PV258 - Software requirements engineering

Requirements Engineering phases:

- **Domain understanding** – All activities related to the understanding of the domain
- **requirements elicitation** – is the collection of requirements from the stakeholders by means of several techniques (meetings, prototypes, interviews, etc...)
- **requirements analysis and negotiation** – Requirements are analysed, and negotiated with the stakeholders
- **requirements specification** – The elicited requirements are modeled in a formal and unambiguous way so that they could be validated and used for the design
- **requirements validation** – Requirements are validated against stakeholder's needs
- **requirements prioritization** – Ordering requirements according to stakeholders' needs

Costs of Changing Requirements: two alternative models

- **In traditional models - Waterfall model**
  - The curve for the Cost of change for the waterfall model tends to grow exponentially depending on the different phases
  - Assuming software development in waterfall model, RE is the first phase of software development
  - NO change of requirements at a later phase, OR huge costs for going through all the phases

- **Other more agile approaches - Extreme Programming (XP)**
  - states that with the right practices and technology, the curve of the cost of change can be kept low across all development phases
  - emphasizes the role of Requirement Engineering – it is a constant activity (Planning Game)
  - because of the assumption of the flat cost of change curve, XPists make big decisions as late as possible
    - to defer the cost of making the decisions
    - to have the greatest possible chance that they are right
  - engineering process happens continuously
  - user stories are used to represent requirements in a concise and unambiguous way
  - The customer is always in contact with the development team to reduce the possibility of misunderstandings in the requirements. As such, small releases and incremental development are 'must have' practices of XP
  - **advantages** - save time and cost, visible and accountable project, increase employee satisfaction
  - **disadvantages** - does not measure software quality, not good if team is not in one place

3 major problems faced in developing software - 3 beasts

- **Uncertainty** - Difficulty of communication among different parties
- **Irreversibility** - Time and most resources are not recoverable
- **Complexity** - Lots of information to be kept for a project

Plan-Driven / waterfall / heavy weight

- Reduce uncertainty and complexity
- Guiding development via long term plans
- Strict end-to-start dependencies among stages
- Irreversibility increases inevitably as project progresses

Agile / Lean

- Original idea and plans changed progressively
- Changes in development is essential, not evil
- Accepts and addresses the 3 beasts rather than avoiding them
- It is up to developers to decide which approach is more suitable for the project

The Iron Triangle

- In a development process we have 3 variables that are interrelated: the requirements to be developed, the cost, and the schedule
- In a waterfall model, requirements are typically fixed and set at the beginning of the project
- if we want to meet costs and deadlines we should lower quality and vice versa

Spiral

initial pass around the spiral is intended primarily to understand requirements and perform some validation of the requirements before more serious development begins. The following larger spiral is involving all the development phases with constant feedback

Rapid Application Development (RAD)

you could build the system fast enough before the requirements changed, you would be more successful. And if you did get it wrong, the tools are sufficiently easy and lightweight

Rational Unified Process (RUP)

RUP was the first model to see the necessity of the overlap between different activities in the development process

