REQUIREMENT

- Requirement engineering is about clarifying the requirements and avoid misunderstandings.
- Important to use complete sentences and verbs such as shall, must & will
  careful with all, never, each

Iterative process – 4 steps

1) Elicitation
   - Understand what the customer truly need
   - Obtained internally or externally:
     - Goals / Domain knowledge / Stakeholders / Environment
   - Can be done through different methods:
     - Interviews / Scenarios / Prototypes / Observations

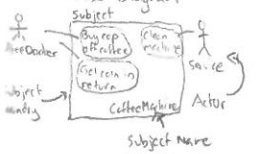
Interviews
- Start
- Q & A
- Summary feedback
- Thank you!
- Whats next

2) Analysis
   - Detect & resolve conflicts between requirements
   - Classify requirements for more effective management
   - Discover bounds of software
   - Define interaction with the software
   - Elaborate high-level req → Define detailed req

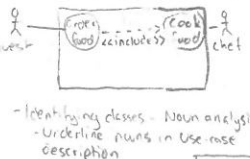
Requirement classification
- Functional vs. Non-functional
- Source
- Product or process req
- Priority
- Volatility vs. Stability

Prioritization
- Ineachral, category, 100$ method
- Good range between req. Not all critical
- Consider cost, risk & timeframe

Conceptual Modeling
- Use-case
  - Description of use-case
    "A collection of approaches ..."
- Use-case name
  Verb phrase
- Use-case Diagram



- Relations between use-cases

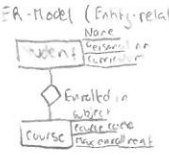


- Identifying classes - Noun analysis
  - Underline nouns in Use-case description
- Class model

To write good use-cases:
- Start by identifying list of actors
- Describe "sunny-day" use-case
- Use them to produce "rainy-day" use-cases

Requirement Abstraction model
RAM
Utilizing level of abstraction
1) Product level (goal)
2) Feature level (features)

- ER-Model (Entity-relationship)



SRS - Software Requirement Specification

Example: IEEE Std 830 - 1998

SRS

1) Introduction
   - Purpose
   - Scope
   - Definitions, acronyms, abbreviations
   - References
   - Overview
2) Overall description
   - Product perspectives
   - Product functions
   - User characteristics
   - General constraints
   - Assumptions & dependencies
   - Lower ambiguity levels
3) Specified requirements
   - Interface requirements
     - User interfaces
     - Hardware interfaces
     - Software interfaces
     - Comm. interfaces
   - Functional requirements
   - Performance requirements
   - Design constraints
   - Software system attributes
   - Other requirements
4) Supporting information
   - Index
   - Appendices

Pro & Cons of IEEE 830
- \+ Good checklist
- \+ Many ways to adapt organization
- \+ Many ways to detail requirements
- \- Takes time to read & understand
- \- Very general, needs to be tailored

Good SRS with the basic issues:
- Functionality
  What the software should do
- External interfaces
  How it interacts people and other software
- Performance
  Speed of functions
- Attributes
  Portability, maintainability, security etc
- Design constraints
  Languages, policies etc

5) Specification
   - Requirement Spec. (RS)
   - Software req. spec (SRS)
   - RS important when outsourcing

- Use-stories

"As a (role) I want to (something) so that (benifit)"
Priority: X
Estimate: Y

- Good for:
  - Smaller sub-projects
  - Frequent releases
  - User feedback
  - Prototyping
  - Elicitation
  - Functional req.

4) Validation
   - Prototyping
   - Simulation
   - Software review
   - Model checking
   - Formal proof
   - Acceptance testing

Traceability is key!

