of the system

- A structural perspective, where you model the organization of a system or the structure of the data that is processed by the system.
- An external perspective, where you model the context or environment of the system.
- An interaction perspective, where you model the interactions between a system and its environment, or between the components of a system.

81

81

(High level) specification of an embedded system

✧ A personal insulin pump is

- an embedded system used by diabetics to maintain blood glucose control
  - A first description can regard the hard part of the system, i.e. its components and the connections among components
- a safety-critical system as low blood sugar levels can lead to brain malfunctioning, coma and death; high blood sugar levels have long-term consequences such as eye and kidney damage.
  - It is necessary to individuate the set of unwanted behaviors

82

82

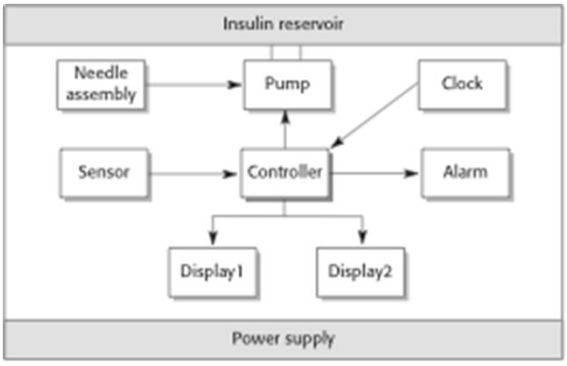
Structural models

- Structural models display the organization of a system in terms of the components and their relationships.
- Structural models may be static models, which show the structure of the system design, or dynamic models, which show the organization of the system when it is executing.
- You create structural models of a system when you are discussing and designing the system architecture.

83

83

Insulin pump hardware architecture



84

84

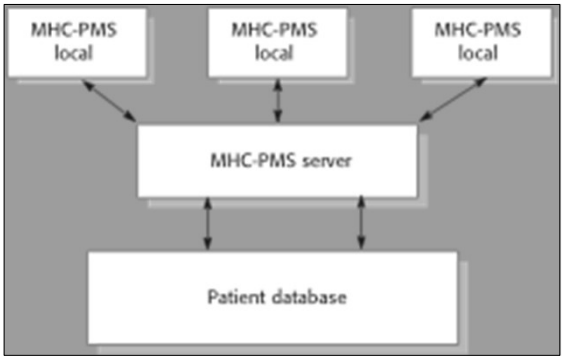
Another type of system

- An information system exploiting a central data base and running on different PC to read and write data
- In this case we have a software architecture

85

85

Software architecture of an information system



86

86

A UML diagram type to define structural models: class

- Class diagrams show the object classes and the associations between these classes; they give a static description of the system.
- An object class can be thought of as a general definition of one kind of system object, where objects represent something in the real world.
- An association is a link between classes that indicates that there is some relationship between them.

87

87

UML classes and association

88

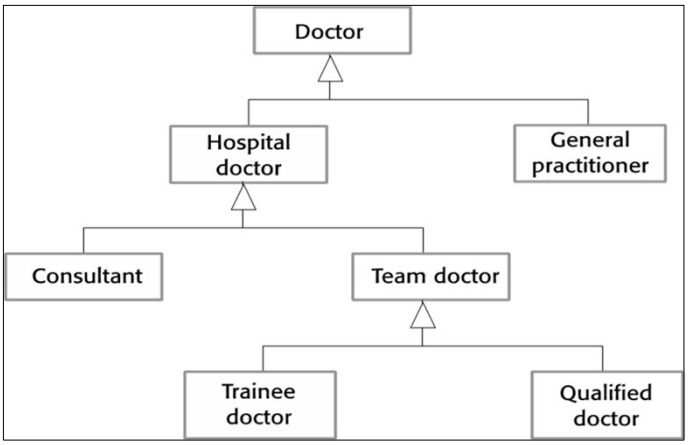
88

- In modeling systems, it is often useful to examine the classes in a system to see if there is scope for generalization.
  - In a generalization, the attributes and operations associated with higher-level classes are also associated with the lower-level classes.
  - The lower-level classes then add more specific attributes and operations.
- An aggregation model shows how classes that are collections are composed of other classes.