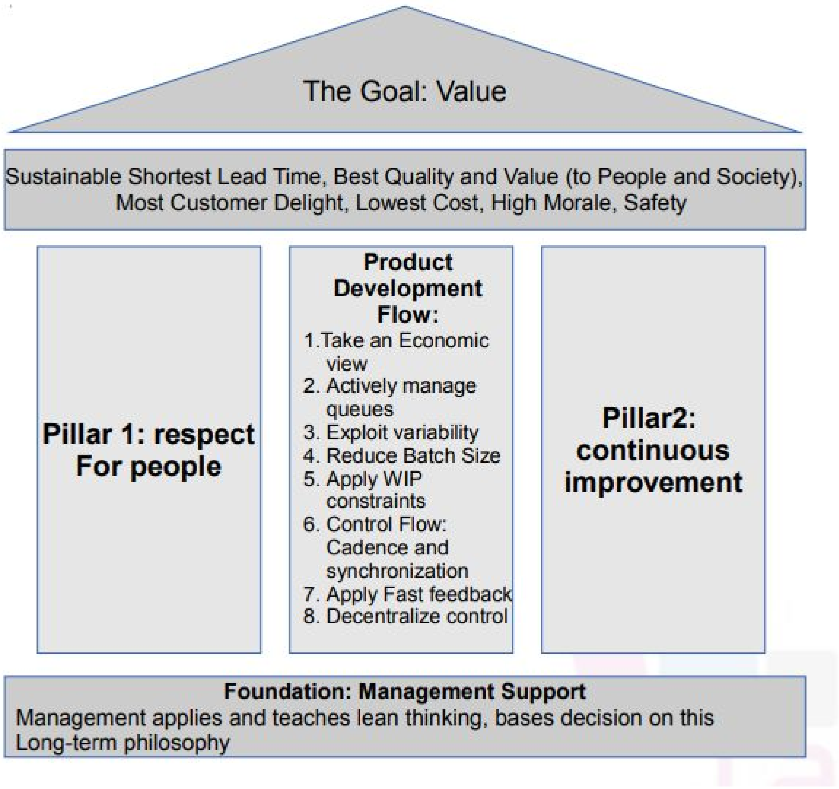
Agile Methodologies

- **Agile Manifesto**
    - Individuals and interactions over Customer collaboration
    - Processes and tools over contract negotiation
    - Working software over comprehensive documentation
    - Responding to change over following a plan

- **Agile principles**
  - satisfy the customer through early and continuous delivery of valuable software
  - Welcome changing requirements, even late in development
  - Working software is the primary measure of progress
  - Deliver working software frequently
  - Business people and developers must work together
  - Build projects around motivated individuals.
  - face-to-face conversation
  - promote sustainable development
  - Continuous attention to technical excellence and good design
  - Simplicity - the art of maximizing the amount of work not done
  - selforganizing Teams

Lean Software Development

- **Eliminate Waste**
- **Amplify learning** Software Development is a continuous learning process
- **Decide as late as possible** helps to fight uncertainty in the software development process
- **Deliver as fast as possible**
- **Empower the team** let the team do self-management and trust their decisions as they are those that actually "do"
- **Build Integrity in** build integrity in the system so that the customers perceive overall quality
- **See the whole**


The Goal: Value
Sustainable Shortest Lead Time, Best Quality and Value (to People and Society), Most Customer Delight, Lowest Cost, High Morale, Safety
Pillar 1: respect For people
Product Development Flow:
1.Take an Economic view
2. Actively manage queues
3. Exploit variability
4. Reduce Batch Size
5. Apply WIP constraints
6. Control Flow: Cadence and synchronization
7. Apply Fast feedback
8. Decentralize control
Pillar2: continuous improvement
Foundation: Management Support
Management applies and teaches lean thinking, bases decision on this Long-term philosophy

Types of Requirements:

- **Business requirements -** Define the business objectives of the organization or the customer → usually represented in a vision and scope document
- **User requirements -** the goals or tasks the users must be able to perform with the product and that can provide value to someone
    - **Use Cases (UC) -** describes a sequence of interactions between a system and an external actor that results in the actor being able to achieve some outcome of value - should be in form of **use case diagram** and **use case template**
    - **User Stories**
        - are a simple, few-sentence description of a functional requirement that provides value to a stakeholder
        - User stories are written by customers
        - User stories are usually written on an index card which is passed to the team member who will work on it
        - Developers estimate how long the stories might take to implement - **ideal development time**
        - **User Story Attributes**
            - **Title**
            - **Acceptance Test -** List the unique identifiers of the acceptance tests for the user story, An acceptance test is a test case written by the customer (in partnership with the developers), Verifies that the user story has been correctly implemented
            - **Priority** most important stories can be done first
            - **Story Points** The number of days of ideal development time
            - **Description**
        - **User story criteria**
            - **Stories must be understandable to the customer**
            - **must provide value to the customer**
            - **Stories need to be of a size that several of them can be completed in each iteration**
            - **Stories should be independent of each other as possible**
            - **Each story must be testable**
- **Functional requirements -** the behaviors the product will exhibit under specific conditions. They describe what the developers must implement to allow users to complete the expected tasks - form of “shall” natural language statements in a **Software Requirements Specification** (SRS) document
- **System requirements -** the requirements for a product that is composed of multiple components or subsystems - **System Requirements Specification**
- **Business rules -** they constrain the development of the project to be complaint with domain rules (e.g. corporate policies, government regulations - **decision tables**
- **Non Functional Requirements -** often referred as quality requirements, they describe different characteristics in term of performance, reliability, maintainability

Product Vision & Project Scope

- **Product vision -** describes briefly the product that will achieve the business objectives. The vision must state what the product will be about and what it will become in the long term
- **Project scope -** defines the boundary of what is within the product under development and what is outside

CRC method

- Finds the classes of objects constituting the system, Defines responsibility of the classes and Finds collaborations (exchange of info) needed among classes to fulfill the responsibilities
- CRC method uses index cards
  - **Class Name** - single object of the class
  - **Main Responsibility**
  - **Responsibilities**
  - **Collaborators -** Can a class fulfill its responsibilities alone?

