By Abdullah Zubair

SE Notes

Software Application Types:-

- Stand Alone Applications: do not need connection
- Interactive transaction based : remote computer based actions, web based applications
- Embedded control systems: software that controls and manages hardware
- Batch processing systems: process data in large batches, credit card payments, bills, etc
- Entertainment systems: media players
- Systems for modelling and simulation: scientists and developers to model physical processes and simulations
- Data collection systems: collect data from environment using sensors and send data to other processing systems
- Systems of systems: systems composed of other systems, electric/powergrid systems

Software Process: A set of activities whose goal is the development or evolution of a software
Generic activities:

- Specification: what a system should do and its developmental constraints
- Design and development: production of software system
- Validation: checking that the software is what the customer wants
- Evolution: changing the software in response to changing demands

- Good processes lead to good software, reduce risk, enhance visibility and enable teamwork
- Process Perspective:
    - Workflow perspective: sequence of activities
    - Data-flow perspectives: information flow
    - role/action perspective: who does what
- Software Lifecycle Activities:
    - Basic Process Steps:
        - Feasibility and planning
        - Requirements
        - System and program design
        - Implementation
        - Acceptance and release
        - Operation and maintenance
    - Quality Control Steps:
        - Validating requirements
        - Validating system and program design
        - Usability testing
        - Program testing
        - Acceptance testing
        - Bug fixing and maintenance

- Process Steps:
    - Feasibility: scope, technical feasibility, benefits, costs, resources, and risks of project
    - Requirements: client viewpoint establishes functionality, constraints and goals. Sometimes split into: analysis, definition and specification
    - Design:
        - System design: establish system architecture both hardware and software
        - Program design: represents software functions. Preliminary testing is done at this step. Models like UML are used to describe requirements
    - Implementation: Design of code and software code level testing. Individual components are tested against the design
    - Testing:
        - User testing: tested by users with a frontend interface
        - Program testing: development teams test components individually (unit testing) or in combination (system testing)
        - Acceptance testing: client tests the final product against the design
    - Acceptance and release: acceptance testing by client and after its success delivered to client
    - Operation and maintenance:
        - Operation: system put into use
        - Maintenance: errors and problems identified and fixed
        - Evolution: system evolves over time as per requirement changes
        - Phase out: the system is withdrawn from service
- Sequence of Process Steps:
    - Sequential: Complete each step before beginning the next
    - Iterative: go through all steps quickly and keep improving the whole system in stages till end product
    - Incremental: Software is made in small increments of tasks
- HeavyWeight Software Development: development team works through process slowly and systematically, with the aim of fully completing each process step
- Lightweight Software Development: work in small increments. Each increment includes all process steps
- Deliverables:
    - Heavyweight: each process step has a deliverable
    - Lightweight: deliverables are incremental working code with little docs

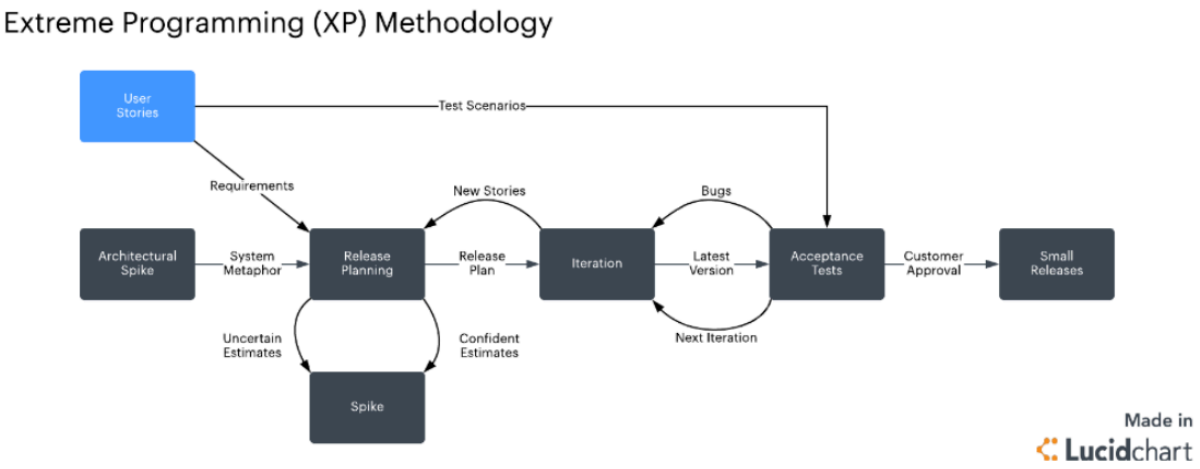
- Incremental:
    - You have all requirements
    - Design the complete product first, leave out details that you can safely decide later
    - Slice design into chunks and do separately
    - Completion of one slice calls for integration with previous