89

89

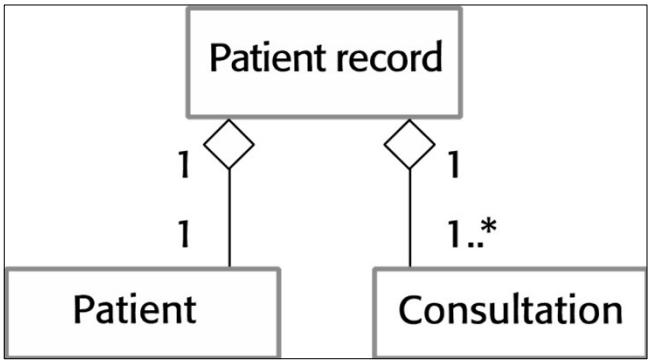
Generalizations



90

90

Aggregations



91

91

Dynamic models: activity and state diagrams

- Dynamic models show what happens or what is supposed to happen when a system responds to a stimulus from its environment. The control algorithm of the system is defined by means of
  - a set of states and a way in which it is possible to pass from a state to another
- Two types of UML diagrams exist
- Activity diagrams show the activities involved in a process or in a data processing.
- State diagrams show how the system reacts to internal and external events.

92

92

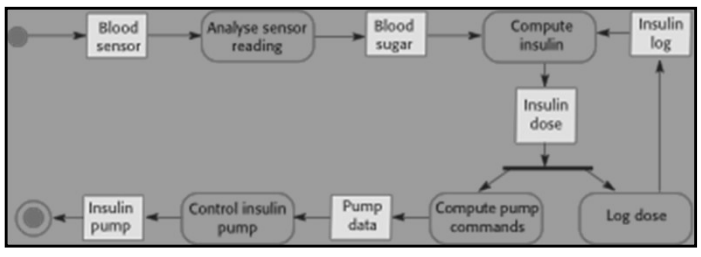
Dynamic (or behavioral) models (1)

- The stimuli from the environment can be :
  - Data input data must be processed by the system to produce an output. In UML this can be accomplished by means of an activity diagram whose semantics is a Petri net. This diagram shows the sequence of actions involved in processing input data.

93

93

Activity model of the insulin pump



94

94

Dynamic (or behavioral) models (2)

- The stimuli from the environment can be :
- Events Some event happens that triggers system processing. Events may have associated data, although this is not always the case.
- Real-time systems are often event-driven, with minimal data processing, and event-driven models show how a system responds to some external and internal events. The basic assumption is that the system has a finite number of states and an event may cause a transition from one state to another.
- Statechart is the diagram type used in the UML to represent event-driven systems and it is an extension of classical state machine models.

95

95

External perspective: Context models

- Context models are used to show what lies outside the system boundaries.
  - They simply show the other systems in the environment, not how the system being developed is used in that environment.
- The position of the system boundary has a profound effect on the system requirements.
  - Social and organisational concerns may affect the decision on where to position system boundaries.
  - A context model helps to decide what features should be implemented in this system and what features are supposed in other associated systems.

96

96

For example,

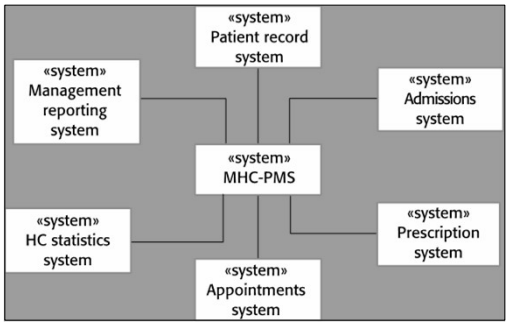
- The system generates monthly management reports …..
  - Then it exists an Administrative reporting system storing such reports
- The system generates monthly a scheduling report …..
  - Then there exists a system to manage appointments that uses the scheduling of the month

