SOFTWARE ENGINEERING

CO3001

LECTURE 2&3 – REQUIREMENTS ENGINEERING

Anh Nguyen Duc
Jingyue Li
Daniela Soares Cruzes

NTNU

Adapted from https://iansommerville.com/software-engineering-book/slides/

TOPICS COVERED

- ✓ Requirements engineering & Requirements
- ✓ Requirements elicitation
- ✓ Requirements specification
- ✓ Requirements validation
- ✓ Requirements management
- ✓ Goal oriented requirement engineering
- ✓ Quality of requirements
- ✓ Non-functional requirements
- ✓ Requirement traceability

TOPICS COVERED

- ✓ Requirements engineering & Requirements
- ✓ Requirements elicitation
- ✓ Requirements specification
- ✓ Requirements validation
- ✓ Requirements management
- ✓ Goal oriented requirement engineering
- ✓ Quality of requirements
- ✓ Non-functional requirements
- ✓ Requirement traceability

REQUIREMENTS

EFFECTS OF INADEQUATE REQUIREMENTS DEVELOPMENT – AIRBUS

- ✓ **Requirement**: *Reverse thrust may only be used <u>when the airplane is landed</u>.*
- ✓ **Translation**: *Reverse thrust may only be used <u>while the wheels are rotating</u>.*
- ✓ **Implementation**: *Reverse thrust may only be used <u>while the wheels are rotating fast enough</u>.*

- **Situation**: Rainstorm – aquaplaning
- **Result**: Crash due to overshooting the runway!
- **Problem**: Erroneous in the requirement phase

The Ariane 5 accident – 1

- Single root cause failure!
- **The "bug": attitude deviation** stored as 2-byte integer (max value 65,535) instead of 4-byte (max value 4,294,967,295)
- SW module was reused from Ariane 4
- Insufficient V&V of detailed requirements: larger attitude deviation tolerated in Ariane 5 than in Ariane 4
- Ariane 5 production cost 10 years and $7 billion; luckily, no victims because it was unmanned.

65,53**5** = 00000000 00000000 11111111 11111111

65,53**6** = 00000000 00000001 00000000 00000000

OTHER SOFTWARE RELATED ACCIDENTS & INCIDENTS

- **Accidents & incidents**
  - Alarm flooding, **power distribution** failure, BP Grangemouth Scotland, 29th May - 10th June 2000
  - failure in a data bus + faulty logic in the software => engine power loss, **Airbus** A340-642, 2005
  - failed accelerometer + software bug => Faulty airspeed metering, **Boeing** 777-200, 2005
  - Software bug => shutdown of radio communication between **ATC and aircraft**, 2004. disrupted 800 flights
  - Safety-related software flaws => recall of 200,000 **pacemakers** in 1990-2000
  - **radiotherapy machines** attacked by computer **viruses**, 2005
  - Buggy software incorporate computer + connection between corporate and control systems networks => shutdown of the **nuclear power plant**, USA 2008
- Many software failures are actually "bugs" in the **detailed requirement specifications,** i.e., poor understanding of the very detailed requirements

Korean Air 747 in Guam, 200 deaths (1997): incorrect configuration of "minimum altitude" warning system

How the customer explained it
How the project leader understood it
How the engineer designed it
How the programmer wrote it
How the sales executive described it
How the project was documented
What operations installed
How the customer was billed
How the helpdesk supported it
What the customer really needed

REQUIREMENTS ENGINEERING

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

WHAT IS A REQUIREMENT?

Requirement engineering = establishing the ***services*** that the customer requires from a system and the ***constraints*** under which it operates and is developed.

- ✓ Requirement = the descriptions of
  - the system services
  - and constraints
- ✓ It may range
  - from a high-level abstract statement
  - to a detailed mathematical functional specification.
- ✓ May serve a dual function
  - The basis for a bid for a contract - must be open to interpretation;
  - The basis for the contract itself - must be in detail;

STAKEHOLDERS

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

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

NON-FUNCTIONAL REQUIREMENTS IMPLEMENTATION

- ✓ 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
  - cross-cutting concern
- ✓ A single non-functional requirement
  - may generate a number of related functional requirements
  - and may also generate requirements that restrict existing requirements.

NON-FUNCTIONAL REQUIREMENTS

- ✓ Define system properties and constraints
  - Properties: reliability, response time and storage requirements.
  - Constraints: I/O device capability, system representations, etc.

- ✓ Non-functional requirements may be more critical than functional requirements.
  - If these are not met, the system may be useless.

NON-FUNCTIONAL REQUIREMENTS IMPLEMENTATION

- ✓ Non-functional requirements may affect the overall architecture of a system
  - rather than the individual components
  - cross-cutting concern

- ✓ A single non-functional requirement
  - may generate a number of related functional requirements
  - and may also generate requirements that restrict existing requirements.

REQUIREMENTS ENGINEERING PROCESS

REQUIREMENTS ENGINEERING PROCESSES

- ✓ Processes to "generate" all requirements
- ✓ Generic activities common to all processes
  - Requirements elicitation;
  - Requirements analysis;
  - Requirements validation;
  - Requirements management.
- ✓ In practice, RE is an iterative activity

"The client kept changing the requirements on a daily basis, so we decided to freeze them until the next release."

The RE process:

A SPIRAL VIEW OF THE REQUIREMENTS ENGINEERING PROCESS

A AGILE VIEW OF THE REQUIREMENTS ENGINEERING PROCESS

Iterative Enhancement Model

REQUIREMENT ELICITATION

REQUIREMENTS ELICITATION AND ANALYSIS

- ✓ ~ requirements elicitation or requirements discovery.

