1.

Software is a technology that affects all domains and businesses. **Software Engineering requires** a plan, project management, Quality management, development steps, verification steps, tools to support the development.

2. Process Models

Process models determines the order of the workflow, respective activities, completion criteria, responsibilities and competencies, guidelines, methods to be applied. **V-Model** is phase model with explicit quality assurance phase for each development phase which emphasizes quality assurance, verification and validation of partial products is a part of v-model. **Verification**: verifying the correspondence between a software product and its specification ex. Is product being developed correctly? **Validation**: suitable respective value of a product with regard to its purpose of use. Ex. Is the correct product being developed? **Problems of traditional process models:** customer is often unable to formulate the requirements on new system explicitly, Terminates the cooperation between developers and users when requirements have been completed, Development department withdraw from the customer after definition phase, there are multiple solutions for one problem. **Prototype model:** Supports early development of executable models of future product in order to demonstrate the implementation of requirements and design in software experiment. **Demonstration prototype:** give the potential customer a first idea, intended application are might look like. **Prototype in the true sense of word:** Visualize aspects of the interface of parts of the functionality. **Lab Sample:** Answer construction related questions and alternatives, technical, no intend for end users. **Piot system:** designed for use in the usage environment and not only under lab condition. **Horizontal**: Only specific levels **Vertical**: Selected part of system completely across all level. **PT model Advantage:** Risk reduced, improve planning, promote creativity, Integrated into other process. **Evolutionary Model:** starts with core requirements, core system delivered first, develop gradually and incrementally. <u>**Pros**</u>: gets operation short time, combinational with prototype, experience use in project, created small size so manageable <u>**Cons**</u>: entire systems architecture needs to be reworked in subsequent version, zero version in not flexible enough to adapt. **Concurrent Model:** originates in manufacturing industry, all dev departments are united in one team, different team works on different task of a product in parallel. <u>Pros:</u> early problem detection, optimal use of time <u>**Cons**:</u> Critical decision will be late, High planning and effort required to avoid error early on. **Principals of agile alliance 2001:** Individuals and interaction over processes and tools, working software over comprehensive documentations, customer collaboration over contract negotiations, responding to change over following a plan.

3. Planning and Requirements Engineering

Task at the beginning of software development: Customer requirements specifications, effort estimation, feasibility study, project plan, functional specification document. **Requirements:** A property or condition-typically defines by the customer-to solve a problem – Process (organization, Management, Produce), Product (*Functional-description-functionality-prospective *Nonfunctional-quality-constrains) **Requirement Must be:** Understandable, Unambiguous, Consistent, Minimal, verifiable and implementable.

4. Customer Requirements Specification

Outline of a customer requirements specification(CRS): Goal determination, Product User, P Overview, P furcation, P data, P service, Quality Requirements, Supplements. **Glossary**: Complements the CRS, defines and explains terms to ensure uniform technology (common term to understand)

Change History: Version-Date of Issue-Name-Changed Section-Reason, *Product Functions-Product Date-Product service*

5. Effort Estimation: Function Points

Principle of the Function Point Method 1. The development effort for a product is influenced 1.1 by the product, (its scope, difficulty, quality requirements) 1.2 by the company's development process, (by its process maturity, staff competence, technology used) 2. The Function Point method takes these aspects into consideration

Categories of Function Points: Every requirement are assigned to categories (input data-queries-outputs-data set-ref data), Every requirement is assigned to one of the classes, entry into calculation form, assessment of the impact factors, calculation of the assessed function points, reading the effort in person months, updating of empirical data.

The Function Method (*previous effort estimation method*) #<u>***Prerequisites***</u>: only used after product requirements are known, focus is supposed to be on the entire product, product viewed from the perspective of the customer, performed by staff members with sufficient knowledge. #<u>***Advantages***</u>: adaptable to (different areas, new techniques, company specific conditions), estimate possible at early time, easy to learn, little time effort, accuracy, tool support #<u>***Disadvantages***</u>: only overall effort can be estimated, too strongly related to function, Quality requirements are not taken into account, methodological deficiencies. **Goal of a document**: Predict overall effort until launch accurately based on function point method. **Impact of product and quality requirements:** Interleaving, Decentralized data, Transaction rate, Difficult computing operation, Control process, Exemption, Complex logic, Reusability, Data Conversion, Configurability.

6. Functional Specification Document

Functional Specific Documents(FSD): Contains summary of all technical requirements that the software product to be developed must fulfill, in addition development priorities from the customer's perspective are defined. **Outline of a FSD:** Goal definition (must have, nice to have, delimitation criteria), Product Use, P overview, P functions, P data, P service, Quality requirements, User interface, Non-functional requirements, Technical product environment.