- Iterative:
    - Discovering what you need to do as you go
    - Start with a basic fair idea
    - Design, build and test small version and take feedback
    - Work on feedback and add features and keep expanding
- Traditional Models:
    - Waterfall (Sequential-HeavyWeight):
        - Flow: Feasibility Study, Requirements, System Design, Program Design, Implementation, Program Testing, acceptance and release, operation and maintenance
        - Advantages:
            - Process Visibility
            - Separation of tasks
            - Quality control at each step
            - Cost monitoring at each step
        - Disadvantages: Each stage reveals new knowledge of previous steps that need to be revised. Backtracking is difficult to do. To overcome this we add a form of iteration in modified waterfall
            - High amount of risk and uncertainty
            - No working software made until late in process
            - Not good model for complex projects
            - Very hard to backtrack
        - Modified Waterfall: all steps have a feedback loop to go back
        - When to use:
            - Product definition is stable
            - Technology is understood
            - Requirements are very well defined
            - No ambiguous requirement
            - Ample resources and expertise
            - Project is short
    - Prototype Method(iterative):
        - Used in: requirement engineering to help elicitation and validation, in design process to explore ui and in testing phases
        - Flow: Loop (Communication -> Quick Plan -> Modeling Quick Plan -> Construction of Prototype -> Deployment Delivery and Feedback)
            - Bottom up approach
            - Requirements and domain not well defined
            - Create small implementation
            - Requirements are explored before created
        - Process Model: Establish Prototype Objectives (prototype plan) -> define prototype functionality (outline definition) -> develop prototype (executable prototype) -> evaluate prototype (evaluation report)
        - Focus on functional requirements
        - Error checking and recovery may not be in prototype

        - Throw away prototypes: impossible top tune system to meet NFR, prototypes are undocumented, prototype structure is usually degraded due to rapid change, will not meet organizational standards
        - Advantages:
            - Client involved in early feedback
            - Improves requirements and specifications
            - No scope of refinement, new requirements can be easily added
            - Ideal for an online system
            - Developers understand system better
        - Disadvantages:
            - Model is costly
            - Poor documentation
            - Can have too many variations in requirements
            - Customer asks for product soon after seeing prototype
            - Customer may not be satisfied after seeing initial prototype
        - When to use:
            - When systems have a lot of interaction with end users
            - Online systems, web interfaces :
            - Users always work with system development using a consistent feedback loop
- Spiral Model (sequential + iterative):
    - Combines waterfall and prototyping
    - Has iterations called spirals
    - Activity centred-looped system with customer feedback in the end:
        - Planning: specific objectives for phase are identified
        - Risk Analysis: activities put into place to reduce risks
        - Engineering:
        - Evaluation
        - Advantages:
            - Allows the usage of prototyping
            - Client can see system early on
            - Changes in requirement can be seen early
        - Disadvantages:
            - Management is more complex
            - Should not be used for small projects
            - Lots of documentation needed
        - When to use:
            - When costs and risk evaluation is important
            - Medium to high risk projects
            - Users are unsure of their needs
            - Requirements are complex
            - New product line
- Rapid Application Development (RAD):
    - Phases:

            - Business Modelling/Business Processes: what information is generated by who and where it goes to and who processes it
            - Data modelling: information flow refined in data objects, relationship between objects are defined
            - Process Modelling: data objects are processed and converted into required information to implement a business function
            - Application Generation: create and use reusable components to speed up and facilitate developmental process
            - Testing and turnover: new components must be tested
        - Drawbacks:
            - For large projects human resource needed more to make the right RAD teams
            - Requires developers to complete a system in a short amount of time
- Agile Models:
    - Traditional Models take too long
    - Agile systems develop systems more quickly with less time used on design and analysis
    - Requires real time close evaluation between individuals
    - What is it:
        - Focus on code rather than design
        - Based on interactive approach
        - Intended to deliver quickly
        - Evolves quickly to changed requirements
        - Current processes are: too heavyweight, too rigid and more active customer engagement is needed
        - Lightweight
        - People based
    - Agile principles:
        - Customer involvement: customer should be closely involved in the developmental cycle
        - Incremental delivery: developed in increments with customer explain requirements for each increment
        - People not process: skills of a team should be recognised and used accordingly
        - Embrace change: expect system requirements to change
        - Maintain simplicity
    - Agile Teams:
        - Hire good people
        - Seat them close together
        - Get them close to customers and users
        - Arrange for rapid feedback and changes
    -
    - Extreme programming:

- Principles:
    - Incremental Planning: requirements are recorded on story cards. Break these stories into tasks
    - Small releases: show minimal useful set of functionality
    - Simple design
    - Test first development: unit testing required
    - Refactoring: all developers are supposed to expect the refactor of their code
    - Pair programming: developers work in pairs
    - Collective ownership: all developers take responsibility for all code
    - Continuous integration: as soon as task done it is integrated into the system
    - Sustainable pace: large amounts of overtime are BAD
    - On-site customer: someone to immediately interact with and answer questions



    - Advantages:
        - Built in quality
        - Overall simplicity
        - Programmer power
        - Customer power
        - Synergy between practices
    - Disadvantages:
        - Informal and little to no documentation
        - Scalability
        - Misconception on the cost of change
    - When to use:
        - Highly uncertain env
        - Internal projects
        - Joint ventures
    - When not to use:

        - Large complex env
        - Safety critical situation
        - Well understood requirements
        - Distant or unavailable customer

- SCRUM:
    - Focus on managing iterative development rather than specific agile practices
    - Three phases of scrum:
        - Outline planning: establish the general objectives of the project and design software architecture
        - Series of sprint cycles: each sprint develops and increment of the system
        - Closure phase: wraps up project, completes documentation, and assess lessons learned
    - Functionality of SCRUM: Product Backlog -> sprint backlog -> 2-4 week development cycle / sprint with daily standups -> potentially shippable product increment
    - Components:
        - Roles: Product Owner, Scrum Master, Team
        - The Process: Sprint Planning, Sprint Review, Sprint Retrospective, daily scrum meeting
        - Artifacts: Product Backlog, sprint backlog, burndown chart
            - Sprint Review: Provide a working Demo, what did we get done, what value did we deliver?
            - Sprint retrospective: what went well, what could we have done better, conduct blameless postmortem of any incident. Stop doing _, continue doing_ , start doing_
            - 15m standup: what have i done, what am i working on, what am i stuck on
    - Time boxed Model:
        - Each sprint is max 1mo
        - Fixed number of tasks in each sprint
        - Daily scrum meeting 15 minutes max
        - Each sprint results in a:
            - Sprint review (0.5-1 hr)
            - Sprint retrospective: 1-3 hrs
    - The sprint cycle:
        - Starting point is the product backlog, which is a list of work to be done
        - Selection phase: involves the product team and user to decide which features and parts to include in a sprint
        - Teams organise each other to develop the software

        - Isolation of developers from customer is mandatory during this developmental cycle
        - All communication between them go through the Scrum Master
        - Scrum master role is to protect development team from external distractions
    - Sprint backlog:
        - No more than 300 tasks per list
        - >16hr task should be broken down
        - Team can add/subtract items from the list, product owner cannot
    - Velocity: amount of work done during one sprint calculated by adding up all successful story points
    - Burn down Chart: work left to do vs time left (y vs x). Usually includes a 1d static prediction of what should the work look like over time

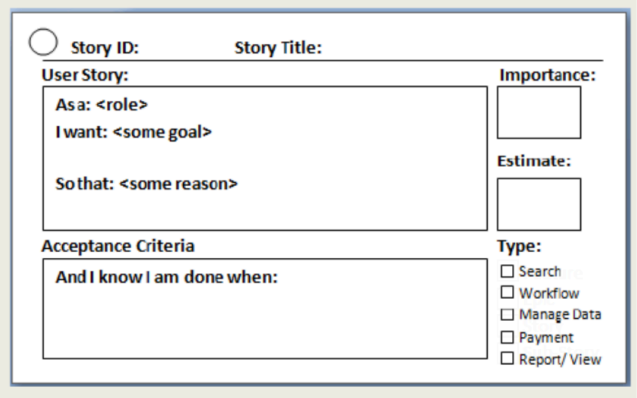
Concept	Description
Project	Discreet set of end user requirements that have been grouped, prioritised and funded.
Requirement	The end user statement that outlines their information need.
Sprint	A sprint is a 1 to 4 week time-boxed event focused on the delivery of a subset of User Stories taken from the Project Backlog.
Project Backlog	The Project Backlog is the current list of User Stories for the Project. User Stories can be added, modified or removed from the Backlog during the Project.
Sprint Backlog	Subset of User Stories from the Project Backlog that are planned to be delivered as part of a Sprint.
User Stories	The User Story is a one or two line description of the business need, usually described in terms of features.
Tasks	Tasks are the activities performed to deliver a User Story.
Technical Debt	This refers to items that were either: • missing from the Planning meeting; or • deferred in favor of early delivery.