- ✓ Work with customers to find out:
  - the application domain, the services and the operational constraints (system performance, hardware constraints, etc.).
- ✓ May involve
  - end-users, managers, engineers involved in maintenance, domain experts, trade unions, etc. These are called stakeholders.

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 change.

THE REQUIREMENTS ELICITATION AND ANALYSIS PROCESS

REQUIREMENTS DISCOVERY

- ✓ To gather information about the required and existing systems and distil the user and system requirements from this information.

- ✓ Main concerns:
  - Stakeholders
  - Discovery techniques/approaches/…

DISCOVERY TECHNIQUE - INTERVIEWING

- ✓ Part of most RE processes.
- ✓ Types of interview
  - Closed vs Open => mixed?
- ✓ Be effective
  - 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.

DISCOVERY TECHNIQUE - ETHNOGRAP

- ✓ Observational technique
  - used to understand operational p
    requirements for these processes
- ✓ How
  - 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.

STORIES AND SCENARIOS

- ✓ Scenarios and user stor system can be used.
- ✓ Stories and scenarios a be used for a particulc
- ✓ Because they are base can relate to them and respect to the story.

REQUIREMENT ELICITATION TECHNIQUES

	Interviews	Workshops	Focus groups	Observations	Questionnaires	System interface analysis	User interface analysis	Document analysis
Mass-market software	x		x		x			
Internal corporate software	x	x	x	x		x		x
Replacing existing system	x	x		x		x	x	x
Enhancing existing system	x	x				x	x	x
New application	x	x				x		
Packaged software implementation	x	x		x		x		x
Embedded systems	x	x				x		x
Geographically distributed stakeholders	x	x			x			

Suggested elicitation techniques by project characteristics

REQUIREMENTS SPECIFICATION

REQUIREMENTS SPECIFICATION

- ✓ The process of writing down the user and system requirements in a requirements document.

- ✓ Notes:
  - User requirements have to be understandable by end-users and customers who do not have a technical background.
  - System requirements are more detailed requirements and may include more technical information.
  - The requirements may be part of a contract for the system development

WAYS OF WRITING A SYSTEM REQUIREMENTS SPECIFICATION

Notation	Description
Natural language	Sentences in natural language. Each sentence should express one requirement.
Structured natural language	Natural language statements on a standard form or template
Design description languages	Uses a language like a programming language, but with more abstract features
Graphical notations	Graphical models, supplemented by text annotations (best for functional requirements); UML use case and sequence diagrams are commonly used.
Mathematical specifications	Based on mathematical concepts such as finite-state machines or sets; Can reduce the ambiguity but hard to understand (and hard to check manually)

NATURAL LANGUAGE SPECIFICATION

- ✓ Used for writing requirements because it is expressive, intuitive and universal.
  - The requirements can be understood by users and customers.

- ✓ Problems
  - 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.

An insulin pump control system

An insulin pump is a medical system that simulates the operation of the pancreas (an internal organ). The software controlling this system is an embedded system, which collects information from a sensor and controls a pump that delivers a controlled dose of insulin to a user.

People who suffer from diabetes use the system. Diabetes is a relatively common condition where the human pancreas is unable to produce sufficient quantities of a hormone called insulin. Insulin metabolises glucose (sugar) in the blood. The conventional treatment of diabetes involves regular injections of genetically engineered insulin. Diabetics measure their blood sugar levels using an external meter and then calculate the dose of insulin that they should inject.

EXAMPLE REQUIREMENTS FOR THE INSULIN PUMP SOFTWARE SYSTEM

- ✓ Req 3.2. The system shall measure the blood sugar and deliver insulin, if required, every 10 minutes.
- ✓ Req 3.6. The system shall run a self-test routine every minute with the conditions to be tested and the associated actions defined in Table 1.

STRUCTURED SPECIFICATIONS

- ✓ Writing on a standard form or template:
  - Name
  - Inputs, outputs
  - The information needed for the computation
  - Action
  - Pre and post conditions (if appropriate)
  - The side effects (if any)

- ✓ This works well for some types of requirements e.g. requirements for embedded control system

A ST PUMP

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.
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

TABULAR SPECIFICATION

- ✓ Particularly useful when you have to define a number of possible alternative courses of action.
- ✓ Example:

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

USE CASES

- ✓ Use-cases are a kind of scenario
  - identify the actors in an interaction and which describe the interaction itself
  - Included in the UML

- ✓ A set of use cases should describe all possible interactions with the system.

- ✓ UML sequence diagrams may be used to add detail to use-cases
  - show the sequence of event processing in the system

USE CASE DIAGRAM

- Actors
- Use cases
- Association
- System boundary boxes

Partial use case diagram for the Chemical Tracking System (CTS)

NTNU

USE CASE TEMPLATE AND EXAMPLE

ID and Name:	UC-4 Request a Chemical		
Created By:	Lori	Date Created:	8/22/13
Primary Actor:	Requester	Secondary Actors:	Buyer, Chemical Stockroom, Training Database
Description:	The Requester specifies the desired chemical to request by entering its name or chemical ID number or by importing its structure from a chemical drawing tool. The system either offers the Requester a container of the chemical from the chemical stockroom or lets the Requester order one from a vendor.		
Trigger:	Requester indicates that he wants to request a chemical.		
Preconditions:	PRE-1. User's identity has been authenticated. PRE-2. User is authorized to request chemicals. PRE-3. Chemical inventory database is online.		
Postconditions:	POST-1. Request is stored in the CTS. POST-2. Request was sent to the Chemical Stockroom or to a Buyer.		