7. Project Planning

Types of date for Task/Milestone: Planned dates, Actual dates, Late dates, Planned and late dates. **Float**: difference between earliest and latest start date of a task Free **Float:** amount of time a task may be delayed without delaying another task. **Total float:** may be delayed without an impact on the end date of the project. **Critical task:** task without float, Critical **Path:** sequence of several critical task. **Date calculation:** forward calculation (start date+ duration = earliest end), Backward calculation (End date – duration = latest start). **Asset Planning:** Asset, Resources, Asset planning, On-schedule asset planning, Capacity oriented asset planning.

8. UML: Analysis

OOP: data are organized into object with their functionality. Every object is able to receive message and send message and process data within own methods. Pros: Reuse due to centralized management of data, Compatibility with concurrence and parallelism. **UML**: modeling language on context of software development, based on object oriented concepts, very extensive. <u>Concepts</u>: <u>*classes*</u>: define a type of object with specific data and functionality, <u>*Objects*</u>: instance of a class. **Application of UML:** Visualization, Specification, Creation, Documentation. **UML Relationships**: Dependencies, Associations, generalizations, Realizations. **Elements of OOP:** <u>*Class*</u> -> Abstraction of a type of entities, usually stateless, blueprint of multiple objects, cab be used like object. <u>*Object*</u> -> behave like a concrete entity, has a state, contains methods and modify its data, attributes can be read and written by objects methods. <u>*Attribute*</u> -> Describes properties of object. *Object Attributes, Class attributes.* <u>*Methods*</u> -> realize the functionality of object. *Object methods, Class methods. Message* -> Used to activate methods of objects. **Features of OO Development:** <u>*Inheritance*</u> -> new classes can be derived from existing classes, relationship between a superclass and a subclass is strictly hierarchical. <u>*Multiple-Inheritance*</u> -> Derived classes can have more than one superclass. *Abstract class*-> Used for structuring a class tree, cannot be instantiated. <u>*Binding*</u> -> Mapping between mechanism and implementation. Static Binding (mapping done at compile time, not flexible), Dynamic binding (mapping between invoking message and executed code done in runtime, flexible handle methods). **Role**: Describes which function an object/class has within an association. **Association**: Models the relationship between object or class. Possible to connect the same class or object itself.

Aggregations: is a kind of association. It implies "is required by" semantic between a related classes/objects. Aggregation is directed but not necessary an acyclic, graph. Ex. Car requires tires.	**Compositions**: is an aggregation with the semantics of strong ownership. A composition implies "consist of" semantic between the related classes/objects. it is a directed acyclic graph. Ex. Building with rooms.

Subclass: has (attributes + methods + associations) of the superclass. **Message**: A method calls from one class to another. The calling class call called and executed class is called callee. Server interprets the message and executes an operation. A message triggers an operation with matching signature.

9. UML: Design

Container class: Manages a set of object of given class. (array, sets, list) **Visibility of operations**: Private operation, Protected Operation, Public operation, Package visible operation. **Abstraction Operations:** Only consist of method signature, no implementation. **Navigation:** Association direction is called "navigation direction of association". **References:** association direction between objects can be realized using references. **Method Overriding:** When subclass implements an operation of superclass and uses the same signature this is called overriding. Same name, number of argument, argument type. Pros: implementation of an operation can be changed in a subclass to modify the class behavior. **Method Overloading:** One or more additional operations are created with same name but differ parameter.

10. Cohesion and Coupling

Cohesion: is a qualitative measure of a system component. It contains relationships between the elements within a system component. A good architecture has strong cohesion

Coupling: is a qualitative measure for the interface between two system components, taking into account the coupling mechanism, the interface width and the type of communication between components. Coupling should be as weak as possible.

Cohesion types: **Sequential**: when several sub-functions that are executed in a succession are combined into one system. **Communicational**: sub-functions are combined from a system component on the basis of communication relationship. **Procedural**: when sub-functions are combined from a system component on basis on common algorithm. **Temporal**: system components are created on the basis of common execution time. **Logical**: exists when sub-functions are combined from a system component due to their similarity. **Coincidental**: is not based on any recognized principal. **Function**: all elements contribute to achieving a single specific goal. Can describe with one verb and one noun. **Informal**: is happen when the purpose of data abstraction is to prove a single, defined service. **Advantage of functional cohesion:** Low error rate, high degree of reusability, easy to achieve extensibility and maintainability. **What weak coupling means?** high degree of independence of the system component, enhance maintainability, quality assurance activities, exchange of components.