Functional requirements
- Describe behavior or feature of software.
- Tested by giving input → check output f(x)=y

"The user shall be able to add an item to the shopping bag"

Non-functional req
- Design constraints
  Limit/Mirror solution space
  "system shall be impl. in PHP"
- Quality req, possible to measure
  "Min response-time of 2 sec"

User interface requirement
- Easy to operate
- Quick to response
- User centric

Features
- Distinguishing characteristic of the system (both functional and non-functional)
- System shall have an SMS delivery notification service"

Requirements in a Good SRS are:
- Numbered
- Inspected
- Prioritised
- Unambiguous
- Testable
- Complete
- Consistent
- Traceable
- Feasible
- Modifiable
- Used for:
  - Operation
  - Maintenance
  - Customer
  - Developer etc

Iterative process
- Living document
- Priorities change
- Tech issues → Modifications

Quality factors
1) Correctness
2) Reliability
3) Efficiency
4) Usability
5) Integrity
6) Maintainability
7) Flexibility
8) Testability
9) Security
10) Portability
11) Reusability
12) Interoperability
13) Survivability
14) Safety
15) Manageability
16) Supportability
17) Replaceability
18) Functionality

Domains
- Scalability
- Reliability
- Security & encryption
- Environmental
- Maintainable
- Usability

Things to keep an eye on:
- Human bias
  More details in areas we know well

Characteristics of good Requirements
1) Unitary - Req addresses only one thing
2) Complete - Fully stated in one place with no missing information
3) Consistent - One req doesn't contradict another req
4) Non-conjugated - split req. if possible
5) Traceable - Meets wants stated by stakeholder, no more, no less
6) Current - Not obsolete reuse of time
7) Unambiguous - Only 1 way to interpret
8) Specify importance
   Req must specify level of importance
9) Verifiable
   Shall be verifiable by basic methods such as inspection, demonstration, testing or

Project Planning and Processes

- Project consists of a starting point, at least one happening and a goal.
- Balance between goal & process

SMART-Goals

Specific - What will you do? Why is it important?
Measurable - Must be able to measure goals
Agreed upon - Agreed upon with all stakeholders
Realistic - Possible with the current resources knowledge & time
Timely - Clear timeframe for the goals

Dependent project parameters

Most important when setting the goals for the project
1) Calender time - How much time do we need? When can we finish?
2) Resources - What resources do we have? Personnel, knowledge, Comp. pwr
3) Features - What features do we want to implement?
4) Quality - What quality do we require on the finished product?

GANTT-Chart

Tool to visualize the process. Makes sure that the project gets done in time
Consists of:
1) Phases (could be an iteration)
   Includes multiple tasks/activities
2) Tasks/activity
3) Duration
4) Dependencies
   One task is dependent on another task
5) Critical path
   Activities with the most time-sensitive dependencies. If postpone, the complete project fails behind
6) Slack-time - "Buffer"
7) Real time - Estimated time
8) Available time = Slack time + Real time

Green: Milestone
Red: Tollgate

Optimal amount of teammembers = 5-12

No	Risk Description	Prob	Impact	Risk factor
1	website can't talk to DB	2	3	6

Avoid risk: ~~~
Transfer risk: ~~~
Accept risk:
- Mitigate: ~~~
- Contingency plan: ~~~

Effort estimation

1) Delphi
   Experts make individual predictions & show them anon. for discussion
2) COCOMO
   Algorithmic formula where a number of factors are estimated using data from old projects (lines of code).
3) COCOMO II
   Also take changes in SE in account, e.g. component reuse & prototype.
   Other inputs as well (eg mock screens)
4) Planning poker / Agile estimation
   - Hours or points on the cards
   - Fibonacci-series
   - Place card, flip & discuss

Risk management

Type of risks:
1) General risk
   "A team member gets sick"
2) Project specific risk
   "Hardware issues"
3) Direct risk
   Project has great control
   "Our system will not scale"
4) Indirect risk
   Project has little control
   "Server failed due to earthquake"

General steps for risk management

1) Risk identification
   "What can go wrong?" Brainstorm risk
2) Risk analysis
   - Probability: Low/moderate/High/Very high (1-4)
   - Impact: Insignificant, tolerable, serious, catastrophic (1-4)
   - Risk management indicator:
     Probability x impact
3) Risk planning
   - Avoidance: Reorganize so that risk disappear
   - Transfer: Reorganize so someone else take the risk
   - Acceptance:
     - Mitigate: Lower probability
     - Contingency plan: Lower impact
4) Risk monitoring
   "Has the probability changed?"

