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 **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

**Disadvantages**: • Lack of a clean requirements definition • Relatively tight quality management • Lack of alignment of development activities with the overall system • High demands on the qualification, discipline and flexibility of the development team • Requires the development team to work in the same place at the same time due to the high communication ratio • Drafting of contract customer / SW-developer problematic. **SCRUM PROCESS:** Scrum is an agile based software mode. It does not have phases like OpenUP, but its has more like a technical milestone with respect to how a sprint ends with a small release every time a sprint ends. In the agile Scrum world, instead of providing complete, detailed descriptions of how everything is to be done on a project, much of it is left up to the team. As an overview, Scrum is a lightweight process, simple to understand but difficult to master. The procedure is defined with the following few roles in the whole process: Team, Scrum Master, Product Owner. The process is organized in “Sprints”. As a summary the process values on commitment, courage, focus, openness and respect. **Artifacts: Product Backlog**: Consists of User stories which define the requirements, so the product backlog defines all the requirements/features of the project, owned by the PO. **Sprint Backlog**: A small amount of user stories that we want to implement, owned by the team and the Scrum Master. Sprint Planning: Here the product owner with the team talks about the priority of the user stories for the upcoming sprints to be implemented. **Daily Scrum**: The whole team comes up together where each member tells • What i did before • What i am doing • What i will do. **Scrum Review Meeting**: SM hosts. We see the current demos of the project, with the project owner in the team, to make sure that the requirements are met. **Scrum Retrospective Meeting**: SM hosts: Team also attends (Usually the PO is not there, so the team can freely talk about the process and how can they improve themselves, their work/performance). **Rare Case (Sprint Cancellation):** Very rare but in a case if Sprint Goal cannot be met by Team, for example, unsolvable impediments, drastic environmental changes and severe misestimation. **MORE-ARTIFACTS**: **Product Backlog Sprint Goal. Sprint Backlog**: • List of the technical tasks per product backlog item • Owned by the team, status and estimates updated daily • Only Team modifies it (PO must not change the scope) **Blocks List** • Impediments, blocks, pending decisions • Owned by SM, updated daily **Product Burndown** Chart • Visualizes overall progress (estimated remaining efforts) and team velocity • Updated after each sprint. **Sprint Burndown Chart** •Visualizes sprint progress (estimated remaining time) • Helps detect problems in sprint • Updated daily **Definition of Done** (DoD) • List of quality criteria applicable to all requirements • Mutually accepted by Team and Product Owner **Product Increment:** • Potentially shippable product version • DOD-complaint (tested, documented etc.) • Delivered once per sprint **Information Radiators — Additional Issues are defined here — More Details:** • Scrum and other artifacts (e.g. architecture diagrams) • Easily accessible for Stakeholders and other interested parties • Big posters/monitors in public areas or project wiki • Purpose: provide max transparency about project **Advantages** • Lightweights - like xp • Copes well with rapidly changing requirements - like xp • Scrum, being agile, adopts feedback from customers and stakeholders – Like by having a executable version sot that the customer can give good feedbacks • Short sprints enables changes based on feedback a lot more easily • The individual effort of each member is visible during scrum meetings **Disadvantages** • Scrum often leads to scope creep (Means taking the tasks from a sprint to another sprint because of lack of resources – Happens in student projects), due to the lack of a definite end-date • The chances of project failure are high if individuals aren’t very committed or cooperatives • Adopting the scrum framework in large teams is challenging —- That's why you can solve this problem by creating sub teams to make the process more effective. • The framework can be successful only with experienced team members • If any team member leaves in the middle of a project, it can have a huge negative impact on the project • Quality is hard to implement, until the team goes through aggressive testing process. **3. 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.