Data structure coupling: exist when structured data is transferred. Data structure must be known. Doesn't apply to predefined data type, ex string. Components lose their independence, error rate high, maintenance are difficult.	**Hybrid Coupling:** occurred as a result of bad cohesion because several function from which a selection must be made with the help of control information have been combined in one component. *Can be eliminated by separating the service providing component.*
External coupling: When communicate with each other using global data. *External coupling can be eliminated by abstracting class.*	**Content coupling:** exist if details of another component are necessary for the correct function of the component. *Content coupling generally requires extensive changes.*

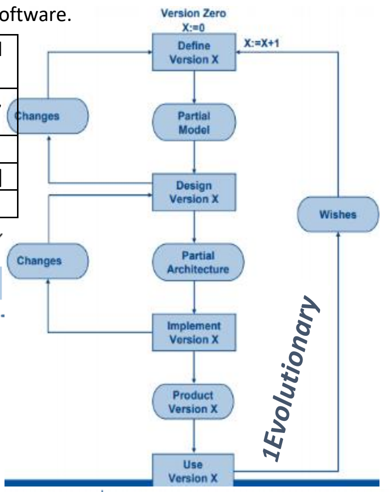
12. Unit Test

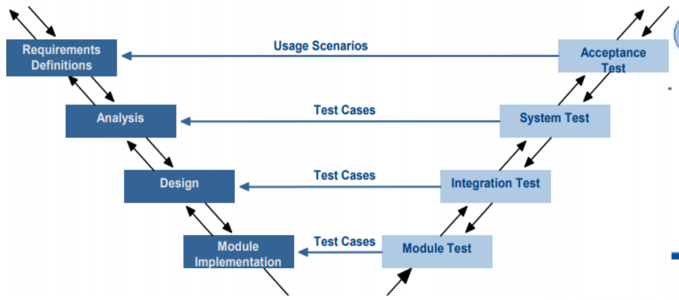
Unit Test: testing operation, single classes or cluster of classes. **Integration test**: Testing the interaction of the base classes and testing inheritance. **System test:** testing complete use cases. **Equivalence partitioning**: Identification of equivalence classes based on specification. All values from equivalence classes shall cause identical behavior. Equivalence classes are also generated from the outputs. Boundary value analysis is used to test equivalence classes. **Branch Coverage**: aims to executing all branch of the program to be tested. This requires the execution of all edges of the control flow graph. It ensures the testing of all statements.

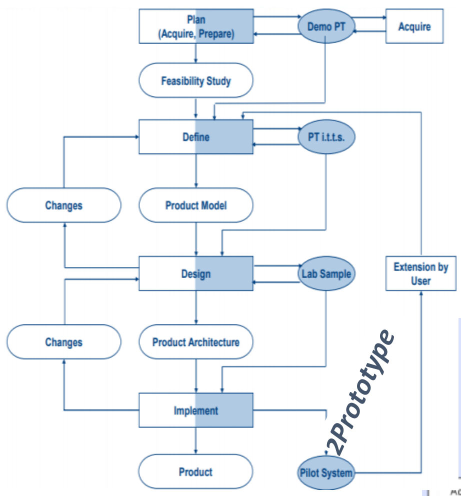
14. System Test

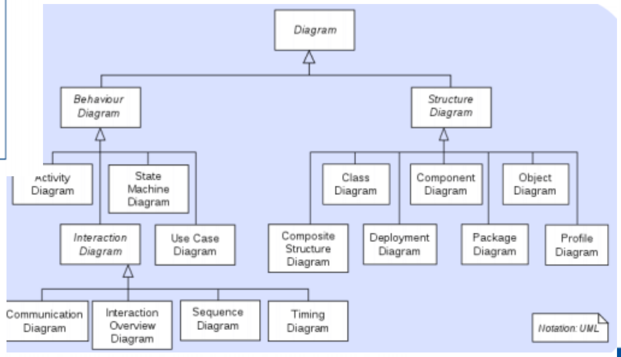
System Test: Testing complete use cases, testing performance, testing stress, also test from customer side. Functional Test: check all defines functions provided ob requirements document. Can be created from OO analysis diagram. Techniques used/, data flow diagram, state machine, sequence charts, use cases. **Integration test**: testing the interaction of base classes and testing inheritance. The purpose of this test is to expose faults in the interaction between integrated units. **Regression Test**: is rerunning function and non-function test to ensure the previously developed and tested software still performs after the change. **Performance Test**: are used to verify the system performance with respect to the requirements definitions. **Stress Test**: that determines the robustness of the software by testing beyond the limits of normal operation. **Beta test**: consist of installing the software specially selected pilot customer system with the aim of detecting aim of eliminating any remaining bugs before launching software.