Software Requirements Elicitation

Requirements elicitation is the process of finding and formulating requirements

Elicitation Stages

- **Objective setting -** The organizational objectives should be established including general goals of the business, an outline description of the problem to be solved, why the system is necessary and the constraints on the system
- **Background knowledge acquisition -** Background information about the system includes information about the organisation where the system is to be installed, the application domain of the system and information about existing systems
- **Knowledge organisation -** The large amount of knowledge which has been collected in the previous stage must be organized and collated.
- **Stakeholder requirements collection -** System stakeholders are consulted to discover their requirements

Requirements Elicitation

- **Stakeholder analysis -** Meetings and interviews may be appropriate to gather such information
- **Interviewing** - 1. Identifying candidates 2. Preparing for an interview 3. Conducting the interview 4. Following up
- **Observation -** observe users in their daily tasks and gather more information
- **Task demonstration** - show how a task is currently performed
- **Study of documents** old systems documents etc.
- **Questionnaires**
- **Brainstorming Sessions**
- **Focus Groups** They stimulate people to come up with problems during current work procedure, and to present possible ideal ways of doing things
- **Domain Workshops** serve to map the business processes to tasks and requirements

- **Design Workshops -** cooperate to produce & design part of the system
- **Prototyping -** prototype is a simplified version of part of the final system.
    - **Product level requirements**: the prototype shows that the required functionality is useful and feasible
    - **Design level requirements**: the prototype shows that it satisfies the goal of the system
    - **Throw-away prototyping -** The requirements which should be prototyped are those which cause most difficulties to customers and which are the hardest to understand. Prototype throw to bin.
    - **Evolutionary prototyping -** intended to deliver a workable system quickly to the customer. Functional part of system.
- **Pilot Experiments -** allows to study a system in an organization on trial basis, but with real production data
- **Study of similar companies**
- **Ask suppliers**
- **Negotiation -** resolve conflicts that can happen among stakeholders
- **Risk Analysis**
- **Cost-Benefit Analysis**
- **Goal-Domain Analysis -** looks at the relation between business goals and tasks
- **Domain-Requirements Analysis -** works on lower level as GDA

Requirements Prioritization

Aspects - importance, costs, penalty, tiem, risk, volatility …

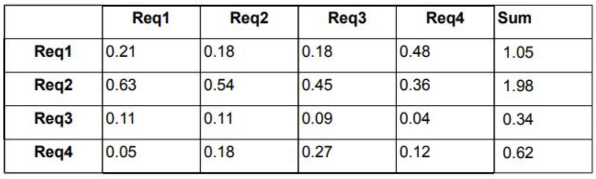
Requirements Prioritization Techniques

- **1-10 Ranking -** the developers pick their top 10 requirements and rank them. They need then to be aggregated to find the overall list of requirements and their ordering
- **Binary search tree based prioritization -** create binary tree with comparing each elements by priorities - higher priority go to right lower left
- **100$ Test**
- **MoSCOW method -** Must, Should, Could, and Won't have requirements
- **Kano Analysis -** based on a model that focuses on customer's satisfaction, Attributes of the product are evaluated according to the fact that they can provide satisfaction/dissatisfaction if fulfilled or not
- **Wieger's method -** Weights several categories such as relative benefit, relative penalty, relative risk, and relative cost, to come out with a weighted priority list.

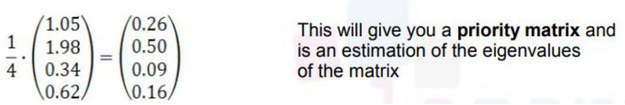
$$P_j = \frac{Value_j}{w_1 \cdot Cost_j + w_2 \cdot Risk_j}$$

Analytic Hierarchy Process (AHP)

- list of unambiguous requirements elicited from stakeholders - create matrix
- compare requirements with importance values - 1,3..7,9
- calculate a new matrix where each element is divided by the sum of the column value