USE CASE TEMPLATE AND EXAMPLE (CONT')

ID and Name:	UC-4 Request a Chemical
Normal Flow:	**4.0 Request a Chemical from the Chemical Stockroom** 1. Requester specifies the desired chemical. 2. System lists containers of the desired chemical that are in the chemical stockroom, if any. 3. System gives Requester the option to View Container History for any container. 4. Requester selects a specific container or asks to place a vendor order (see 4.1). 5. Requester enters other information to complete the request. 6. System stores the request and notifies the Chemical Stockroom.
Alternative Flows:	**4.1 Request a Chemical from a Vendor** 1. Requester searches vendor catalogs for the chemical (see 4.1.E1). 2. System displays a list of vendors for the chemical with available container sizes, grades, and prices. 3. Requester selects a vendor, container size, grade, and number of containers. 4. Requester enters other information to complete the request. 5. System stores the request and notifies the Buyer.
Exceptions:	**4.1.E1 Chemical Is Not Commercially Available**

DATA FLOW DIAGRAM

- ✓ How data moves through a system

NTNU

DECISION TABLE/DECISION TREE

- ✓ Complex logics

Requirement Number					
Condition	**1**	**2**	**3**	**4**	**5**
User is authorized	F	T	T	T	T
Chemical is available	—	F	T	T	T
Chemical is hazardous	—	—	F	T	T
Requester is trained	—	—	—	F	T
Action					
Accept request			X		X
Reject request	X	X		X	

STATE-TRANSITION DIAGRAM

- ✓ A set of complex state changes in natural language creates a high probability of overlooking a permitted state change or including a disallowed change.

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.

REQUIREMENT VALIDATION

REQUIREMENTS VALIDATION

- ✓ Requirements error can be costly !

Cost to Fix Requirements Error (in Ratios)			
Development Discipline/Phase	**Composite of Studies (NASA)***	**NASA Software#**	**Davis' Composite+**
Requirements	1 (baseline)	1 (baseline)	1 (baseline)
Design	5x	8x	2.5x – 5x
Code	10x	16x	5x – 10x
Test	50.5x	21x	Unit Test: 10x – 20x Acceptance Test: 25x – 50x
Post-Deployment	n/a	29x	100x – 200x

* NASA reviewed McGibbon (2003), Pavlina (2003), Cigital (2003), Rothman (2002), Hoffman (2001), Rothman (2000), and Boehm (1981) and assessed the median of their data, presented here.
# NASA analyzed its own software projects to assess the relevance of reviewed studies.
# Leffingwell & Widrig reviewed GTE, TRW, IBM, and Davis (2003), who himself reviewed several studies. The results of Davis' work are presented here.

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.
- ✓ Test-case generation
  - Developing tests for requirements to check testability.

REQUIREMENTS MANAGEMENT

"The client kept changing the requirements on a daily basis, so we decided to freeze them until the next release."

CHANGING REQUIREMENTS

- ✓ The business and technical environment of the system always changes after installation.
- ✓ The people who pay for a system and the users of that system are rarely the same people.
- ✓ Large systems usually have a diverse user community, with many users having different requirements and priorities that may be conflicting or contradictory.

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.

REQUIREMENTS MANAGEMENT PLANNING

- ✓ Establishes the level of requirements management detail that is required.

- ✓ Requirements management decisions:
  - Requirements identification
  - A change management process
  - Traceability policies
  - Tool support

REQUIREMENTS CHANGE MANAGEMENT

- ✓ Deciding if a requirements change should be accepted
  - Problem analysis and change specification
  - Change analysis and costing
  - Change implementation

RESPONSE TO CHANGE: AGILE APPROACH!

SUMMARY

- ✓ Requirements: what the system should do and constraints on its operation and implementation.
- ✓ Functional requirements = the services
- ✓ Non-functional requirements = constraints (development & use)
  - apply to the system as a whole.
- ✓ The software requirements document (i.e. SRS) is an agreed statement of the system requirements.
- ✓ The RE process is an iterative process
  - requirements elicitation, specification and validation.

GOAL ORIENTED REQUIREMENT ENGINEERING

"The client kept changing the requirements on a daily basis, so we decided to freeze them until the next release."

WHAT IS GORE [3]?

- ✓ GORE is approach and modeling method to
  - Identify goals and sub-goals of software systems
  - Identify who and how to achieve the goals
  - Identify domain hypothesis and conflicts
- ✓ Similar principle to use cases and user stories
- ✓ Can be applicable to cyber-physical systems and user-system interaction systems

PROCESSES TO CREATE GORE MODELS

- ✓ Initial goal identification
- ✓ Eliciting new goals through WHY and HOW questions
- ✓ Identifying agents and responsibility assignments
- ✓ Identifying operations
- ✓ Anticipating obstacles
- ✓ Identifying domain hypothesis
- ✓ Handling conflicts

GOAL MODEL

- A goal is an objective the system under consideration should achieve
- Goal is condition or situation that can be achieved or not. Goal is used to define the functional requirements of the system.
- Goals provide a precise criterion for sufficient completeness of a requirements specification

GOAL MODEL

- Hierarchy and relationship of goals
- Goals can be of different types

GOAL MODEL USING BEHAVIOR AND SOFT GOAL TYPES

- Soft goal
  - Satisfaction cannot be established in a clear-cut sense
  - Useful for comparing alternative goal refinements
- Hard/behavioral goal
  - Satisfaction can be established through verification