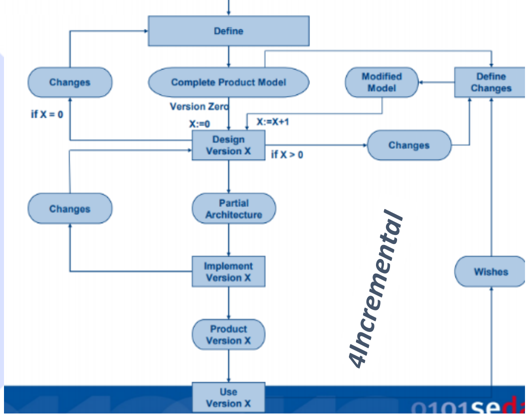
High Quality + clear system requirements = V-Model, Usability & customer Satisfaction +Involve users = Prototype + incremental model, clear requirements + alpha version test = incremental model, two task parallel + clear requirements = concurrent model
[Class (upper class, sub class), Responsibility, Collaborator] [definition, attributes], Quality Criteria: Understandable, unambiguous, Consistent, Minimal, Verifiable and implementable ,Customer Specification, Requirement Specification, Product Functionality, Product Service
Final assessed FP: unassessed FP*(0.7+0.01*impact factors assesment) MPM: SS, SF-Task may finish, when preceding task start, FS, FF
[Extends, Belongs to, Composed of] [+public, -private, #protected, ~package], [person – singer – stageNameContainer - stageName]
Statement coverage: every statement (if, while, else) Branch Coverage: every branch, [valid equivalence class, Invalid]

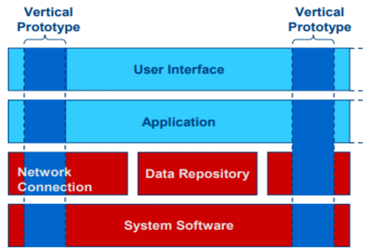


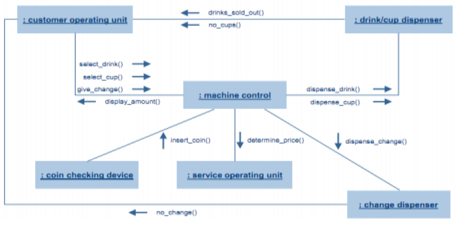


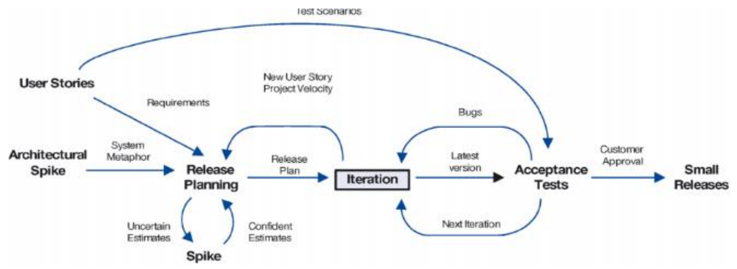












Answer the following questions about the specifications: Which of these documents? 1.CS contains the requirements from the customer's point of view? 2.RS serves as the basis for the acceptance test? 3. CS RS contains the quality requirements for the product?4. ... precedes the requirements analysis? 5.RS CS serves as contract basis. 6. ... contains the first draft of the solution to be implemented? 7. RS summarizes the development priorities from the client's point of view? 8. CS serves as a basis for examining the feasibility of the project? 9. CS RS defines a vocabulary shared by the client and the contractor? 10. ... should contain unclearly formulated specifications? 11. CS is typically included in a tender? 12. ... determines the financial scope of the project?

Does the modeling of a class diagram depend on the programming language used? Yes, because all features are not available in every programming language,e.g multiple inheritance is available in C++,but not in java. **When can multiple inheritance cause problems?** If there are accessible attributes or methods with the same signature in the parent classes. It is then unclear which attribute or method is to be accessed. **When can child classes access the attributes of the parent classes?** If the attributes of the parent class are in the visibility level public or protected. One of your colleagues uses weak cohesion and high coupling between the components in his design. What are the consequences of this structure: higher communication load between the components, changes are often cross component, structure becomes increasingly chaotic, the separation of responsibilities is becoming increasingly blurried. **Which type of cohesion has the new utilities component? And why?** coincidental cohesion because the elements within utilities component have nothing on common between them, no criteria to join them. How do you assess the designer solution and what are the consequences of this solution? This is the worst type of cohesion which leads to high communication and maintenance effort.