	Req1	Req2	Req3	Req4	Sum
Req1	0.21	0.18	0.18	0.48	1.05
Req2	0.63	0.54	0.45	0.36	1.98
Req3	0.11	0.11	0.09	0.04	0.34
Req4	0.05	0.18	0.27	0.12	0.62

- 

$$\frac{1}{4}\cdot\begin{pmatrix}1.05\\1.98\\0.34\\0.62\end{pmatrix}=\begin{pmatrix}0.26\\0.50\\0.09\\0.16\end{pmatrix}$$

This will give you a **priority matrix** and is an estimation of the eigenvalues of the matrix

- then prioritization due to cost and value of requirement
- ahp may be inconsistent - then we should define index for pair comparison

Software Quality Function Deployment

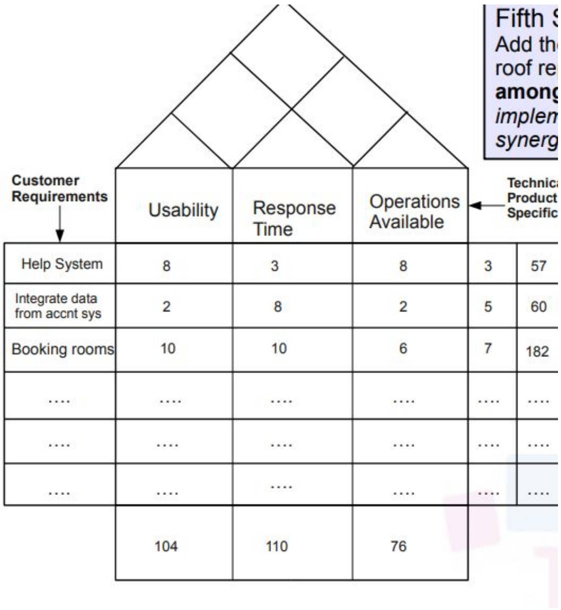
Consists in the application of a technique called Quality Function Deployment (QFD) to Software Engineering

QFD

- provides a systematic approach to identify which requirements are a priority for whom, when to implement them, and why
- **Product Planning, Product Design, Process Planning, Process Control**
- In QFD this process is repeated for the different phases

SQFD

- focused on building the house of quality
- It is a simplified version of the QFD that does not contain all the information
    - collect customer rq. in rows and technical functionality in columns
    - Customers start filling the level of relation between customer requirements and technical product specifications
    - start filling the customer requirements priorities - Total priority values for technical functionalities are calculated by multiplying each cell value with the priority and summing up
    - Add the roof to the quality house. The roof represents the interactions among different column factors



Customer Requirements	Usability	Response Time	Operations Available		
Help System	8	3	8	3	57
Integrate data from accnt sys	2	8	2	5	60
Booking rooms	10	10	6	7	182
....	....	....	....	....	....
....	....	....	....	....	....
....	....	....	....	....	....
	104	110	76		

The Planning Game

- Planning Poker is run on the story by the development team
- Customers decide on the priority of the story
- The process is repeated - result ordered set of cards
- **Effort Estimation is needed in AMs**
  - **Determining cost -** Effort is used indirectly to calculate cost
  - **Establishing prioritization -** Effort is used in conjunction with other aspects for the prioritization of the user stories
  - **Scheduling and commitment**
- **Story point -** It represents the size of a story in terms of different aspects
  - Knowledge: what is our level of understanding of a story?
  - Complexity: what is the difficulty of implementation?
  - Volume: How much implementation is involved?
  - Uncertainty: What do we not know about it and how it can affect the estimate?

The Concept of Velocity

Velocity is the term we use to refer to the number of points that a development team can implement in an iteration

- based on historical data
- Velocity depends on the team composition
- cannot be compared across other teams because story points are estimated according to the specific project

$$ndr = dpi \cdot \frac{bse}{velocity}$$

ndr = number of days required
dpi = days per iteration
bse = backlog size estimate
v = velocity

- Velocity is very often used together with burndown charts, to track the effective execution of iterations
- Velocity is used to track the expected speed of each iteration

Non-Functional Requirements

Place restrictions on the product being developed, the development process, and specify external constraints that the product must meet

Non functional requirements must have precise metrics to measure. e.g. Performance - processes/transactions per second

McCall & Matsumoto dividing factors across three levels: Operation, Revision, and Transition