GOAL MODEL USING BEHAVIOR GOAL TYPES

Informal temporal patterns

Achieve[TargetCondition]
Cease[TargetCondition]

Maintain[GoodCondition]
Avoid[BadCondition]

Improve[TargetCondition]
Increase[TargetQuantity]
Reduce[TargetQuantity]
Maximise[ObjectiveFunction]
Minimise[ObjectiveFunction]

- Achieve: Some target property to be eventually satisfied
- Maintain: Some target property to be permanently satisfied
- Avoid: Some target property to be permanently denied
- …

PROCESSES TO CREATE GORE MODELS

- ✓ Initial goal identification
- ✓ Eliciting new goals through WHY and HOW questions
- ✓ Identifying agents and responsibility assignments
- ✓ Identifying operations
- ➢ Anticipating obstacles
- ➢ Identifying domain hypothesis
- ➢ Handling conflicts

ELICITING GOALS

Each sub-goal is refined into finer-grained goals until we reach a software requirement associated with a single agent and expectation (assumption) on the environment.

CASE STUDY: TRAIN TRANSPORT SAFETY

NTNU

SOFT AND BEHAVIOR GOALS

ELICITING NEW GOALS THROUGH "HOW" QUESTIONS

CONTINUE ELICITING NEW GOALS THROUGH "HOW" QUESTIONS

NTNU

GOALS REFINEMENT

Mean-ends relationship ( )
Decomposition relationship ( )
Contribution relationship(→)
Dependency relationship ( )

ELICITING NEW GOALS THROUGH "WHY" QUESTIONS

- ✓ High level goals are identified by asking WHY questions

AATC SOFT GOALS

How

Why

LINKS:

+ : Contribute
- : Conflict
Double line : OR
Black circle : AND
… : More sub-goals

++ : Contribute strongly
-- : Conflict strongly

NTNU

PROCESSES TO CREATE GORE MODELS

- ✓ Initial goal identification
- ✓ Eliciting new goals through WHY and HOW questions
- ✓ Identifying agents and responsibility assignments
- ✓ Identifying operations
- ➢ Anticipating obstacles
- ➢ Identifying domain hypothesis
- ➢ Handling conflicts

OBJECT MODEL

- ✓ Agent: independent active object, e.g.,
  - Speed/Acceleration control system
  - On-board train controller
  - Tracking systems
- ✓ Agent has entities
  - E.g., tracking systems have several sensors

AGENT RESPONSIBILITY MODEL

- A **goal** has to be placed under the responsibility of an **agent**.

NTNU

PROCESSES TO CREATE GORE MODELS

- ✓ Initial goal identification
- ✓ Eliciting new goals through WHY and HOW questions
- ✓ Identifying agents and responsibility assignments
- ✓ Identifying operations
- ➢ Anticipating obstacles
- ➢ Identifying domain hypothesis
- ➢ Handling conflicts

OPERATION MODEL

- ✓ Operation is input-output relation over objects
- ✓ Defines state transitions
- ✓ Characterized by pre-, post-, and trigger conditions

OPERATION MODEL – TRAIN CONTROL SYS.

- ✓ Operation: SendCommandMessage
- ✓ Input: Train ID
- ✓ Output: Command to the train
- ✓ Pre-condition: Message not sent
- ✓ Post-condition: Message sent to correct train

State transition

PROCESSES TO CREATE GORE MODELS

- ✓ Initial goal identification
- ✓ Eliciting new goals through WHY and HOW questions
- ✓ Identifying agents and responsibility assignments
- ✓ Identifying operations
- ➢ Anticipating obstacles
- ➢ Identifying domain hypothesis
- ➢ Handling conflicts

ANTICIPATING OBSTACLES

- Obstacles (Possible failures that may negatively impact goal achievement)
  - CommandMsgNotIssued
  - CommandMsgIssuedLate
  - CommandMsgSentToWrongTrain

IDENTIFY DOMAIN HYPOTHESIS

- ✓ **Domain property**: A descriptive statement about the problem world.
- ✓ **Domain invariants:** Should hold invariably, regardless of how the system behaves. This usually corresponds to some physical laws. E.g. *The light is either on or off*.
- ✓ **Domain hypothesis**: To satisfy a service request, an infrastructure/environment to perform the service must be available.

IDENTIFYING DOMAIN HYPOTHESIS

✓ Domain hypothesis related to the goal: *Every track segment leading to a switch is ended with a gate*.

NTNU

HANDLING GOAL CONFLICTS

CAN GORE HELP DERIVE BETTER REQUIREMENTS ?

- ✓ "Requirements definition must say why a system is needed, based on current or foreseen conditions, which may be internal operations or an external market. It must say what system features will serve and satisfy this context. And it must say how the system is to be constructed. [1]"

Goal models

Agent models

Responsibility models

Operation models

Domain hypotheses

SUMMARY (CONT.)

- ✓ Requirements elicitation and analysis = iterative process
  - requirements discovery, classification and organization, negotiation and requirements documentation.
- ✓ Techniques for requirements elicitation
  - interviews, scenarios, use-cases and ethnography, etc.
- ✓ Requirements validation = checking the requirements
  - for validity, consistency, completeness, realism and verifiability.
- ✓ Business, organizational and technical changes inevitably
  - => changes to the requirements for a software system.
- ✓ Requirements management = managing and controlling the requirement changes.

