**Key stakeholder groups:** End Users, Specialist department, audit department, Client, Financier, Management, Board of Directors, Management. System administrators, service personnel, training personnel, hotline, support System Developer, System Maintenance. Project opponents and supporters. **There are three categories of contact persons as their identity in the project:** Domain experts. Requirements manager. People affected by the system. The quality of this categorizing is naturally decisively influenced by the contact person. Therefore, communication is with them is of curial importance.  
**3. Identify stakeholder’s interests**: **(Objective):** Ensure that the requirements of all stakeholders are considered. **(Action):**We describe the objectives and interests of each stakeholder. Identifying existing problems and vulnerabilities from the stakeholder perspective. Describing the important required system properties from the stakeholders’ point of view. **(Artifact)** Vision Doc. + Stakeholder request – Detailed description of individual interests. **Typical difficulties** Interest holders know what they want, but they can't express it. Interest holders don't know what they want. Interest holders think they know what they want until you give them what they want. Analysts believe they understand user problems better than the users themselves. Everyone thinks everyone else is politically motivated. **4. Collect and study material Learning from previous practices and accessing additional sources of information. (Objective):** Learning from previous practices and accessing additional sources of information. **(Action)** identification and analysis of objects, examples and patterns from the domain. Evaluation of the material with regard to relevance and usability for the current project. **(Artifacts)** A list of all collected materials **5. Create Glossary (Objective)** Creating a uniform, consistent understanding of terms and minimise misunderstandings between developers and users. **(Action)** Creation of technical glossary and definition of all important technical terms. Defining all classes of the class model as a term in the glossary. Defining all association roles as a term in the glossary. Define all other important technical subjects, concepts and states of these subjects in the glossary. Defining all important general and technical process words in the glossary. **(Artifact)** Glossary. **Structure of how to create a glossary.** Example below

|  |  |
| --- | --- |
| **Term** | **Invoice** |
| **Synonyms** |  |
| **Shortcut** | **-** |
| **Definition** | **Each invoice results from a contract. It invoices services rendered or deliveries and is addressed to a customer.** |
| **Delimitation** | **There are individual, monthly, partial and collective invoices. An invoice has an invoice recipient, a date, an invoice number, and invoice items that are used to list the individual services and deliveries to be billed for. Each item contains a description, a number, an individual amount and a total amount (item total). The invoice contains a final total (sum of all items). The sales tax is displayed separately for each item and all totals.** |
| **Constraints** | **-** |
| **Contact person** | **-** |
| **Status** | **Final** |
| **Changes** | **...** |

**Things to consider while creating this**: Use Active instead of passive formulation. Do not use synonyms, homonyms or tautologies. Use verbs instead of nouns that are not technical terms. Use terms only in justified cases in the plural. **Use Case:** A use case is a written story. It describes business processes or procedures when using a planned system at a high level of abstraction. In other words, a use case describes the interaction with a system by means of a coherent workflow. A use case is always initiated by an actor and usually leads to a visible result for the actor. **Scenario** is a specific sequence of actions and interactions between actors and the system. Each concrete path through use case represents scenario. A use case is a collection of related scenarios. **Business use case** describes a process flow on a business level independent of a technical system implementation. Triggered by a business event and typically leads to a result that represents a business value. **System use case** describes the behaviours ofa system (hardware or software) that can be perceived by external actors (users, neighbouring system).