- **Operation**
    - **Integrity** (How well system handles physical disturbances and prevents malicious access attempts
    - **Correctness** (How many errors are present in the system)
    - **Reliability** (How frequently the system malfunctions)
    - **Usability** (How easy it is to learn the system, how efficient it is for carrying out day to day task, etc.)
    - **Efficiency** (How fast the system responds, how many resources it uses, how accurately it computes values, etc.)
- **Revision** (Maintenance and extension)
    - **Maintainability** (How easy it is to locate and repair errors)
    - **Testability** (How easy it is to test the system)
    - **Flexibility** (How easy it is to expand the system with new features)
- **Transition** (Use in new technical surroundings)
    - **Portability** (How easy it is to move the system to a new software or hardware platform)
    - **Interoperability** (How easy it is for the system to cooperate with other systems
    - **Reusability** (How easy it is to reuse parts of the software in other systems)

ISO/IEC 9126

Defines Software Quality as a series of characteristics ●
Defines six main categories: **Functionality, Reliability, Usability, Efficiency, Maintainability, Portability**

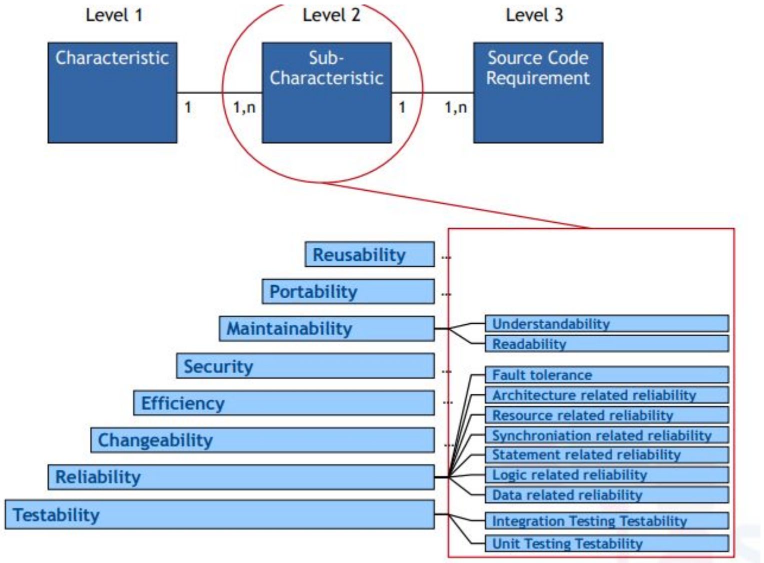
- **Functionality** (Set of attributes that resemble a relation to functionality)
    - **Suitability**. The capability of the software product to provide an appropriate set of functions for specified tasks and user objectives.
    - **Accuracy**. The capability of the software product to provide the right or agreed results or effects with the needed degree of precision.
    - **Interoperability**. The capability of the software product to interact with one or more specified systems.
    - **Security**. The capability of the software product to protect information and data so that unauthorised persons or systems cannot read or modify them and authorised persons or systems are not denied access to them.
    - **Functionality Compliance**

- **Reliability** (Set of attributes related to keeping the stated performance under the given conditions)
    - **Maturity**. The capability of the software product to avoid failure as a result of faults in the software.
    - **Fault Tolerance**. The capability of the software product to maintain a specified level of performance in cases of softwarefaults or of infringement of its specified interface.
    - **Recoverability**. The capability of the software product to re-establish a specified level of performance and recover thedata directly affected in the case of a failure.
    - **Reliability Compliance**
- **Usability** (Set of attributes related to the effort of using the system)
    - **Understandability**. The capability of the software product to enable the user to understand whether the software is suitable, and how it can be used for particular tasks and conditions of use.
    - **Learnability**. The capability of the software product to enable the user to learn its application.
    - **Operability**. The capability of the software product to enable the user to operate and control it.
    - **Attractiveness**. The capability of the software product to be attractive to the user.
    - **Usability Compliance**.
- **Efficiency** (Set of attributes related to the performance of the system given the resources used)
    - **Time Behaviour**. The capability of the software product to provide appropriate response and processing times and throughput rates when performing its function, under stated conditions.
    - **Resource Utilisation**. The capability of the software product to use appropriate amounts and types of resources when the software performs its function under stated conditions.
    - **Efficiency Compliance**.
- **Maintainability** (Set of attributes related to the effort needed to make changes and adapt it to changing conditions)
    - Analyzability. The capability of the software product to be diagnosed for deficiencies or causes of failures in the software, or for the parts to be modified to be identified.
    - Changeability. The capability of the software product to enable a specified modification to be implemented.
    - Stability. The capability of the software product to avoid unexpected effects from modifications of the software.
    - Testability. The capability of the software product to enable modified software to be validated.
    - Maintainability Compliance
- **Portability** (Set of attributes related to the effort needed to system from one environment to another)