The Project Plan

- Tool for the project management
- Communication medium between project members and other stakeholders
- What should be done when & by who

Content of Project Plan

1) Project description
   - Background to project
   - Relevant constraints
   - Project goals
   - Start & expected end date
2) Project organization
   - Roles, knowledge/skills, training, cons. & report.
3) Time & Resource Plan
   - Milestones & tollgates
   - Deliverables, activities & resources
4) Risk management
   - Risk, probability & impact
   - Mitigation & contingency plan

Project status report
1) Current status
2) Happened since last report
3) Happens next (long & short term)
4) Problems & risk

Common Roles

1) Project manager
   Ensures plan is being followed & goals reached
2) Product manager
   - Strategic (Product owner/sponsor) - Market comm & analysis, budget responsibility & decides features
   - Operational
     Technical management & expert, effort estimation
3) Configuration manager - Select & maintain tools, decides what's in a release
4) Line Manager - Legal employer, ensure competence developments & good working environment
5) Process Manager - Decide on processes & adherence to these
6) Analysts - Handle everything with requirements, writes SRS
7) Architect - Ensures requirements are met & specify high-level architecture
8) Lead Designer - Handles prototypes & design issues, UX & dialogue etc, and control in architecture designs
9) Environment manager - Creates & maintains environments for development & testing
10) Developer - Develops the system
11) Procurement responsible - Buy components & licenses
12) Component adaptor - Adapt external or reuse components to a system
13) Integrator - Puts the pieces of software together to a system
14) Tester - Tests and evaluates the requirements
15) Quality coordinator - Measure quality & organizes reviews
16) Deployment manager - Ensures that the product is installed and made available
17) Technical writer - Responsible for documentation
18) Course developer & leader - Creates & performs training material & courses
19) Helpdesk - Helps with & documents customer issues
20) Operations manager - Ensures that services are provided to the customer
21) Systems Engineers - Performs maintenance & monitoring on active systems

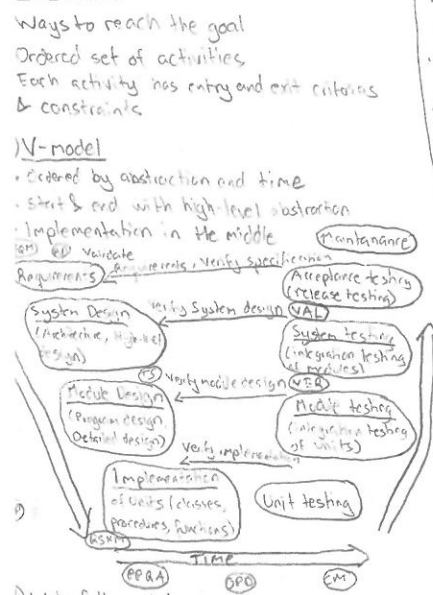
forts. Project Planning & Processes

Processes

Ways to reach the goal
Ordered set of activities
Each activity has entry and exit criterias & constraints

1) V-model

- Ordered by abstraction and time
- start & end with high-level abstraction
- Implementation in the middle



2) Waterfall-model

Remove abstraction (Y-axis) from V-model and we have the Waterfall-model
Originate from manufacturing & construction industry
Very common, very critiqued
Finish each phase before next starts
Milestones & deliverables at each step

Advantages

- simple & easy to understand
- suitable for short projects (some weeks)
- Can be used at large system levels (several years)
- Suitable for stable projects, req. don't change
- Suitable for fixed-price projects

Disadvantages

- Software requirement change
- Early commitment, Change at end ⇒ Large impact
- Difficult to estimate time & cost of phases
- Risk not part of model ⇒ Pushes risk forward
- Little room for problem solving

3) Iterative Model

- Royce 1970
- Utilize either Waterfall or V-model & apply iterations to it
- Fix time & resources and revisit features & quality
  Negotiate
- Incremental model
  when you add features to iterations to reach the goal

Advantages

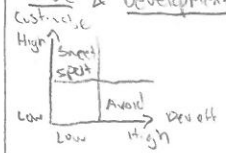
- Flexibility and spread out workload
- Continuously improve process
- Misunderstandings are made clear early

Disadvantages

- Mapping req. to different iterations
- Prioritizing features.