DONE TILL PAGE 46

or send events (as regular state diagrams can) • Transitions can have preconditions and postconditions shown in square brackets []. You draw your protocol state machine as a group of substates within one large frame. You must name the protocol state machine as such; place the keyword protocol in curly brackets {} next to the name. **Sequence diagram**: Purpose: Scenario oriented Shows objects involved in the scenario from left to right Shows sequence of messages in a sequence from top to bottom Allows specification of runtime scenarios in a graphical manner  
Object Constraint Language (OCL)  
**Requirements**: A requirement is a statement about a property to be fulfilled or the performance of a product, a process or the persons involved in the process.  
**Requirements Management**  
Comprehensive, systematic approach to identifying, documenting, organizing and tracking requirements  
**Functional requirements** Functional Requirements describe the domain-oriented functionalities or services that the system is supposed to provide from the user’s point of view. These requirements are specific to the type of system being developed and its intended users. It focuses on what the system basically should be able to do. It captures the specific behaviours or functions of the system. It directly impacts the operations performed by the system. Examples might include user authentication, data processing, reporting, and other specific actions that the system performs. **Non-Functional Requirement:** They describe all other required characteristics of the system that do not directly affect the specific function but are crucial for the system’s overall performance and usability. The key points are that it focuses on how the system performs its functions. Often refers to important system attributes that affect the system. Do not define specific behaviours but rather the qualities or constraints of the system. Examples: User-friendliness: How easy the system is to use. Trustworthiness**:** The system’s reliability and security. Efficiency: Performance metrics such as response time and resource utilization. Maintainability; How easy it is to update and fix the system. **Types of Non-Functional-Requirements: Operational Requirements** (Describe interfaces, data, functionalities and reactions of the system to events) also extraordinary events -> exceptions. Operational requirements are also referred to as a business concept. **Quality requirements:** concern software quality criteria such as reliability, maintainability, efficiency and usability. Should be specified quantitatively if possible (-> Verifiability). **Technical Requirements:** include constraints such as devices to be connected, interfaces to external systems and the use of development tools.  
**Validity and maintenance requirements:** here we prepare tests (e.g.by specifying test cases). Defining the acceptance test. Describing the scope of warranty conditions, maintenance conditions, training,.. **Implementation requirements** concern the process model and the documentation. The resources available (personnel, date, costs). Additional conditions such as legal regulations, (company-internal) guidelines and standards. **Requirements Analysis according to the Unified Process**: A systematic approach to finding, documenting, organizing and controlling changes in the requirements of a system. • So basically, the requirements are recorded iteratively and further developed. • Its NOT considered as A PHASE but a discipline that lasts the whole project (with emphasis on inception and elaboration). • Requirements are developed in workshops in cooperations with customers, application experts, businesspeople and technicians. • A list of documents is proposed. Functional requirements are specified with the help of use cases • The challenge is the requirements are neither obvious nor unchangeable over time • An incorrect level of detail in the formulation of requirements eliminates the flexibility for design decisions •   
**Artifacts of the UP requirements analysis** • **Stakeholder requests** (Describes all stakeholders, identifies their wishes and assesses relevance/risk for the project) • **Vision Doc**. (Describes in general terms the objectives and conditions, the business plan; executive summary) • **Business Case** (Describes the objectives, conditions, and business plan of the project. It includes an executive summary and outlines the rationale, benefits, and overall strategic importance of the project.) • **Glossary** (The main terms of the domain and Data Dictionary) • **Risk List** (Describes risks (business, technical, resource, and schedule related) and ideas for management / mitigation.) • **Supplementary specifications** (Describes non-functional requirements) • **Use-Case Model** (Describes functional requirements) • **Analysis Model** (represents the relevant business issues (use cases, domain model, system process model). **Requirements Analysis Methods: Actual State Analysis: ACTIVITES** • Capturing of the user/user environments (structural and process organization, service regulations, decrees, laws, guidelines and similar more). • Capturing existing data programs • Capturing existing IT equipment • Capturing of time and quantity structure (current processing times, turn around times , waiting times, data quantities) • Capturing of business and technical factors that cannot be influenced • Presenting the threat and the equipment gap • Identification of weak points • identifying the causes of the identified vulnerabilities **SORUCES TO GET THE INFO:** Conversations with the management, end users, IT/System administrators. Observing people at work. Recording the current task completion. Using forms that provide information on data requirements and workflows. Through flyers/brochures we can learn about our competencies that still needs to strengthen or how a company wants to be seen. **Object -Oriented Analysis. 1.** **Develop a system idea: (Objective)** is to find the fundamental objective and system idea. What should be achieved with the system to be developed. **(Action)** Developing the system idea together with the client, product recipient, user and the developer, actively clarifying conflicts of interest and contradictions. We also formulate the system idea briefly and concisely. Consider most important characteristics, features, framework conditions, prerequisites and explicit exclusions of performance. Make sure the clients, product recipients, users and developers know the system and support it without reservation – **(Artifact)** Vision-Doc. **. 2.** **Identify Stakeholders: (Objective)** Ensure that all relevant stakeholders are considered. Find out which groups of people can provide requirements for the system. **(Action)** Identifying stakeholders. Assess the importance of stakeholders based on relevance and risk by having if they “must - should – could” be considered. Identification of concrete projects contact persons (name, functions, contact data). Classifying the contact persons into expert, or those responsible for the requirements and those affected by the system on a more detailed level