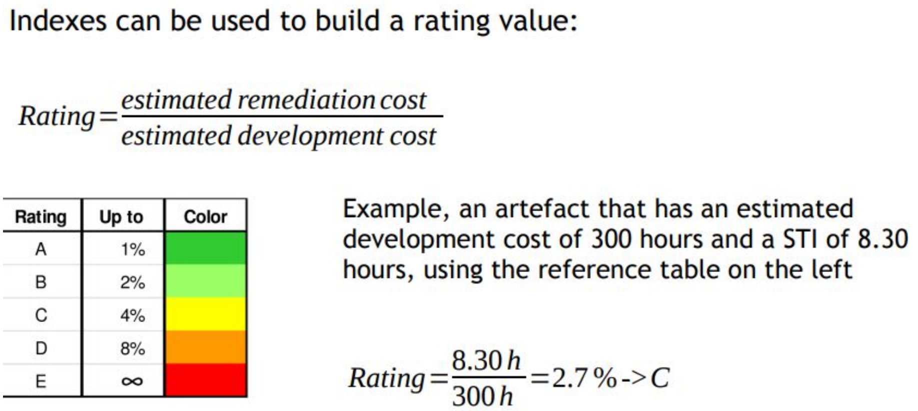
- **Adaptability**. The capability of the software product to be adapted for different specified environments without applying actions.
- **Installability**. The capability of the software product to be installed in a specified environment.
- **Co-Existence**. The capability of the software product to co-exist with other independent software in a common environment sharing common resources.
- **Replaceability**. The capability of the software product to be used in place of another specified software product for the same purpose in the same environment
- **Portability Compliance**

SQALE (Software Quality Assessment Based on Lifecycle Expectations)

- is a quality method to evaluate technical debts in software projects based on the measurement of software characteristics
- shows an application of ISO/IEC 9126 Standard
- SQALE quality model is based around three levels, the first one including 8 software characteristics



- For each of the source code requirements we need to associate a remediation function that translates the non-compliances into remediation costs
- Sums of all the remediation costs associated to a particular hierarchy of characteristics constitute an index:



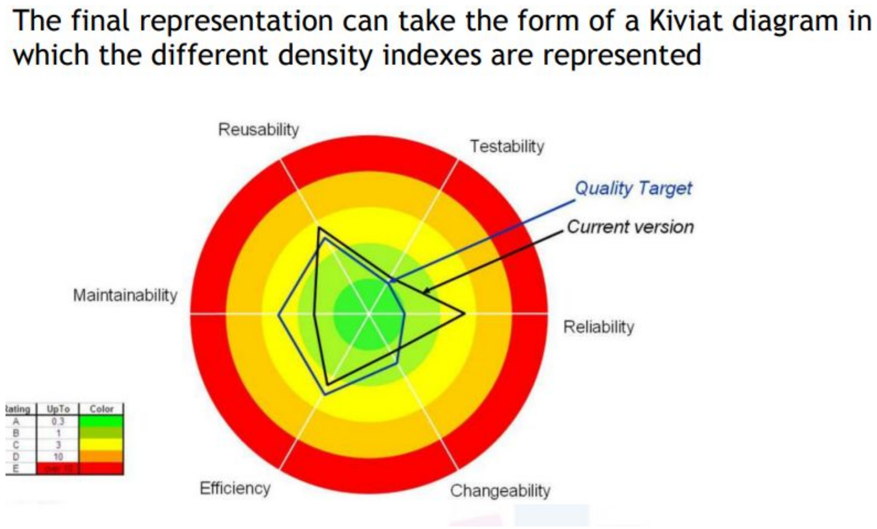
Indexes can be used to build a rating value:

$$Rating = \frac{estimated\ remediation\ cost}{estimated\ development\ cost}$$

Rating	Up to	Color
A	1%	🟩
B	2%	🟩
C	4%	🟨
D	8%	🟧
E	∞	🟥

Example, an artefact that has an estimated development cost of 300 hours and a STI of 8.30 hours, using the reference table on the left

$$Rating = \frac{8.30\,h}{300\,h} = 2.7\,\% \text{->} C$$



The final representation can take the form of a Kiviat diagram in which the different density indexes are represented