Words, it can also be known as the phase that defines feasibility. Moreover, we are also roughly editable the time and the cost this project would require. So, the target is to decide whether to execute or cancel the project. **Elaboration: (CHECK PAGE 47- MAYBE SOEMTHINGS NEEDS TO BE ADDED FROM THERE)**This phase defines the core architecture of the project, most of the requirements are attempted to be identified here. Sometimes we also might have to refine the vision that was created before, due to some limitations or change in the requirements. We can also have multiple iterations in this phase to finally have our final version of the core architecture. This phase has more realistic estimations, because the requirements are getting defined. We also analyse through simulation, benchmarking the possible projects risks that could occur. Furthermore, we need to have a deep understanding of the system that must be developed, so we need to know the scope, its functionality, non-functional requirements (quality, reliability, etc) and performance requirements. **Constructions**. In this phase, we develop the core architecture from the elaboration phase. This phase also has a lot of iterations. Once the code is implemented and tested according to the requirements, we integrate it to the main system. During the implementation, parallel work on the work on the components is possible to shorten the project durations. The target of this is to achieve a beta version of the system, which means that the software is mature enough to deliver to the customer. **Transition:** In this phase we put the system into the operation at the end user site, which means the target is to release the product. This phase also allows as experience the following things: • User documentation which includes the user manual • Training of User can also be expected • Support of user (e.g. through a support centre) • Quick reaction to user problems. **The whole Procedure: Procedure:** The slide presents the Rational Unified Process (RUP), a developmental approach structured into four main phases: Inception, Elaboration, Construction, and Transition. The process is iterative, with each phase consisting of multiple iterations that allow for incremental development and refinement. In the Inception phase, the project's basic viability and scope are established. The Elaboration phase focuses on defining the project's architecture and detailed planning. During Construction, the actual product is developed through repeated iterations, each ending with a minor release of a stable executable part of the final product. The Transition phase is the final stage, where the product is fine-tuned through further iterations until the final production release is ready for deployment to the end-users. Throughout this process, milestones serve as checkpoints for significant decision-making and evaluation, ensuring the project aligns with its objectives at every stage. **Coreflows/Workflows/ Discipline (9 core)** **Workflows:** **Business Modelling** • Use cases of business processes (Business Cases) are defined • Target: Ensure a common understanding between system developers and users of the project **Requirements** • What the system must do • Document Functional or Non-Functional requirements + general conditions **Analysis & Design** • How should the system be implemented? • Design model (System architecture) and optionally create an analysis model, to achieve a better understanding for the tasks that the developers must do, which makes the software development process more efficient and productive. • The design should reflect with the Functional or Non-Functional requirements + general conditions **Implementation** • Develop and integrate individual modules (Classes) • Components can also be developed in parallel, in order to shorten the time of the implementation process, those components can also be further used **Test** Interaction of components and check compliance with requirements defined for the system **Deployment** All activities that here are dealt with training and user support, the compilation and installation of the software, if required, we also address beta tests and integration/migration problems when replacing legacy system. **Supporting Workflows to help the Software Development Process (RUP)** - **Project management** The main task of this workflow is to help and assist the team to complete goals, managing risk and overcoming framework conditions in order to develop a product that meets the needs of the client (those who give the money) and the user **Configuration & Change Management**: • Management of artifacts created in the project (documents, source code,...) and their assignment to certain releases. • Usually produced jointly by several persons **Target**: Ensuring consistency, Possibility of recourse to older versions, Notification of affected persons in case of changes in documents, further: Management of change requests **Environment** • Provision (Installing) of the software development environment. e.g. IDE. • Provision of hardware and software • Appropriate support of the project staff. **Artifacts** Stakeholder Request, Vision Doc. , Business Case, Risk List, Glossary, Software Development Plan, Deployment Plan, Software Architecture Document, Implementation Model, Design Model, Analysis Model, Test Plan, Use-Case Model, Supplementary Specification. **Advantages**: • Process adaptable to individual needs • Has a focus on key best practices • THE UML process • 2-dimensional process that can depict overlapping activities and phases. • Covers the entire life cycle • Supports object-oriented development and reuse. **Disadvantages**: •complex • Less suitable for small projects • Commercial process • Requires computer support. **Agile Process Model eXtreme Programming**: There are no phase.in this model but there are iterations that are similar to the concepts of sprints, which are designed to be at the same length. The major idea of extreme programming is to concentrate on implementation activities, it is important to know that in this process change costs grow logarithmically at most over time but still this process can tolerate the changing with the requirements over time. **Procedure**: • The development and the delivery of the very small subsystems/releases (1-2 months). • Planning of the individual releases • Iterative programming • Acceptance tests for the individual releases • Strong involvement of the customer in the process • Continuous improvement of the code **YOU SKIPPED PAGE 64**  