✧ And so on ……

97

97

The context of the information system



98

98

Interaction perspective: the first step

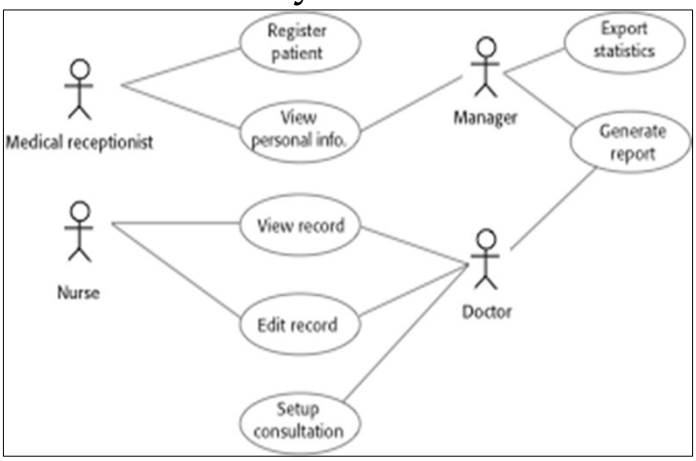
- We identify the actors (people or things external to the system, but that use the system)
- Then we define the use cases (things that actors can do with system), and the relationship between actors and use cases

- UML use case diagrams show the interactions between the system and its environment.

99

99

Use cases for a medical information system



100

100

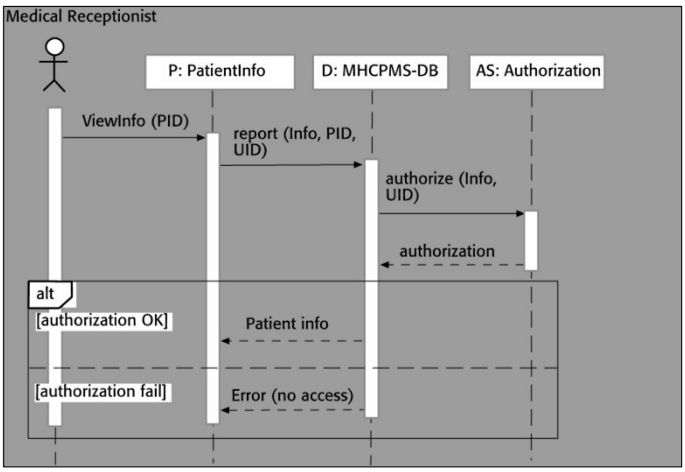
Interaction perspective: sequence diagrams

- UML Sequence diagrams show interactions between actors and the system and between system components; they show the sequence of interactions that take place during a particular use case.
- The objects and actors involved are listed along the top of the diagram, with a dotted line drawn vertically from these.
- Interactions between objects are indicated by annotated arrows.

101

101

Sequence diagram for view personal information



102

102

Interaction perspective: advantages

- Modeling user interaction helps to understand user requirements.
  - Use cases and Sequence diagrams can be used to specify system requirements
- Modeling component interaction allows to understand when the required performance can be obtained.
- Modeling system-to-system interaction highlights the communication problems that may arise.

103

103

Finally: Model-driven engineering

- Model-driven engineering (MDE) is an approach to software development where models rather than programs are the principal outputs of the development process.
- The programs that execute on a hardware/software platform are then generated automatically from the models.
- Proponents of MDE argue that this raises the level of abstraction in software engineering so that engineers no longer have to be concerned with programming language details or the specifics of execution platforms.

104

104

Pros and Cons of MDE

- Model-driven engineering is still at an early stage of development, and it is unclear whether or not it will have a significant effect on software engineering practice.
- Pros
  - Allows systems to be considered at higher levels of abstraction
  - Generating code automatically means that it is cheaper to adapt systems to new platforms.
- Cons
  - Models for abstraction are not necessarily right for implementation.
  - Savings from generating code may be outweighed by the costs of developing translators for new platforms.