- Time-boxing
  Time period is fixed for each iteration

- Prioritization of Requirement
  Comparison between customer value & Development effort



4) Rational Unified Process (RUP)

- Visually describe workload of different areas.
- Has 4 main Phases
  1) Inception - Req Spec, Prototyping
  2) Elaboration - Architecture, project plan
  3) Construction - Development & testing
  4) Transition - Delivery to customer

Agile development methods

- Early and continues delivery of valuable software

- Manifesto
  1) Individuals & interaction over processes & tools
  2) Working software over comprehensive documentation
  3) customer collaboration over contract negotiation
  4) Responding to change over following a plan

1) eXtreme Programming (XP)

- for small/medium-sized teams
- used when vague & changing requirements

Values

1) communication
   on-site customer, user-stories, Pair programming, daily standup meetings
2) Simplicity
   "Do the simplest thing that could possibly work"
3) Feedback
   - Unit tests tells programmers status of system
   - New releases every 2-3 weeks for customer to review
4) Courage
   - Accept feedback
   - Throw away code
   - Refactor architecture

Practicies

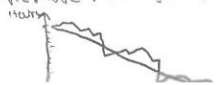
1) Pair programming
2) Continuous integration
3) Refactoring
4) Stories
5) Test-first programming

Disadvantages

- Some claim that it's to unstable
- Lack design & documentation

Burn Down Charts

- Showing remaining work in the sprint backlog
- Updated every day
- X: Days in sprint
- Y: Remaining effort in hours
- Remove meeting time & vacation



2) SCRUM

- Small, cross-functional teams
- Product split into small, roughly estimated stories.
- Work in iterations (sprints)

SCRUM-master

- Ensures that scrum is followed
- Removing things that is between the team and their goal.
- Coach the team

Product owner

- Represent product's stakeholders & customers
- in charge of that the team delivers business value
- Defines user-stories & adds them to backlog & prioritize

Product Backlog

- Prioritized list of what is required, features & stories.
- contains bug-fixes, features, non-func req etc.

Sprint Planning Meeting

- Beginning of each sprint
- Discuss & agree on scope of sprint
- Select product backlog items
- Prepare sprint backlog

Sprint backlog

- Selected during sprint meeting.
- What shall be addressed & done during next sprint
- Task board is often used.

Daily SCRUM-meetings

- Same time and place every day
- Questions to answer
  1) what did I complete yesterday?
  2) What will I complete today?
  3) Any problems that can stand in the way of team meeting our sprint goals?

Sprint review meeting

- Review work done and planned work not done
- Present complete work to stakeholders
- Team + stakeholders collaborate

Sprint retrospective

- Reflect on past sprint
- Discuss & agree on improvements
- Two main questions:
  1) What went well during sprint?
  2) What could be improved in next sprint?

3) Kanban

- Lean approach to agile software development
- Focus on flow of progress

How it works

1) Split work into pieces
2) Write each item on card & put it on the wall
3) Use named columns to illustrate where in the workflow they are

- Limit WIP (Work in progress)
  - Limit number of items in each workflow state
  - Limit designed to
    - Reduce multitasking
    - Maximize throughout
    - Enhance teamwork

- 20% time lost to context switching per task
- Perform sequentially yields results sooner

Typical Measurements

1) Cycle time
   Measured from when you started working on it
2) Lead time
   Measured from when customer ordered
3) Quality
   Time spent fixing bugs per iteration
4) WIP
   Average number of "stories" in progress
5) Throughput
   Number of "stories" completed per iteration