**Practices: Small Releases**: • Release cycle of one to three months • Release consists of several iterations of one to three weeks • Iterations break down into work packages of one to three days **Planning Game**: • Requirements are created by the customer in the form of so-called user stories. • Customer priorities stories • Customer determines which stories should be implemented in the upcoming iteration. • Developers specify the effort they expect for the realization of a particular features **Automatic Testing** • Besides the user stories, test cases represent the only definition of the desired functionality • A program feature for which stories should be implemented in the upcoming iteration, which means that for most of the functionalities/code we have test cases. • Developers specify the effort they expect for the realization. • Every time a programmer makes a change to the system, he is obliged to run all existing tests again. **System Metaphor** Instead of design activities XP provides the use of so-called metaphors construct replaces a software architecture System metaphor should be understandable for both developers and customers. **Simple Design**: • A change in the internal structure of software to make it easier to understand and change without changing its observable behaviour. • Simplest the design of an existing system while retaining the semantics. • Refactorization aims to improve the comprehensibility and modifiability of the code **Documentation through self-explanatory program code**: • Corresponding program structure. • Meaningful naming. • If a program needs an explanatory comment at one point, it should be refactored. • Reaction to the problem that additional documentation becomes obsolete very quickly due to lack of ongoing adaptation and is therefore unusable of misleading **Programming in pairs:** • The first partner takes care of the current coding. • The second partner checks the code for typos and logical errors and develops strategies for further implementations. • Paring is dynamic: New partners are sought for each work packages **Common code property:** Consequence: Each developer pair may make changes anywhere at any time **Continuous Integration:** • Newly developed or modified program parts are to be reintegrated into the current code base after only a few hours. For this purpose, an integration and test system is used, on which all the test cases must be checked again after the changes have been imported. **Continuous Integration:** Newly developed or modified program parts are to be reintegrated into the current code base after only a few hours. • For this purpose. An integration and test system are used, on which all test cases must be checked again after the changes have been imported. **40-hour week:** Because development in Paris places high demands on the concentration and performance of employees. **On-site Customer:** Should represent the view of the customer, especially in connection with the business games. **Coding Standard** Ensure uniform code that can be understood and modified by all the developers. **Advantages**: Lightweight, opposite of bureaucratic. Copes well with rapidly changing requirements

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

system functions. **Evolutionary prototyping** is a development approach where an initial version of the system, which is functional but not complete, is built and continuously refined through successive iterations based on user feedback and requirement changes until it evolves into the fully functional final product. Unlike **incremental software development**, where each increment is a fully functional and potentially shippable version of the product that adds specific functionalities in a step-by-step manner, evolutionary prototyping may involve building a base version of the system that has limited functionality and is then expanded and refined over time to meet the complete set of user needs and features. **Iterative Software Development: Idea** is that it tries to combine the strength of other models: Phase Model: Easy to manage, more flexible than other models. • Guiding principle: Risk mitigation Start on a small scale Keep the spiral as tight as possible Achieve the development goals with minimum costs. **Procedure**: • Dismantling of system development into intermediate products/partial products. • Development of each intermediate / sub-product in 4 recurring steps: **->** Determination of objectives, alternatives and framework conditions **->** Evaluation of alternatives and reduction of risks **->** Realization of the intermediate product/subproduct and inspection **->** Project continuation planning • Cyclical through these steps for all intermediate products/partial products. **4 steps in each cycle:** **1. Planning 2. Risk analysis 3. Engineering 4. Evaluation** In the **3rd step** often coupling with other process models: Waterfall model Evolutionary prototyping. **Advantages**: • Periodic review of the process depending on the risks • Process model is not defined for the entire development • Integration of other process models as special cases • Errors and unsuitable alternatives are eliminated at an early stage • Flexible model • Development can be redirected much more easily if insights requires it • Supports the reuse of software by considering alternatives. **Disadvantages**: • High management costs • Less suitable for small and medium-sized projects • Knowledge of risk identification and management is not yet sufficiently widespread. **Rational Unified Process: Idea:** Its a process that can be adapted and extended. The main causes of software development problems are addressed by 6 best practices. The RUP focuses on these 5 best practices which are, Iterative Software Development, requirements management, Use of component-based software architectures, Visual Software Modelling, Software quality testing, Controlled change management. **NOW EXPLAINATION:**  