- Electronic Healthcare Record (EHR) system is a repository of patient data in digital form, stored and exchanged securely, and accessible by multiple authorised users. As a normal flow, medication will be first prescribed by authorized nurses to a patient, after that the medication is dispensed and then taken by the patient. EHR ServerRX deployed in a London hospital, required all the patients to wear a bracelet ID for identification purposes. Nurses had to scan this bracelet before dispensing any medicine in order to reduce the risk of administering the medication to the wrong patient.
- Describe at least three sub goals to the goals
- https://app.diagrams.net/#G1pL-o8b58wxjlz1BakfxWQo-B6EWvYL8W

EXERCISE

GOAL MODEL
Achieve [Prescribed Medication Taken By Patient]
Achieve [Patient ID Verified]
Achieve [Dispensed Medication Taken By Patient]
Achieve [Prescribed Medication Dispensed When Patient ID Verified]

Maintain [Medication in stock]
Avoid [Medicine Overdose]
Achieve [Prescribed Medication Taken By Patient]
Achieve [Patient ID Verified]
Achieve [Prescribed Medication Dispensed when Patient ID Verified]
Achieve [Dispensed Medication Taken By Patient]
Nurse
Patient Wears Identification Bracelet
Achieve [Patient Identification Bracelet Scanned]
Achieve [Prescribed Medication In Patient Drawer]
Achieve [Patient Drawer Opened When Patient Bracelet Scanned
Nurse
Nurse
Electronic Dispensing System

QUALITY OF REQUIREMENTS

"The client kept changing the requirements on a daily basis, so we decided to freeze them until the next release."

EXAMPLE OF NIGHTMARE REQUIREMENTS

- ✓ The system shall perform at the maximum rating at all times except that in emergencies it shall be capable of providing up to 125% rating unless the emergency condition continues for more than 15 minutes in which case the rating shall be reduced to 105% but in the event that only 95% can be achieved then the system shall activate a reduced rating exception and shall maintain the rating within 10% of the stated values for a minimum of 30 minutes.

More than one requirement in a paragraph
Rambling: like a novel
Vague terms: emergencies

ANOTHER EXAMPLE OF NIGHTMARE REQUIREMENTS

- ✓ The system shall provide general word processing facilities which shall be easy to use by untrained staff and shall run on a thin Ethernet Local Area Network wired into the overhead ducting with integrated interface cards housed in each system together with additional memory if that should be necessary.

More than one requirement in a paragraph
Rambling: like a novel
Let-out clauses: if that should be necessary
Vague words: be easy to use, additional memory

CHARACTERISTICS OF GOOD SOFTWARE REQUIREMENTS SPECIFICATION*

- Complete
- Unambiguous
- Consistent
- Correct
- ✓ Verifiable
- ✓ Traceable
- ✓ Ranked for importance and/or stability
- ✓ Modifiable

* IEEE Std 830-1998

COMPLETENESS

- ✓ Definition of the responses of the software to all realizable classes of input data in all realizable classes of situations (IEEE Std).

- ✓ All possible situations must be covered

  "If X then….", "If Y then…." Must also consider what will happen "If neither X nor Y…"

COMPLETENESS (CONT')

- ✓ Automatic door opener

  If the door is closed and a person is detected, then send signal Open_Door.

  If no person is detected after 10 sec., send signal Close_Door.

- ✓ What are missing?

UNAMBIGUOUS

- ✓ A software requirement is unambiguous, if and only if, every requirement stated therein has only one interpretation (IEEE Std).

- Ambiguity: Requirement with terms or statements that can be interpreted in different ways.

UNAMBIGUOUS (CONT')

- ✓ Requirement of autonomous cruise control (ACC) system is:

The maximum speed of a vehicle in a busy road shall be 10 mph.

Ambiguity: What type of vehicle do you mean?
What is busy road?

CONSISTENT

- ✓ A software requirement is internally consistent if, and only if, no subset of individual requirements described in is conflict.

CONSISTENT (CONT')

- Two requirements of the PROFIBUS network are in conflict
  - The PROFIBUS shall have a short reaction time of 60ms
  - PROFIBUS used in hazardous area shall have a low power dissipation of 3.6W

Conflict: Fast PROFIBUS has to have a high power

MAIN TYPES OF CONFLICTS

- ✓ Three main types of conflicts in software requirements
  - Specific characteristics of real-world objects
  - The logical or temporal conflict between two actions
  - Different terms for describing the same real-world object

- Consistency is a challenge since we, at least in the general case, need a complete overview of all requirements that are related to the same event, function, or parameter.

CORRECT

- ✓ Possible incorrect requirements
  - Forward referencing
  - Opacity
  - Noise
  - Etc.

REQUIREMENTS IMPRECISION

- ✓ Problems arise when requirements are not precisely stated.
  - Ambiguous requirements may be interpreted in different ways by developers and users.

- ✓ For the term 'search' in requirement 1
  - User intention – search for a patient name across all appointments in all clinics;
  - Developer interpretation – search for a patient name in an individual clinic. User chooses clinic then search.

FORWARD REFERENCING

- ✓ Requirement items that make use of problem world domain features that are not yet defined.

E, C, and D need to be mapped to a requirement item

FORWARD REFERENCING (CONT')

✓ A requirement of autonomous cruise control (ACC) system is:

ACC system shall maintain the preset speed of an ego-vehicle if there is no forward vehicle.

- Missing inference
  - Who sets the preset speed?
  - What is the value of the preset speed?

OPACITY

- ✓ Requirement items for which rational or dependencies are invisible.

Multiple unrelated concept mapping. A is not related to B

OPACITY (CONT')

✓ **Requirement:**

Each time the freight train (Freight trains are trains used for carrying goods) doors are closed, the passengers must all seated.