Benefits of Kanban

- Eliminates over-production
- Produce only what is ordered
- Flexible to meet customer demand
- Several things are optional:
  sprints estimations agile practices

4) Lean Software Development

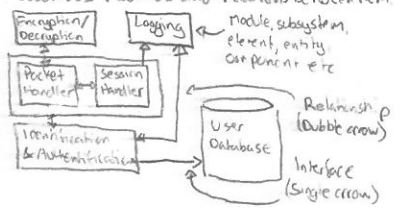
- Adaption of lean principles for Software dev.
- Favoured by start-ups. trying to
  - Penetrate markets
  - Eliminate waste
  - Decide as late as possible
  - Empower the team

Design & Architecture

- Start of with:
  - Comms with stakeholders
  - Early design decisions
  - Try to find similar systems & components to reuse
- Designing is an iterative process

Box-and-Line diagram

- Prototyping often starts with Box-and-line Diagram
- Describes modules and relations between them



Architectural views

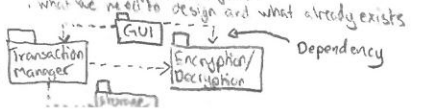
- The step after Box-and-Line Diagrams.
- More detailed
- Often done in UML
- Three main views
  1) Implementation view
     show packages, components, artifacts
  2) Execution view
     show components, connectors & subsystems
  3) Deployment view
     show physical machines

Unified Modeling Language (UML)

- Standard for modeling systems & software
- It's divided into different diagrams:
  1) Structure diagrams
     - Package diagram
     - Component diagram
     - Deployment diagram
  2) Behavior diagrams
     - Use case diagram
     - State machine diagram
     - Sequence diagram

1) Package diagram

- Packages to support the system
- What we need to design and what already exists



Tests, Testing & SCM

System testing

4 types:

1) Function testing
2) Performance testing
3) Acceptance testing
4) Installation testing

1) Function testing

- Test functional requirements
- Tests one function at a time
- Useful to have an independent testteam that knows the input/output
- Test both valid & invalid inputs
- Have stop criteria

2) Performance testing

- Test non-functional requirements
- Stress/timing/volume/configuration/compatibility/regression/security/physical environment/quality/recovery/maintenance/documentation/usability
- Often done by gathering statistical data

3) Acceptance testing

- Test that the users/customers needs are fulfilled
- Consists of
  - Benchmark tests
    specially designed tests
  - Pilot tests (in everyday work)
    - Alpha test - At dev site
    - Beta test - At customers site
  - Parallel testing
    - Test new system in parallel with old

4) Installation testing

Same as Pilot beta testing

Smoke test

- Important selected tests on module/system
- Possible to run fast
- Run to make sure you are on the right way

Logical faults in system

1) Algorithmic - Divide by 0
2) Loop potential & precision - Order of op
3) Documentation - Doc -> code
4) Stress/overload - Data structure size
5) Capacity/bounding - x devices, y parallel tasks, z interrupts
6) Timing/synchronization - Real time systems
7) Throughput/performance - Speed in req
8) Recovery - Power failure
9) Hardware & system software - Modem
10) Standards & procedures - Org. standards

Fault classified by severity

1) Mild
2) Moderate
3) Annoying
4) Disturbing
5) Serious
6) Very serious
7) Extreme
8) Intolerable
9) Catastrophic

- **Full path coverage**
  All possible paths executed
- **Termination problem**
  When to stop?
  - Deadlines
  - Certain percentage pass test
  - Test budget depleted
  - Coverage of code/functionality

Software Configuration Management (SCM)

- System that keep track of all the changes in the development process
- Only authorized ppl can make changes

Change control board (CCB)

- If large changes they make change decisions

Continuous Integration (CI)

- Integrate changes frequently
- Important practices:
  - Automated build of system
  - Automated test of system (unit, system, regression, smoke)
  - Automated integration (input tags in SCM etc)
  - The use of an SCM

Revision control

System that manage changes of files, code, documents etc

Two common revision control systems:

1) Git
2) SVN

Software Quality

- **Quality factors**
  See other sheet

Software Review

- Systematic peer examination of software products
- NOT testing: 1) Inspection
- What can be inspected?
  - Req spec
  - Architecture description
  - Design documents
  - Source code
  - Dev. process description
  - System build procedure
  - Test documents
  - Release notes
  - Installation procedure
  - Maintenance manuals

Participants in inspection? (Roles)

1) **Leader** (moderator)
   - Planning & organizing the inspection
   - Ensures inspection data is collected
2) **Recorder** (can be leader)
   Documents defects, results & recommendations
3) **Reader**
   Informs the software product to be tested
4) **Author** (not Leader, recorder or reader)
   Perform rework to meet criteria

- **Inspection process**

  6 parts:

  1) Input for entry
  2) Plan & overview
  3) Individual checking
  4) Inspection meeting
  5) Edit & follow up
  6) Collected data

1) **Input** (Author)
   - Objective statement
   - Software product/artifacts to be inspected
   - Inspection procedures
   - Reporting forms
   - Known defects
   - Source documents (SRS, descriptions etc)

2) **Planning & overview** (leader)
   - Identify inspection team
   - Assign responsibilities
   - Schedule meetings
   - Distribute material
   - Specify scope & priorities
   - Introduce the product (Author)

3) **Individual checking** (Inspectors)
   - Exam product individually
   - Report defects to inspector leader
   - Prepare for inspection meeting
   - Inspection rate
     - 2-3 pages/hour
     - 100-200 lines of code/hour

4) **Inspection meeting**
   - Leader, reader, recorder, inspectors
   - Agenda
     - Reader present product
     - Inspect product, produce defect list
     - Review defect list
     - Make exit decision:
       1) Accept no further verification
       2) Accept with rework verification
       3) Reinspect, redo the process

5) **FAI & followup**
   - Author resolves items
   - Inspection leader verifies that all items are closed

6) **Collected data**
   - Software product identification
   - Date & time of inspection
   - Inspection team
   - Inspection time (meeting & individuals)
   - Volume of inspected material

Defected data items

- Classification
  Logic problem, data access problem etc
- Categories
  Missing, extra, incorrect etc
- Ranking

Other software reviews

1) **Management reviews**
   - Check deviation from plan
   - Products are plans & reports
   - Performed by management staff

2) **Technical reviews**
   - Evaluate conformance to specifications & standards
   - Performed by tech-leadership
   - High volume of material

3) **Walk-through**
   - Informal atmosphere where author present, lead & controls the discussion

4) **Audit**
   - 3rd party independant evaluation of conformance to specification & standards

Software Metrics

- Purpose is to measure the Quality factors
- Most commonly measurements:
  - Document, no. of pages
  - Design, no of model elements
  - Code, no of lines
  - Process, iteration length
  - Quality, avg no of hours to learn system

Metrics is a combination of measurements!

- Measurement of some Quality factors

1) **Reliability**
   Probability that software executes with no failures during a specific time interval
   MTTF: Mean time to failure
   Reliability = MTTF/(1+MTTF)
   or
   Failure intensity = Failures/hours of execution time

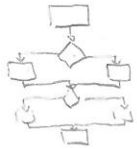
2) **Availability**
   MTTR = Mean time to repair
   MTTF = Mean time to failure
   $$\text{Availability} = \frac{MTTF}{(MTTF+MTTR)}$$

3) **Maintainability**
   $$\text{Maintainability} = \frac{1}{(1+MTTR)}$$

Cyclomatic Complexity

$v(G) = E - N + 2 \cdot P$

E = No of edges
N = No of nodes
P = No of disconnected parts



E = 9
N = 8
P = 1
v(G) = 3

E = 11
N = 7
P = 1
v(G) = 5

Specific Software Metrics

1) **Usage based metrics** (Relevance)
   - Description: No of good and bad features recalled by users
   - Obtain data: Set up test scenario let users run scenario, record no of good & bad features
   - Calculate metrics: Average no. good & bad features