**Iterative (evolutionary) Software Development.** As large projects cannot be developed in one step. Increasing understanding of the problem area during the project. The focus of consideration: Reduction of the project risk -> Division into manageable sections, each of which leads to an executable version. **Requirements Management (with Use Cases)** Here the development, organization and documentation of the required functionality and the general conditions are to be observed. The approach is by creating use cases from UML, in order to process the modelling of the business transactions by the system. The use cases serve as a reference point for quality assurance, this practice also gives an check whether all the requirements are covered in the system design. **Use of component-based software architectures:** Objective is to support the use of prefabricated components, that is why it is advised to have an early development of a system architecture. The advantages of this practice are isolated development and testing components is quite possible. Reusability in later projects. Semi-finished products (CORBA/EJB) can be used. **Visual Software Modelling:** The goal is to promote readability and clarity of documents. For this it is essential to create graphical notations for the documents. UML diagrams can be used as they are generally known in the world of software development. **Software quality testing**: Integration of QA activities take place here in all sub activities. Important thing to know is, quality assurance is not a subordinate matter, but an integral part of the process. **Controlled change management:** This practice makes changes to requirements, documents, programs, etc. Traceable and enable access to current and past documents. **Process Structure**: 2-dimensional: 1st dimension: dynamic aspects, cycle, phase, iteration, milestone 2nd dimension: static aspects, components, activities, artefacts **Why** **the Rational Unified Process (RUP) is two-dimensional** because it incorporates time-phased project milestones (Inception, Elaboration, Construction, Transition) and concurrent development activities (such as requirements, design, implementation, testing). This allows for activities to overlap and be revisited as needed throughout the development lifecycle. E.g. • During the analysis implementation activity during the realization of a prototype. • Analysis activity during the implementation phase to clarify open analysis questions that have only now become clear. **Concepts of the cycles in RUP**: In RUP, the development process is iterative, meaning it cycles through four distinct phases—Inception, Elaboration, Construction, and Transition—multiple times, with each cycle yielding a more advanced version of the software. Each complete pass through these phases results in a 'generation' of the software. After the first generation is complete, the process begins a new with Inception for the second generation, cycling through each phase again. These cycles allow the software to evolve with improved features and refinements from one generation to the next. **Milestones in RUP**: Once a phase gets fully completed then we have a milestone as an output but inside of the phase we have iterations, where at every iteration we don’t get milestones as an intermediate result, but the intermediate steps are just creating a prove that communication exists within the team. The goal of each of the iteration is to develop a stable executable subset of the final product. **Management Milestones** : Inception: LCO = Lifecycle Objectives Elaboration: LCA = Life Cycle Architecture Construction: IOC = Initial Operational Capabilities Transition: PR = Product Release.  
**The effort and time distribution across different phases of the Rational Unified Process in a medium-sized project**. Inception involves about 5% effort and 10% time, while elaboration takes 20% effort and 30% time. Construction requires the most resources with 65% effort and 50% time, emphasizing the need for clear requirements. Transition has 10% each for effort and time, possibly for final adjustments. Notes on the graph indicate the importance of precise planning and adjustments for frameworks during transition. **PHASES EXPLINATION: Inception:** This phase allows us to prepare an overview of the project by developing the vision, the business cases, we check if this project is even feasible to develop, in other

**Implementation:** • **Input of phase** is the output of the last phase, so once when we have enough Details from the previous phase, we can finally start the implementation process, which are the software specification • **Activities:** • Conversion of specifications into programs **->** Implementation of components according to specification, Systematic testing of components/modules **Result(s) of the Phase:** Software components, documented , tested. The result of this phase is the different components of the Software, documentation and tests. **Test** : **Input:** Tested and documented components / modules**. Activities:** • Integration tests, System Test and acceptance test Integration tests are used to test the interaction points of the Components System test (tests the system against the specification -> Verification) Acceptance test (tests the system in the production Environment -> Validation. **Results of the phase**: System running in the production environment. **What is verification?** Verifications means if the software is correctly developed, in order words it means the conformity between a software product and its specification, so one can question, is the product being developed correctly. **What is validation?