47

I couldnt be asked to type this shit out

    - User Stories:
        - As a <type of user> I want <some goal> so that <some reason>

- Detail Adding:
  - Split user story into multiple stories
  - By adding conditions of satisfaction



-

- Requirement Engineering:
  - Process of establishing the services that the customer requires from a system

User requirement definition

1. The MHC-PMS shall generate monthly management reports showing the cost of drugs prescribed by each clinic during that month.

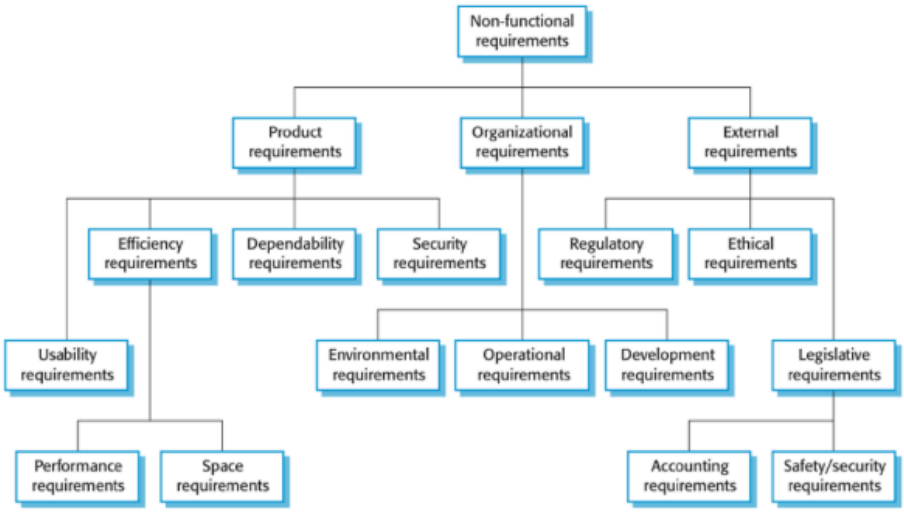
System requirements specification

1.1 On the last working day of each month, a summary of the drugs prescribed, their cost and the prescribing clinics shall be generated.
1.2 The system shall automatically generate the report for printing after 17.30 on the last working day of the month.
1.3 A report shall be created for each clinic and shall list the individual drug names, the total number of prescriptions, the number of doses prescribed and the total cost of the prescribed drugs.
1.4 If drugs are available in different dose units (e.g. 10mg, 20 mg, etc.) separate reports shall be created for each dose unit.
1.5 Access to all cost reports shall be restricted to authorized users listed on a management access control list.

-
  - User Requirements audience: client managers, system end-users, client engineers, contractor managers, system architects

- System Requirements audience: System end users, client engineers, system architects, software engineers
- System Stakeholders: End Users, System Managers, System Owners, External Stakeholders
- Requirements (Traditional):
    - Functional: describe functionality, what the system should do, expected outcome
    - Non-functional requirements: system properties and constraints

Types of nonfunctional requirement



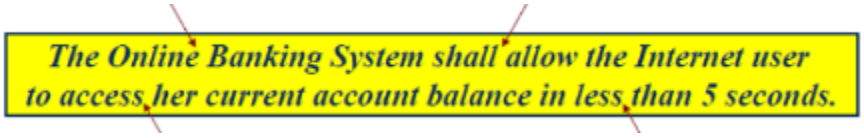
    - Non functional metrics:

Property	Measure
Speed	Processed transactions/second User/event response time Screen refresh time
Size	Mbytes Number of ROM chips
Ease of use	Training time Number of help frames
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portability	Percentage of target dependent statements Number of target systems

-

- Requirement Engineering Processes:
    - Requirement Elicitation
    - Requirement Analysis
    - Requirement Validation
    - Requirement management
        - Elicitation and Analysis:
            - Requirements Discovery
            - Requirements classification and organisation
            - Requirement prioritisation and negotiation
            - Requirement specifications
        - Elicitation techniques:
            - Traditional: questionnaires and surveys
            - Group techniques: brainstorming and focus groups, RAD workshops
            - Contextual techniques: ethnographic participation(observation in person) - focused ethnography: ethnography + prototyping, answering questions on prototypes
            - Cognitive techniques: protocol analysis (thinking outloud for observer to take insights and corrections)
            - Laddering: answer of user becomes next question
            - Card sorting: ask stakeholders to sort cards on importance and domain of a tasks
        - Requirement Checking:
            - Validity: system has functions that support customer needs

