Requirements for a system are the descriptions of the services that a system should provide and the constraints on its operation

The process of finding out, analyzing, documenting & checking these services is called requirements engineering (RE)

- User requirements are broad statements, NL plus diagrams
- System requirements are more detailed descriptions of services
- SRS document

User story

As a <type of user> I want <some goal> so that <some reason>

Acceptance Criteria - step by step

RE Tasks

Inception, Elicitation, Elaboration, Negotiation, Validation, Management

Inception, 1st, 2nd, 3rd step

DFD Level 0 or context diagram

DFD

- □ → entity
- → → flow
- ○ → process
- = → DB

one process node & no data store

max processes in DFD Level 1 = 5

verb is process, noun is everything else

DFD Level 1 contains data stores

Use Case Diagram - remember SDA

actor, include, extend, << >> sub-actor

System actors on right side

ER Diagram - DB

- ○ → attributes
- 1-1, 1-n, n-n relationship
- ◇ → relationship
- ▭ → entities
- ◎ → multivalued

F. requirement describes interaction btw system & environment

Class Diagram

Assoc, Aggreg, Comp

→ classes, variables & functions

how the system should behave with stimuli

CRC (Class-Responsibility Collaborator)

Class	
Responsibility	Collaborator

CRC Modelling

→ no reviewer should have 2 cards that will collaborate

→ leader reads use case - as leader comes to a naming object, passes a token to a person holding the corresponding class card

→ when token is passed, card person tells responsibilities on card

→ Even if error then revising list or new classes

State Diagram - SDA

- ▭ → state
- ● → start ◉ end
- [ ] → guard condition (if statement)

Sequence Diagram

alt, loop

Swimlane Diagram

Actor 1 | Actor 2

Petri Nets

Decision Table

Condition	Condition Entries
Action	Action Entries

Requirements Prioritization

→ essential, desirable, optional

Characteristics of Requirements

→ correct, consistent, complete, unambiguous, feasible, relevant, testable, traceable

Functional / NF Requirements

Functional describes what product should do

NF describes how the product should do it

F → system must do [req]

NF → system shall be [req]

NF → constraint, design, process

Testable Requirements

Unambiguous, quantitative, nouns

Validation ensures we build right system

Verification ensures we build system right

Measuring Req

→ from 1 to 5, lower score is good

low means you understand requirement

Cloning: borrow a design/code & make minor adjustments

reference model: standard generic architecture

Decomposition & Views

Functional Decomposition

→ top level → first → second

Feature oriented → features to modules

Data " → the data to modules

Process " → process to modules

Event " → events to modules

Object " → object to modules

Architectural Styles

NF Requirement → restriction in the system that limits our choice

Pipe & Filter

pipe → ▭ → filter

achieved by passing data input through sequence of data transforming components called filters

Adv

→ designer can understand system effect on input, output, filters used easily on other systems, evolution is simple, allows concurrent execution

Dis

→ support fixed data, parse data, & compute hamper performance

→ Not good for handling interactive application

UI Rules

- Make interface consistent, Place the user in control, Reduce user's memory load

Client Server

request reply

- ○ client
- □ server
- ⟷ request/reply

Peer-to-Peer

each component executes as its own process is client & server to other components

→ has interface

Adv

→ scale

→ added, system capabilities increase

→ Network failures tolerant

Dis

→ add requests

Not to use

file contents change

speed has importance

file quality critical

trust required

Publish Subscribe

Adv

→ strong support for evolution

→ component can be added without affecting others

Dis

→ need shared repository

→ Difficult to test

Repositories

- ○ → knowledge source
- = blackboard
- → read/write

Layering

○ components

Adv

→ useful decomposition

Dis

not easy to structure layers

→ performance cost

Call & Return

Main Program/Subprogram

Quality Attributes

modifiability, performance, security, reliability, robustness, usability, business goals

Cost-Benefit Analysis

ROE = Benefits/Cost

%age ROE = $\frac{\text{Benefits} - \text{Cost}}{\text{Cost}} \times 100\%$

Payback = $\frac{\text{Cost}}{\text{Benefit}}$

Trade off Analysis

- Data Abstraction Solution

circular shift, alphabetic shift

changes to system functionality not easy

- Implicit Invocation Solution

Data in ADT

- Repository solution
- Pipe & Filter

Implicit Invocation best - easy to change algos, good performance, efficient data representation

Analysis Module

Scenario based, use case, stories

Class " class diagram

Behavior (state, sequence)

Flow (DFD, data models)

Data Flow

→ Make DFD Level 1, 2, 3

→ Make boundary

→ Put rule diagram

→ Do first level factoring & refine it

Trade-off

Repository

→ breaks problem into four primary functions, input, circular shift, alphabetizer & output

→ coordinates by a single program

→ data are localized

→ Design is difficult to change modules access & manipulate the data directly

→ some of the elements are reusable

Data Abstraction Solution

data computed by each computational module is stored in model

- circular shift contains index to words in text
- Alphabetic shift maintains a sorted version of this
- easier to reuse
- changes to functionality not easy

Implicit

Data in ADTs

ADTs generic more reusable than data abstraction

→ easily extended

→ multiple compute methods may execute by event, which order

Pipe & Filter

filter modules

- easily extended
- → operate in parallel
- space inefficiencies

circular shifts can no longer be rep as indices of text, duplicated copies of text

Component

→ requires, → provides, interfaces (2 icons)