105

105

Model-driven architecture

- Model-driven architecture (MDA) was the precursor of more general model-driven engineering
- MDA is a model-focused approach to software design and implementation that uses a subset of UML models to describe a system.
- Models at different levels of abstraction are created. From a high-level, platform independent model, it is possible, in principle, to generate a working program without manual intervention.

106

106

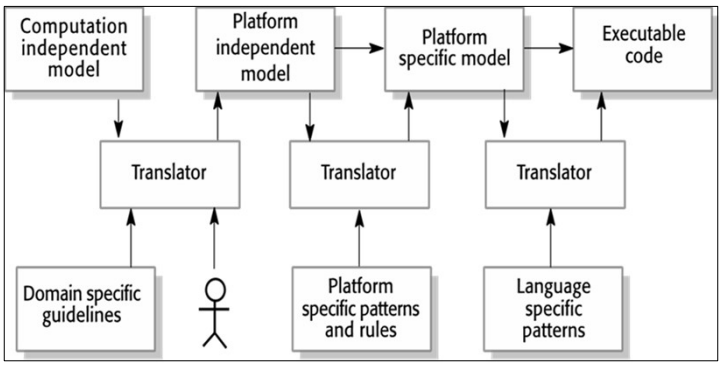
Types of models

- A computation independent model (CIM)
  - The important domain abstractions used in a system are modeled. CIMs are sometimes called domain models.
- A platform independent model (PIM)
  - The operation of the system is modeled without reference to its implementation. The PIM is usually described using UML models that show the static system structure and how it responds to external and internal events.
- Platform specific models (PSM)
  - The platform-independent model is transformed with a separate PSM for each application platform. In principle, there may be layers of PSM, with each layer adding some platform-specific detail.

107

107

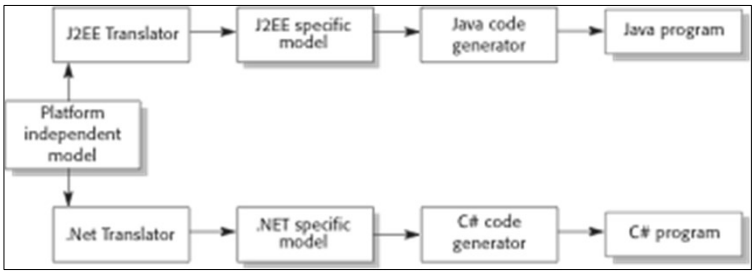
MDA transformations



108

108

Multiple platform-specific models



109

109

Executable UML

- Completely automated transformation of models to code is possible using a subset of UML 2, called Executable UML or xUML, where the number of different models has been dramatically reduced.

110

110

Adoption of MDA

- A range of factors has limited the adoption of MDE/MDA
  - Specialized tool support is required to convert models from one level to another, but there is limited tool availability and organizations require tool customisation to their environment
  - The arguments for platform-independence are only valid for large, long-lifetime systems. For other software products, the savings from the use of MDA are likely to be outweighed by the costs of its introduction and tooling.

111

111

Adoption of MDA (2)

- Models are a good way of facilitating discussions about a software design but they may not be the right abstractions for implementation.
- For most complex systems, implementation is not the major problem – requirements engineering, security and dependability, integration with legacy systems and testing are all more significant.

112

112

- The widespread adoption of agile methods over the same period that MDA was evolving has diverted attention away from model-driven approaches.

113

113

Agile methods and MDE

- The developers of MDE claim that it is intended to support an iterative approach to development and so can be used within agile methods.
- But the notion of extensive up-front modeling contradicts the fundamental ideas in the agile manifesto and few agile developers feel comfortable with model-driven engineering.
- If transformations can be completely automated and a complete program generated from a PIM, then, in principle, MDE could be used in an agile development process as no separate coding would be required.

114

114