            - Consistency: requirement conflicts?
            - Completeness: are all functions needed included?
            - Realism: Are they implementable?
            - Verifiability: can requirements be checked
        - Validation Techniques:
            - Requirements reviews
            - Prototyping
            - Test-case generation
        - Requirement Reviews: regular reviews, client and staff should be involved
        - Review Checks:
            - Verifiability: testable?
            - Comprehensibility: properly understood?
            - Traceability: origin of requirement stated
            - Adaptability: can they be changed?
    - Requirement Prioritization: Weigers Prioritisation Technique
        - Value Total= Relative BenfitxWeight + Relative DetrimentxWeight
        - Sum Value Total
        - Value%=value total /sum value total *100
        - cost% = cost/total cost *100
        - risk% = risk/total risk *100
        - Prioirty: Value%/(Cost%*Relative_Cost_Weight+Risk%*Relative_Risk_Weight)
        - Give rank (1 is highest priority)
    - Requirement Management:
        - Changing requirements:
            - Problem analysis and change specification
            - Change analysis and costing
            - Change implementation
        - Management planning:
            - Requirement identification
            - A change management process: (impact and cost of any sort of requirement if changed)
            - Traceability policies: relationships between requirements
            - Tool support: tools used to record all this
- User Stories:
    - Used for AGILE Models in place of functional and nonfunctional requirements:
    - Parts:
        - Description
        - Conversation: Capturing further detail, could be infographics
        - Confirmation: what tests to be carried out that will define its success

    - Template for description:
        - As a <user Role> i want to <goal> so i can <reason>
        - Who (user role), what(goal), why(reason)
        - Has a title
    - How to split user stories:
        - Cake slices (equal chunks of increments)
        - INVEST:
            - Independent
            - Negotiable
            - Valuable
            - Estimable
            - Small
            - Testable
- Writing Requirements:
    - Parts:
        - Define system under discussion
        - Verb with correct identifier (shall or may)
        - Define positive result
        - Quality criteria
        - 

    - Unambiguous: open abbreviations and be clear
    - Testable (verifiable): “many” should be a number, adjectives are too vague like “accurate, correct” adverbs like “quickly”
    - Clear: no unnecessary verbiage
    - Correct: factually correct
    - Understandable: follow standard requirement given above
    - Feasible: realistic milestones
    - Independent: should not depend on another requirement verbally
    - Atomic: single traceable element or outcome
    - Necessary: should not be unnecessary to stakeholders
    - Implementation free: no code and working explanation as to how it will be made
    - Consistent: formats and recurring patterns should be similar (dates)
    - Nonredundant: same requirement on multiple requirements
    - Complete: specified for all conditions that could occur
    - Conflicts: should not conflict another requirement
- Requirement Document: Preface(expected readership and version history), Introduction, glossary, User requirements definitions, system architecture, system requirements specifications, system models (graphics), system evolution (assumptions and changes), appendices (configuration and development details like databases and hardware), Index

- Use “shall” for mandatory requirements, use “should” for desirable requirements
- Usage of natural language: lack of clarity, requirement confusion, requirement amalgamation
- Tabular specifications:

Condition	Action
Sugar level falling (r2 < r1)	CompDose = 0
Sugar level stable (r2 = r1)	CompDose = 0
Sugar level increasing and rate of increase decreasing ((r2 – r1) < (r1 – r0))	CompDose = 0
Sugar level increasing and rate of increase stable or increasing ((r2 – r1) ≥ (r1 – r0))	CompDose = round ((r2 – r1)/4) If rounded result = 0 then CompDose = MinimumDose

-
- Structure specification:

A structured specification of a requirement for an insulin pump

Insulin Pump/Control Software/SRS/3.3.2

Function Compute insulin dose: safe sugar level.

Description

Computes the dose of insulin to be delivered when the current measured sugar level is in the safe zone between 3 and 7 units.

Inputs Current sugar reading (r2); the previous two readings (r0 and r1).

Source Current sugar reading from sensor. Other readings from memory.

Outputs CompDose—the dose in insulin to be delivered.

Destination Main control loop.

-

Parameters	Functional Requirement	Non-Functional Requirement
What it is	Verb	Attributes
End-result	Product feature	Product properties
Capturing	Easy to capture	Hard to capture
Objective	Helps you verify the functionality of the software.	Helps you to verify the performance of the software.
Area of focus	Focus on user requirement	Concentrates on the user's expectation.
Documentation	Describe what the product does	Describes how the product works
Type of Testing	Functional Testing like System, Integration, End to End, API testing, etc.	Non-Functional Testing like Performance, Stress, Usability, Security testing, etc.
Test Execution	Test Execution is done before non-functional testing.	After the functional testing
Product Info	Product Features	Product Properties