Rating	UpTo	Color
A	0.3	🟩
B	1	🟩
C	3	🟨
D	10	🟧
E	over 10	🟥

NFR as Constraints

From the point of view of user stories, NFR can be seen as constraints placed on top of them

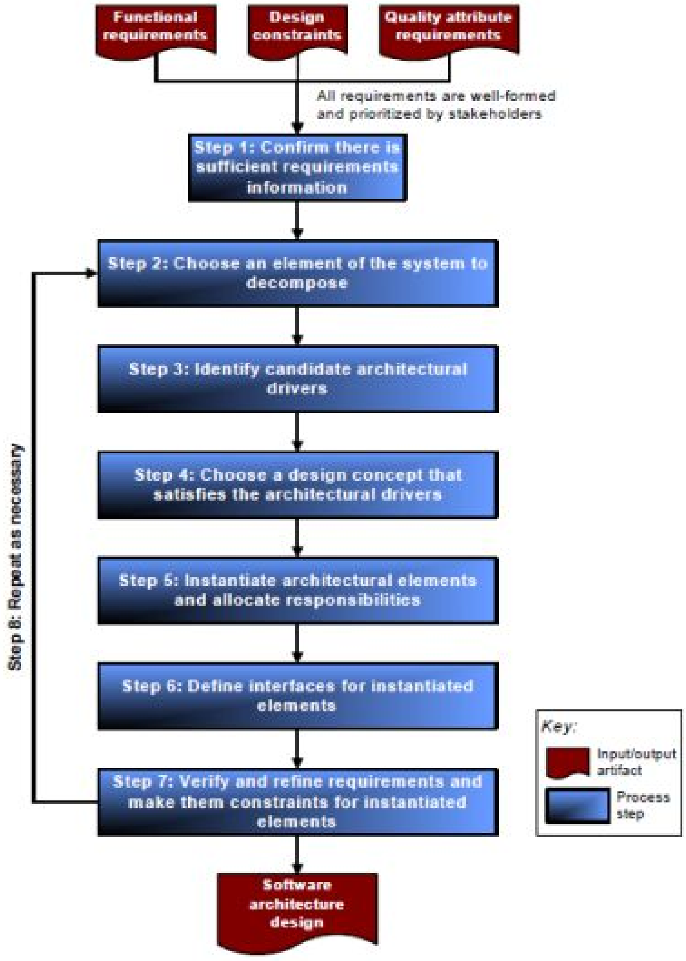
Some strategies that can be applied to maintain NFRs:

- Create a separate backlog
- Store and manage them in a wiki
- Maintain a supplementary specification
- Use NFR in the definition of “done” of an iteration/user story – linking them to the backlog/wiki

The Attribute-Driven Design

ADD follows a Plan-Do-Check cycle

- **Plan**. The quality attributes and the design constraints are used to determine the type of elements to be used in the architecture
- **Do**. Elements are instantiated to respect quality requirements
- **Check**. The design is analyzed to check whether all the requirements are met


Functional requirements
Design constraints
Quality attribute requirements
All requirements are well-formed and prioritized by stakeholders
Step 1: Confirm there is sufficient requirements information
Step 2: Choose an element of the system to decompose
Step 3: Identify candidate architectural drivers
Step 4: Choose a design concept that satisfies the architectural drivers
Step 5: Instantiate architectural elements and allocate responsibilities
Step 6: Define interfaces for instantiated elements
Step 7: Verify and refine requirements and make them constraints for instantiated elements
Step 8: Repeat as necessary
Software architecture design
Key:
Input/output artifact
Process step

Business Rules

they constrain the development of the project to be compliant with domain rules (e.g. corporate policies, government regulations, industry standards, computational algorithms, etc...). Not strictly part of the requirements → modeled by means of decision tables