- Opacity
  - There is no visible relationship between the freight trains and passengers.

NOISE

- ✓ Requirement items that yield no information on problem world features.

X refers to a concept undefined in the dom

NTNU

NOISE (CONT')

✓ Requirement: The train system shall guarantee safe transportation of all passengers on their residence.

- Noise
  - A residence is an unknown concept within the train domain
  - The train can only transport passengers to the train station and not to their residence

CHARACTERISTICS OF GOOD SOFTWARE REQUIREMENTS SPECIFICATION*

- ✓ Complete
- ✓ Unambiguous
- ✓ Consistent
- ✓ Correct
- ➢ Verifiable
- ✓ Traceable
- ✓ Ranked for importance and/or stability
- ✓ Modifiable

* IEEE Std 830-1998

TESTABLE REQUIREMENT

- ✓ If a method cannot be devised to determine whether the software meets a particular requirement, then that requirement is not testable.

TESTABILITY CONCERNS

- ✓ Testability touches upon two areas of concern
  - How test-friendly is the requirement?
  - How easy is it to test the implementation?

- ✓ These two concerns are not independent and need to be considered together

THREE WAYS OF CHECKING GOAL ACHIEVEMENT

- ✓ **Executing a test**
  - Give input
  - Observe and check the output
  - Involves the computer system and peripherals
- ✓ **Run experiments**
  - Input / output but also involves the users
- ✓ **Inspect the code and other artifacts**
  - Evaluation based on documents

WHEN TO USE WHAT

T: Run test on code
E: Experiment
I: Inspect

NTNU

TO MAKE REQUIREMENTS TESTABLE

- ✓ A requirement needs to be stated in a precise way
- ✓ For some requirements, this is in place right from the start

When the ACC system is turned on, the "Active" light on the dashboard shall be turned on.

- ✓ For some other requirements, we need to change it to get a testable version

The system shall be easy to use.

REQUIREMENT TESTABILITY CHECKLIST

- ✓ Modifying phrases
- ✓ Vague words
- ✓ Pronouns with no reference
- ✓ Passive voice
- ✓ Negative requirements
- ✓ Assumptions and comparisons
- ✓ Indefinite pronouns
- ✓ Boundary values
- ✓ …

MODIFYING PHRASES

- ✓ The meaning of the following words and phrases are subject to interpretation and make requirement optional:
  - as appropriate
  - if practical
  - as required
  - to the extent necessary / practical

- ✓ “Shall be considered" only requires the contractor to think about it
- ✓ “And/or” mandatory/optional?
- ✓ “Shall/should/may”, should not be used for priority

VAGUE WORDS

- ✓ Vague words inject confusion, e.g.,
  - Manage, handle, process, necessary, appropriate
  - Usually, approximately, systematically, quickly
- ✓ Use words that express what the system must do
- ✓ ***Requirement***: The system shall process ABC data to the extent necessary to store it in an appropriate form for future access.
- ✓ ***Correction***: The system shall edit ABC data.

PRONOUNS WITH NO REFERENCE

- ***Requirement***: It shall be displayed.
- When this occurs, the writer is usually relying on a nearby requirement in the requirements document for the meaning of "it."
- As requirements are assigned for implementation, they are often reordered and regrouped, and the defining requirement is no longer nearby.

PASSIVE VOICE

- ✓ Requirements should be written in an active voice, which clearly shows X does or provides Y
- ✓ **Requirement with passive voice:** Z shall be calculated
- ✓ **Correction by using active voice:** The cruise control system shall calculate Z.

NEGATIVE REQUIREMENTS

- ✓ *Everything* outside the system is what the system does not do
- ✓ Testing would have to continue forever to prove that the system does not do something
- ✓ State what the system does

Negative requirement: Dangerous characters should not be used (blacklist).

Correction: Only characters A, B, C are allowed to be used (whitelist).

ASSUMPTIONS AND COMPARISONS

- ✓ **Requirement**: the system shall increase throughput by 15%.
- ✓ Sounds testable, but isn't
- ✓ The assumption is "over current system throughput"
- ✓ By comparing to another system, the meaning of the requirement changes when the other system changes

INDEFINITE PRONOUNS

- ✓ Indefinite pronouns are “stand-in” for unnamed people or things
- ✓ Their meaning is subject to interpretation

- All
- Another
- Any
- Anybody
- Anything
- Each
- Either
- Every
- Everybody
- Everyone
- Everything
- Few
- Many
- Most
- Much

BOUNDARY VALUES

- ✓ Vacation requests of up to 5 days do not require approval. Vacation requests of 5 to 10 days require supervisor approval. Vacation requests of 10 days or longer require management approval.

HOW EASY IS IT TO TEST THE IMPLEMENTATION?

- ✓ Some requirements are more difficult to test than others
- ✓ The volume of tests needed, e.g.,
  - Response time or storage capacity
- ✓ Type of event to be tested, e.g.,
  - Error handling or safety mechanisms
- ✓ The required state of the system before testing, e.g.,
  - A rare failure state or a specific transaction history

HOW EASY IS IT TO TEST THE IMPLEMENTATION? (CONT')

- ✓ Four concerns related to implementation and testability
  - Autonomy of the system under test
  - Observability of the testing progress
  - Re-test efficiency
  - Test restartability

AUTONOMY OF THE SYSTEM UNDER TEST

- ✓ How many other systems are needed to test this requirement?
- ✓ Requirement: “If the door is closed and a person is detected then send signal Open_Door”
- ✓ Can be tested using a simulated actuator
  - Simulate a “person detected” signal on the sensor and check if an Open_Door signal is sent to the actuator
- ✓ More difficult/costly to be tested if the actual sensors and actuators must be installed for running tests.