** Validation means if the product is rightly developed which simply means that if the software meets the customers requirements, in order words, it means suitability of a product in relation to its intended use, so one can question, is the right product being developed? **Operation and Maintenance: Input** to this phase is the accepted system which is already on the production environment. **Activities** of this phase are administration, bug fixing, optimizing, adaptation to the other system environments and the changes and enhancements to functionality. **Results** of this the system on the production environment **Advantages:** • Clearly defined tasks per phase. • Easy to understand. • Many developers are familiar with this procedure. • Management seems to be comparatively simple, low overhead. **Disadvantages:** • Taking this into account, the idea that we cannot go back to the previous phases once it's fully completed, which in fact is one of the biggest disadvantages of this model, as realistically it's not possible to know all the requirements at the start of the project. • Another disadvantage is the due to lack of communication with the customer, as in this model we will typically meet the customer at the analysis and definition stage and then later on once we are at the Operation/Maintenance we will have to meet him again naturally to show him the final product but the issue due to a big communication gap can lead to these issues that the customer could not even remember what he really wanted or his demands entirely got changed. • Coding starts very late which leads to system integration related issues. Although this model can still be used, but with different variants of it. **Waterfall Model Variants** Another variant of waterfall is possibly combined with the creation of a throwaway prototype. In this prototype we have a validation check by the end of every phase to make sure that we fulfill the customers requirements. Also, we can go back to each phase depending upon our work and need. At last, this time we will be able to meet the customer not just in the system requirements phase but also in the software requirements and operations and maintenance phase. V-Model is a process of software development methodology that represents an extension or variation of the waterfall model, emphasizing the validation and verification phases. The V-Model is sometimes referred to as the "Validation and Verification Model.". It is subgrouped: • System creation (SE), • Quality Assurance (QA), • Configuration Management (KM), • Project Management (PM), The whole idea is that while the development activity is proceeding, at the same time the Quality assurance activities can work in parallel. **The "Acceptance Test" in the context of the V-Model means the following**: The purpose of this testing phase is to evaluate the system against the original requirements and needs of the users to ensure that it fulfills them appropriately which simply means that it meets the business requirements. **Advantages**: • Very well worked out and comprehensive • Generic process model with defined tailoring options • Enables standardized processing • Integration representation of 1. System Creation 2. Quality Assurance 3. Configuration Management 4. Project Management. **Vulnerabilities**: • Intended for large embedded systems. • For small projects, in some cases disproportionate bureaucratic effort. • Maintenance phase not considered. • Heavyweight process. **Evolutionary Prototyping:** A throwaway prototype, also known as a rapid prototype, is a type of prototyping that involves creating a model of a system or application with the intention of discarding it after it has served its purpose of collecting and analysing user feedback. So the main **Idea** is that, instead of a throwaway prototype, a series of prototypes is developed that converges directly into the production version of the software. Continuous improvement of prototypes. **Procedure**: • Develop a first, functionally very limited implementation • Involve customer to comment prototypes • Developnext prototypes based on customer feedback, etc. **Advantages**: • In case of unclear / unknown requirements or missing data the customer can develop his ideas of the target system step by step based on the prototypes. • The customer has a prototype available very early → important for acceptance by end users • The development risk will be significantly reduced. • Very well suited as a supplement to other process models. **Disadvantages and problem areas**: • Documentation: Prototype is considered as a replacement for missing documentation • Maintenance problems: worsening of the system structure due to constant changes. • Management: Covering up a code and fix procedure? Contract: On what basis is the service provided? • Performance: The limitations and limits of prototypes are often unknown to the customer. **Incremental Software Development.** • Stepwise extension of the functionality of the system by increments. • Development in stages of expansion. • The process is very similar to evolutionary prototyping, but the generated software artifacts have no prototypical character but can be used productively. **Procedure**: • Starting point is a small, manageable core • Further functions will be developed gradually (successively or in parallel) • When another increment is finished, it is added to the core. **Advantages**: • Usable product in short cycles → early use! • Experiences from the usage can be brought into the further development at an early stage. •Incremental development leads to manageable sub-projects. **Disadvantages**: • The originally selected software architecture may not be sufficient and may have to be completely revised at a relatively late point in time. • Only useful for relatively independent

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)  