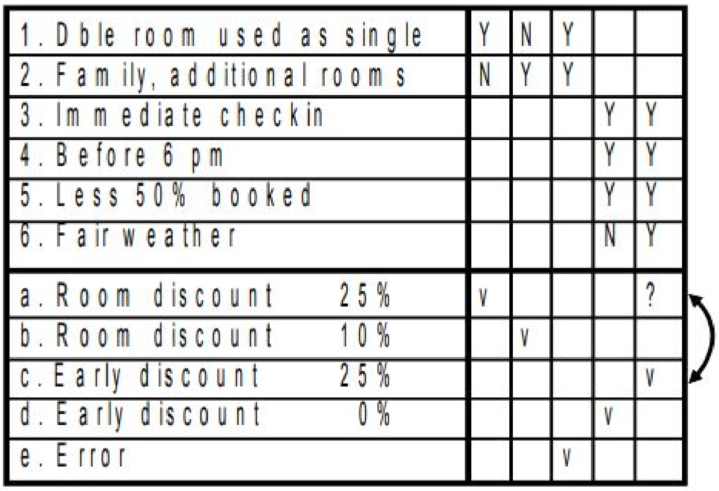
- **Facts** - statements that are true about the business at some point in time - Every chemical container has a unique bar code identifier.
- **Constraints** - statements that restrict the actions that the system or the users are allowed to perform - All software applications must comply with government regulations for usage by visually impaired persons

- **Action Enablers** - rules that trigger activities according to specific conditions - If the chemical stockroom has containers of a requested chemical in stock, then offer existing containers to the requester
- **Inferences** - creates a new fact from other facts - If a payment is not received within 30 calendar days after it is due, then the account is delinquent
- **Computations -** transformations of existing data in new data using mathematical formulas or algorithms - The domestic ground shipping charge for an order that weighs more than two pounds is $4.75 plus 12 cents per ounce or fraction thereof

HOW TO PROCESS BUSINESS RULES

Decision Tables

- Tables listing all possibilities given a certain domain & problem
- Easy to validate
- Easy to verify for user as well as developer
- Steps
  - Identify the conditions and the possible values
  - Compute the combinations of values
  - Identify the Repeating Factor, that means to divide remaining combinations by the number of possible values for that cause and start filling the rows
  - Find cases in which values for conditions are indifferent and make them as '-' or ' '
  - Join columns that are identical
  - Add the Actions

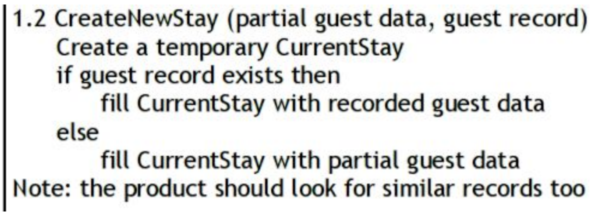


1. Dble room used as single	Y	N	Y		
2. Family, additional rooms	N	Y	Y		
3. Immediate checkin				Y	Y
4. Before 6 pm				Y	Y
5. Less 50% booked				Y	Y
6. Fair weather				N	Y
a. Room discount 25%	v				?
b. Room discount 10%		v			
c. Early discount 25%					v
d. Early discount 0%				v	
e. Error			v		

- **Advantages** :
  - Excellent for describing business rules
  - Easy to validate and verify
  - Suitable for rules with medium complexity – No loops or recursions within the rules

Textual Process Descriptions

- Outline of the program (the algorithm) in textual form
- Sometimes called mini-specs or pseudocode
- Useful for design-level requirements
- Sometimes useful for business rules



```
1.2 CreateNewStay (partial guest data, guest record)
    Create a temporary CurrentStay
    if guest record exists then
        fill CurrentStay with recorded guest data
    else
        fill CurrentStay with partial guest data
Note: the product should look for similar records too
```

- **Pros and Cons**
  - **Verification**: Textual description can be used as a basis for implementation
  - **Verification** is done through testing
  - **Validation**: Can be done by customers by testing, but usually takes so much time that the customer is not willing to do it

State Diagrams

- Diagram showing how something changes from one state to another
- Good for finding missing or obscure functions
- Describe how a certain entity changes state as a result of various events
- **Pro and Cons**
  - **Verification**: Good for development and testing, most developers are familiar with them
  - **Validation**: Huge state diagrams can easily extend over several pages but still retain their readability
  - Difficult to understand for some customers since some can not distinguish state diagrams from dataflow or activity diagrams

Activity Diagram

- A diagram showing the flow of control and the flow of data
- Handles concurrency and synchronization
- Good for analysis
- Activity diagrams can be useful for
  - Designing new human activities in the domain
  - Augment information derived from business rules
  - Specifying communication between technical components, including thread and synchronization aspects
  - outlining the internal structure of large programs
- **Pros and Cons**
  - Validation: it is difficult for customers to see if the diagram correctly reflects the requirements
  - Actual data communication cannot be described in detail
  - Can take quite considerable amount of space - all obvious actions might be omitted