OBSERVABILITY

- ✓ How easy is it to observe the progress of the test execution?

  This is important for tests that do not produce output – e.g., the requirement is only concerned with an internal state change or update of a database.

- ✓ How easy to observe results of the test?

  Important for tests where the output is dependent on an internal state or database content.

RE-TEST EFFICIENCY

- ✓ How easy is it to perform the “test – check – change – re-test” cycle?

- ✓ This includes
  - Observe the test result and run only failed tests
  - Run only tests that are related to the change

TEST RESTARTABILITY

- ✓ How easy is it to
  - Stop the test temporarily
  - Study current state and output
  - Start the test again from the point where it was stopped

INVOLVE TESTERS IN REQUIREMENT PHASE

- ✓ That a requirement is testable does not necessarily mean that it is ***easy*** to test

- ✓ In order to have testable requirements it is important that

  The testers are involved right from the start of the project. It is difficult to add testability later.

- ✓ The tests are integrated parts of the requirement

NON-FUNCTIONAL REQUIREMENTS

"The client kept changing the requirements on a daily basis, so we decided to freeze them until the next release."

WHAT ARE NON-FUNCTIONAL REQUIREMENTS (NFR)

- ✓ Many definitions can be found of NFR
- ✓ A NFR is basically an attribute of or a constraint on a system [1]
  - Attributes
    - **-ilities**: understandability, reliability, portability, flexibility, availability, maintainability, scalability, …
    - **-ities**: security, simplicity, clarity, ubiquity, integrity, safety, modularity, …
    - **-ness**: user-friendliness, robustness, timeliness, responsiveness, …
  - Constraint
    - **Physical**
    - **Environmental**
    - **Legal**
    - **Cultural**
    - **…**

ISO/IEC 25010 PRODUCT QUALITY MODEL

The ISO model is a factor – criteria – metrics model. Only the first two levels are shown in the diagram.

MAKE RELIABILITY MEASURABLE

The quality concepts (abstract notions of quality properties)

Measurable Quantities (define some metrics)

Counts taken from design/execution (realization of the metrics)

NTNU

RELIABILITY REQUIREMENTS SPECIFICATION

- **Maturity:**
  MTTF (Mean Time To Failure, the expected time to failure for a non-repairable system) > 500 hrs

- ✓ **Availability:**
  The system shall meet or exceed 99.99% uptime

- ✓ **Recoverability:**
  In case of an error, the time needed to get the system up and running again (MTTR) shall not exceed one hour

RELIABILITY TESTS

- ✓ MTTF calculation
  - Assume you tested 3 identical systems starting from time 0 until all of them failed
  - The first system failed at 10 hours, the second failed at 12 hours and the third failed at 13 hours
  - The MTTF is the average of the three failure times, which is (10+12+13)/3 = 11.6667 hours

- How to verify MTTF > 500 hrs?
  - Use 10 PCs. Test for two weeks (40 hr per week) => TTT (Total Test Time) = 800
  - Failed more than once? Satisfied/Not satisfied

TOTAL TEST TIME IN LAB VS. USAGE TIME

- ✓ We need to distinguish
  - Total test time (TTT) in the lab
  - Usage time (UT) of the system in real life

- ✓ The usage time is related to
  - How often is the system used at the users' site?
  - Number of users each time the system is used

TOTAL TEST TIME IN LAB VS. USAGE TIME (CONT')

A simple example:

- ✓ We have TTT = 800 hrs in the lab.
  - Use 10 PCs. Test for two weeks (40 hrs per week)
- ✓ The system will be used for only 20 hrs per week – e.g., for accounting purposes
- ✓ The 800hrs in the lab. TTT is equivalent to 1600 hrs of usage time
- ✓ Failed twice? Satisfy/Not satisfy MTTF > 500 hrs?

EXERCISE : AVAILABILITY REQUIREMENTS

- ✓ If the system shall meet or exceed the following availability, what is the maximum downtime per year to satisfy such availability requirements?

maximum downtime = (total time per year) x (1-availability)

Availability requirement	Maxium downtime per year?
99%	– 3.65 days/year
99.9%	– 8.76 hours/year
99.999%	– 5 minutes/year
99.9999%	– 31 seconds/ year

PERFORMANCE

Response speed and amount of resources used

- Software products
- Hardware facilities
- Materials (e.g., print paper, diskettes)

Criteria

- **Time behavior:** Response and processing times and throughput rates

- **Resource utilization:** Amount and types of resources used

- **Capability**: Minimum/maximum limits

PERFORMANCE REQUIREMENT EXAMPLES

- ✓ Authorization of an ATM withdrawal request shall take no more than 2.0 seconds.

Response time

- ✓ Webpages shall fully download in an average of 3 seconds or less over 30 megabits/second Interne

Resource utilization

- ✓ The on-line banking systems should support minimum 10.000 credit card transactions proceed per second.

Capability

USABILITY

Help users achieve specified goals with

- Effectiveness
- Efficiency
- Satisfaction

Criteria

- **User interface aesthetics**
- **Easy to operate**
- **Easy to learn**

Usability criteria are subjective

Tests should have a strong component of subjectivism

SETTING REQUIREMENTS – THE MBO METHOD

- ✓ The method used in the example is based on T. Gilb's ideas of MbO – Management by Objectives [5]
- ✓ We start with the requirement – e.g., the system shall be easy to learn
- ✓ We then follow up with "what do you mean by…" until we reach something that is observable and thus testable

REQUIREMENTS – FIRST STEP

- ✓ The first step is to apply the MbO for each criterion

- ✓ Examples
  - Learn its application
    What do we mean by "learn application."
  - Is the software product liked by the user
    What do you mean by "beautiful / pleasing / like"?