**2. Systems development life cycle (SDLC) Life-Cycle Phases Initiation:**  The customer/sponsor with the software company identifies the opportunity concepts that the software can bring along. They talk about the possibilities of the successfulness of the creation of this software and also what could be the possible future of it. Typically, stakeholders create a vision document for defining the further processes. **System Concept Development:** Here we cover the scope of the system boundaries where we also document that. Moreover, we analyse the cost benefits and plan the Risk Management Plan which means the risk list which covers the risks that the software development life cycle face along the development process. **Planning:** Develops a project management plan and other planning documents which provides the basis for gaining the resources that are needed to achieve a solution. **Requirements Analysis:** The analyst with the team develops the user requirements to have an attempt to fix the system. In this phase they gather as many user requirements as possible for the development team to understand the tasks. Additionally, this phase achieves functional and nonfunctional requirements related documents. **Design**: In this phase, the architect translates the user requirements into a system design document, that document focuses on the details that the for example the skilled programmer needs to develop the software with sufficient additional input design, like an architecture notebook that can include the class domain diagrams, architectural layouts, database schemas, etc. **Development:** Converts a design into a complete information system, which means installing environments or libraries, database prepare statements, coding, test cases, refining programs, review of the tests and procurements activities. **Integration and Test:** This phase makes sure that the developed system follows the functional requirements that were set up previously. Typically, this phase is conducted by a quality assurance officer. The outcome of this phase creates test analysis reports. **Implementation (Deployment- more accurate to say):** We include the implementation that was successfully tested before to be pushed on the production environment, which means we primarily already have to know which piece of software should be now part of the existing software on the production environment. Furthermore, it's a good practice to document what we implemented, so if new members in the team join, it would be easier for them to understand what the project is about. **Operations & Maintenance**: This phase generally lasts longer than the pre-development phase, because there could be further changes and enhancements that have to be included in the production environment, also this includes post-implementation and in-process reviews. **Disposition**: Describes end-of-system activities, emphasis is given proper preparation of data. So, if an existing system on the Production environment has to be shut down. This phase then is taken into consideration**. Purpose of the Software Development Process:** • Structuring the process of software development in software projects • "Engineering" approach to software development Division into well-defined activities, which are to be processed with predefined methods. Providing a structured set of activities required to develop a software system. **Central goals of a process model:** • Supporting the team members to perform the key activities • Specification of the artifacts that will be developed through out the system development • Unify the tasks of individual developers with those of the entire team, which would make the software development process smoother. As a results it would eliminate any dependencies related issues that are experienced by the developers (For example: Experience using Scrum Daily) • Project managers are able to monitor and evaluate products and the activities that's going on the software development process; **Common points that all of the software processes have in them** • Software Specification: This means that the functions of the software and the restrictions must be defined, in order words its called requirements. • Software design and implementation: This concept brings more details to the software process, as before things were considered at a high level. As a result, this helps the project stay accordingly towards the requirements of the software. • Software validation: Most of the software development process ensures that the software does what the customer requires. • Evolution: The software must evolve in order to keep pace with changing boundary conditions: **MODELS: CODE AND FIX MODEL:**Unsystematic approach This approach is very unsystematic, and it cannot be even considered as a process model, because of lack of design, documentation and plan, which leads to unexpected results. This approach makes poor task performance due to insufficient requirements analysis. **CLASSIC WATERFALL MODEL.** This process consists of different isolated phases. There are typically 5 phases in this early model; Analysis and Definition, Design, Implementation, Integration and Test, Operation/Maintenance  
**Phases of waterfall model: Analysis and Definition:** In this phase we define and analyse the goals of the project. Moving on we also define the product requirements. We identify the weak points**.  
Result of this phase:** We get the specific requirements of the project. This phase can bring the following artifacts as an outcome: • Functional Scope • Specification of user interface • Interfaces to the system environment • Required performance of the system Intended hardware and, if applicable, system software • Documentation guidelines • Effort and schedule planning. **Design**: From the previous phase we take requirements specifications as an input quantity for this phase the activities involved in this phase are the following: • Develop and internal structure of the software system. **a)** Functional design —> Application model. **b)** Technical design: **i)** Development of the software architecture - System Design. **ii)** Dividing into components/modules (interface specifications). **iii)** Specification of components/modules and interaction. • **Result of this phase**: brings a clear picture of how the software must be developed, so we are now able to get detailed specifications of the software, this is also an artifact

**1. 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