2) **Verification & Validation metrics** (Safety)
   - Description: Rate of some defects found in inspection
   - Obtain data: Run inspection
   - Calculate metrics: Divide number of defects with highest severity with total no. of defects

3) **Volume metrics** (Reliability)
   - Description: No of non-commented lines of code
   - Obtain data: Count no. of non-commented lines of code
   - Calculate metrics: See above

4) **Structural metrics** (Understandability)
   - Description: Maximum depth of inheritance tree
   - Obtain data: Count depth for all classes
   - Calculate metrics: Maximum value found

5) **Effort metrics** (Testability)
   - Description: Time spent in testing
   - Obtain data: Make sure that testing are distinguished in time
   - Calculate metrics: Sum no. of hours for all activities & all ppl

Process areas of CMMI

1) **Requirement Management** (REQM)
   - Level 2 - Requirement
   - Purpose: Manage requirements, ensure enlignment to project plan
   - Requirement

2) **Requirements development** (RD)
   - Level 3 - Requirement
   - Purpose: Elicit, analyse and establish customer, product & product component requirements

3) **Technical Solution** (TS)
   - Level 3 - Design
   - Purpose: Select design & implement solutions to requirements

4) **Project Planning** (PP)
   - Level 2 (Project Management)
   - Purpose: Establish & maintain plans that defines project activities

8) **Configuration Management** (CM)
   - Level 2 - Process
   - Purpose: Establish & maintain integrity of work products using configuration control

9) **Verification** (VER)
   - Level 3 - Testing
   - Purpose: Ensure that selected work products meets their specific requirements

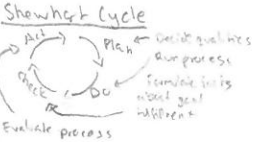
10) **Validation** (VAL)
    - Level 3 - Testing
    - Purpose: Demonstrate that a product or product component fulfills its intended use when placed in its intended environment

Software Quality Assurance (SQA)

- Views of Quality
  1) Transcendent - Something we learn to recognize
  2) Product-based - Measurable value
  3) Usage-based - In the eyes of the beholder
  4) Manufacturing-based - Conformance to requirement
  5) Value-based - Market sets the value

  If many opinions => statistical techniques

- Shewhart cycle



- Levels of quality assurance
  1) Appraisal - Defect detection
  2) Assurance - Prediction of defects
  3) Control - Adjust the process
  4) Improvements - Reduce variation, improve prediction

- Mature Organizations (Weinberg)

  They have:
  - Inter-group commus & coordination
  - Work accomplished according to plan
  - Practice consistent with processes
  - Processes updated as necessary
  - Well defined roles/responsibilities
  - Management formally commits

  Mature orgz do things well, not necessarily good

5) **Risk Management** (RSKM)
   - Level 3 - Project Management
   - Purpose: Identify potential problems before they occur so risk handling can be planned

6) **Process & Product Quality Assurance** (PPQA)
   - Level 3 - SQA
   - Purpose: Provide staff & management with objective insights into processes & associated work products

7) **Organizational Process Definition** (OPD)
   - Level 3 - Process
   - Purpose: Establish & maintain a usable set of org process assets, guides & standards, rules

Capability Maturity Model Integration (CMI

- Used to define maturity level of an organization
- Has 5 stages (steps/levels)

1) **Initial**
   - Over-committed, processes abandoned in crises, no repetition of success
   - Success depends on heroes

2) **Repeatable**
   - Fewer surprises
   - Process based on org policies
   - Process adherence is evaluated
   - Process are established and followed even in crises
   - Project ensure adequate resources (experienced people)
   - We know stakeholders' needs
   - We can control changes
   - Project visible in milestones/stagegate
   - We can repeat previous success
   - Works well for individual projects

3) **Defined**
   - Tailoring processes from your own standard definitions
   - Standard processes are improved
   - Process descriptions are more complete & detailed
   - Teams for development & alternatives
   - Works for a range of projects
   - Only really the minimum level