REQUIREMENTS – SECOND STEP

- ✓ Learn application: After a two weeks course and after two weeks use of the system, a normal user can solve a set of standardized problems

- ✓ Pleasing/like: Score high on a likeability scale – e.g., 90 % score 7 or higher on a scale from 1 to 10 – after a two weeks course and two weeks of real use

We need to develop a course, a set of standardized problems, and a likeability questionnaire.

USER INVOLVEMENT IN SETTING REQUIREMENTS

When we use MbO or other related techniques for setting requirements, we will in most cases have a situation where:

- ✓ The user will have to participate in the tests
- ✓ There will be a strong link between requirement and test
- ✓ In many cases, ***passing the test by the user will be the requirement***

NFR REQUIREMENTS – TO REMEMBER

- ✓ Some NFR requirements are conflict with each other
- ✓ Trade-off of NFR requirements will impact system design and implementation

CHARACTERISTICS OF GOOD SOFTWARE REQUIREMENTS SPECIFICATION*

- ✓ Complete
- ✓ Unambiguous
- ✓ Consistent
- ✓ Correct
- ✓ Verifiable
- ➢ Ranked for importance and/or stability
- ➢ Traceable
- ➢ Modifiable

* IEEE Std 830-1998

RANKED FOR IMPORTANCE AND/OR STABILITY

- ✓ If, and only if, each requirement in it has an identifier to indicate either the importance or stability of that particular requirement.

- ✓ Rank and distinguish classes of requirements
  - Essential
  - Conditional
  - Optional

REQUIREMENT TRACEBILITY

"The client kept changing the requirements on a daily basis, so we decided to freeze them until the next release."

REQUIREMENTS TRACEABILITY

"Requirements traceability refers to the ability to describe and follow the life of a requirement, in both a forwards and backwards direction, i.e., from its origins, through its development and specification, to its subsequent deployment and use, and through periods of on-going refinement and iteration in any of these phases."

Gotel and Finkelstein

BIDIRECTIONAL TRACEABILITY

PURPOSES OF REQUIREMENT TRACEABILITY

- ✓ Forward traceability
  - Ensure all requirements are implemented
  - Change impact analysis

- ✓ Backward traceability
  - Avoid “gold plating”
  - Change impact analysis
  - Defect impact analysis
  - Root cause analysis of defects

HOW TO IMPLEMENT TRACEABILITY?

- ✓ Traceability matrix

Project Name

Unique No.	Requirement	Source of Requirement	Software Reqs. Spec / Functional Req. Doc.	Design Spec.	Program Module	Test Case(s)	Successful Test Verification	Modification of Requirement	Remarks

EXAMPLE OF TRACEABILITY MATRIX

Requirement Source	Product Requirements	HLD Section #	LLD Section #	Code Unit	UTS Case #	STS Case #	User Manual
Business Rule #1	R00120 Credit Card Types	4.1 Parse Mag Strip	4.1.1 Read Card Type	Read_Card _Type.c Read_Card _Type.h	UT 4.1.032 UT 4.1.033 UT 4.1.038 UT 4.1.043	ST 120.020 ST 120.021 ST 120.022	Section 12
			4.1.2 Verify Card Type	Ver_Card _Type.c Ver_Card _Type.h Ver_Card _Types. dat	UT 4.2.012 UT 4.2.013 UT 4.2.016 UT 4.2.031 UT 4.2.045	ST 120.035 ST 120.036 ST 120.037 ST 120.037	Section 12
Use Case #132 step 6	R00230 Read Gas Flow	7.2.2 Gas Flow Meter Interface	7.2.2 Read Gas Flow Indicator	Read_Gas _Flow.c	UT 7.2.043 UT 7.2.044	ST 230.002 ST 230.003	Section 21.1.2
	R00231 Calculate Gas Price	7.3 Calculate Gas price	7.3 Calculate Gas price	Cal_Gas_ Price.c	UT 7.3.005 UT 7.3.006 UT 7.3.007	ST 231.001 ST 231.002 ST 231.003	Section 21.1.3

HOW TO IMPLEMENT TRACEABILITY? (CONT')

- ✓ Trace tagging

SRS

R00104 - The system shall c
time prior to the actual dispe
requests cancellation.

SDS

SDS Identifier	SRS Tag	Component Name
7.01.032	R00104	Cancel_ Transaction

UTS

Test Case #	SDS Identifier	Test C
23476	7.01.032	Cancel_Befor
23477	7.01.032	Cancel_After_
23478	7.01.032	Cancel_After_

CHARACTERISTICS OF GOOD SOFTWARE REQUIREMENTS SPECIFICATION*

- ✓ Complete
- ✓ Unambiguous
- ✓ Consistent
- ✓ Correct
- ✓ Verifiable
- ✓ Ranked for importance and/or stability
- ✓ Traceable
- ➢ Modifiable

* IEEE Std 830-1998

MODIFIABLE

- ✓ If, and only if, a requirement structure and style are such that any changes to the requirements can be made easily, completely, and consistently while retaining the structure and style.
- ✓ Redundancy can easily lead to low modifiability
- ✓ Boilerplate can help improve modifiability
  - E.g., “will” -> “shall”