**Software Engineering** is the systematic application of scientific and technological knowledge, methods, and experience to the design, implementation, testing and documentation of software. **Objective of software engineering**: ensuring the required product can be put into operation: • on time, • cost-efficient and • (as far as possible) error free and to ensure that it fulfills its purpose. The product must be developed in such a way that • operation, • maintenance and • further development of the product   
**Software crisis** meansthe technical artistic character of software development, the unacceptably high maintenance effort for software and the associated high maintenance costs, it can also mean the overall quality of the software product does not meet the expectations of the users, so that many projects fail. The term "software crisis" is coined in the 2nd half/end of the 1960s, when for the first time the cost of software exceeds the cost of hardware. It can occur in various forms, an appearance that accompanies us all the time, expressing that the expenditure to be made for software production and operation exceeds or will soon exceed the available forces for this. The main causes of the so-called software crisis are: • Software is becoming increasingly complex. • Software is prone to errors. • Software is increasingly needed (independently or as part of a more comprehensive product).   
**FURPS+** **Functionality**: Veracity: The software must produce accurate and correct results. Appropriateness: The software should meet the specific needs of its users. Interoperability: The ability of the software to work with other systems or products without special effort from the user. • **Usability** - Comprehensibility: Users should easily understand how to use the software. Learnability: The software should be easy to learn for new users. Usability: General user-friendliness, ensuring the software is easy and pleasant to use. Documentary: Availability of helpful documentation for users. • **Reliability** Reliability / Trustworthiness: The software should perform consistently and predictably. Access Control: Proper mechanisms to restrict access to authorized users. Operational Safety /Robustness: The software should operate safely under predefined conditions. Fault Tolerance: The ability of the software to continue operating properly in the event of a failure. Recoverability: The software should be able to recover quickly from failures.• **Performance**: Responsiveness: The speed at which the software responds to user inputs. Processing Time: The time the software takes to process inputs and produce outputs. Memory Utilisation: Efficient use of memory resources. • **Supportability**: Further Development and Adaptation: Ease of making future enhancements and adaptations to the software. Portability and Compatibility: The ability of the software to run on various platforms and systems.   
**The “+” in FURPS+** **•** Design constraints - Do things like I/O devices or DBMS constrain how the software must be built? • Implementation requirements: Do the programmers need to adhere to standards? Is the use of TDD required? Is statistically sound testing in the context of Cleanroom required? • Interface requirements - What downstream feeds must be created? What other systems must this one interface with? How frequent are feeds produced? • Physical requirements - What hardware must the system be deployable on? Must we be able to deploy to a machine no larger than 12" square, to be stationed in the Iraqi desert?   
**The Unified Modelling Language (UML)** 4+1 view model divides the architecture of a system into five interrelated views to address different stakeholder concerns and provide a comprehensive understanding of the system: **Logical View:** Focuses on the static structure of the software using UML diagrams like class, object, package, composite structure, and state machine diagrams. It addresses how the software will be developed. **Implementation View:** Concerned with the organization of the software components. It includes component diagrams and helps developers understand the code structure. **Process View:** Deals with the dynamic aspects of the system, showing how processes interact. It includes sequence, communication, activity, timing, and interaction overview diagrams. This view shows how components interact within the logical view. **Deployment View:** Focuses on the physical deployment of the system, detailing how the software is distributed across hardware resources. It uses deployment diagrams to map software onto hardware. **Use Case View**: Central to the model, it describes the functionality of the system from the user's perspective. It includes use case and activity diagrams. Customers can only see this view, which illustrates the use cases and user activities. **HOWEVER,** the UML does not guarantee in any way higher quality or shorter development times, its not a replacement for any programming languages  
**Use Case Diagram**: Purpose: It is used an early stage of a project. It is used to show user interactions with a system. It only summarizes some of the relationships between use cases, actors, and systems. It does not show the order in which steps are preformed to achieve the goals of each use case. It is an effective technique for communicating system behaviour in the user's terms by specifying all externally visible system behaviour. **Extend** use case defines optional behaviour, whereas **include** use case is integral part of the including use case. purpose of these two things are to simplify large use case by splitting it into several use cases, also to extract common parts of the behaviours of two or more use cases. **Activity Diagram:** Purpose: Activity diagrams may be used for various purposes: Analysing and depicting processes, documenting workflows, Showing the algorithms in a graphical way, Modelling use case steps. Modelling behaviour aspects of software – methods, services. **Class diagram**: Purpose: Class diagrams may be used in various scenarios: Static structure design and analysis System responsibilities modelling Software reverse engineering Source code generation and scaffolding. **State Machine diagram**: Purpose: There are two types of state machine diagrams: **Behavioural state** machine: specifies the behaviour of a model element. In this type of State Machine diagram: • it is an event driven • transitions originating from a state are triggered by relevant events specified by transition • different transitions from the same state should not be able to be triggered by the same event (otherwise they are not unique) • given sequence of events implies sequence of states, where "on the way" arbitrary behaviour can be executed • on execution the system is either in a state or in a transition, alternately. **Protocol State Machine Diagram** is a specialization of behavioural state machine It specifies the allowed usage of the behavioural features of a classifier • A few special rules apply for protocol state machines : States can have names but can’t show entry actions, exit actions, internal actions, or do activities • Transitions show operations but not actions

-0,2 cm

What you need to do here is to make sure that you have one example from each type of the diagram from the course. Where you need to mark every symbol meanings for example  
A diagram of a tree

Description automatically generated