4) **Managed**
   - Qualitative analysis of goals, product & processes
   - Higher predictive capability
   - Frequent measures

5) **Optimizing**
   - Everyone is committed to continuous improvements of processes
   - Innovation & rapid ability to evaluate new technology
   - Outcome of improvements are evaluated at all relevant levels in the organization
   - Challenge: Changing cultural, new markets

ISO 9000-3

- Guidelines which is built on
  - Customer focus, leadership, involvement of people, process approach, system approach to management, continual improvement, factual approach to decision-making, mutually beneficial supplier relationships

Total Quality Management (TQM)

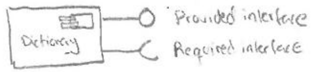
- Management practices focusing on that the customers needs are met/exceeded

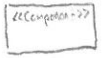
7 principles:

1) Quality can & must be managed
2) Processes, not people, are the problem
3) Don't treat symptoms, look for the cause
4) Every employee is responsible for quality
5) Quality must be measurable
6) Quality improvements must be continuous

forts Design & Architecture

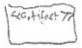
2) Component diagram with interfaces



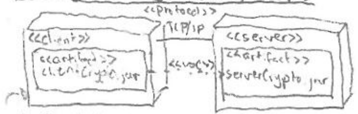


Alternative notation

- Port - Represented by a square in the boundry of a subsystem. connects the components inner part with the outside interface
- Artifact - <<artifact>> Physical code, file or library
- Manifest - Implement



3) Deployment diagram (view) in UML



node: Physical hardware

Coupling - Dependency between modules

- Uncoupled - No dependencies
- Loosly coupled - few dependencies
- Highly coupled - Many dependencies

We want low coupling. because
- Replaceable, enable changes
- Testable, isolate faults
- Understandable

Cohesion - Relation between internal parts of the module

- Low Cohesion - Function less or no incommon
- Medium cohesion - some logically related functions
- Highly cohesion - Does only what it's designed for

We want high cohesion
- Understandable & easier to maintain

Important factors during design & Architecture

1) Performance
2) Security
3) Safety
4) Modifiability
5) Usability
6) Testability
7) Availability
8) Scaleability
9) Portability
10) Maintainability

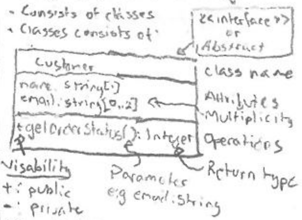
- Performance
  - Timing, throughput, response time
  - Scale up or scale out
- Security
  - Confidentiality: Only authorized ppl can read
  - Availability: Right info at right time
  - Integrity: Only authorized users can modify, edit or delete data
- Safety
  - Absence of critical faults
  - All critical operations in one or few modules or subsystems
- Modifiability - Cost of change
  - What can change?
    - Platform, functions, protocol etc
  - When can change?
- Usability

  Easy to use vs. efficiency

  Word vs. Latex

1) Class diagram

- Most common UML-diagram
- Consists of classes
- Classes consists of:



Visability
- +: public
- -: private
- #: protected
- ~: Package

- Relationships

1) Association A → B

"A" has reference(s) to instance(s) of "B"

2) Composition A ◆→ B

- An instance of "B" is part of an instance of "A"
- "B" is not allowed to be shared
- ◆← Must be 1 or 0..1

3) Generalization A ▷ B

- "A" inherits all properties & operations of "B"
- Instance in "A" can be used when instance in "B" is expected
- "A" can also implement own functions

4) Realization A - - ▷ B

- "A" provides an implementation of the interface specified by "B"
- "A" must implement all functions in "B"

5) Dependency A - - -> B

- "A" is dependent on "B"
- Changes in "B" cause changes of "A"

Behaviour Diagrams

1) Use-case diagram
2) Activity diagram
3) State-machine diagram
4) Sequence diagram

1) Use-case Diagram

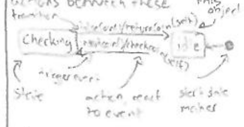
See other sheet

2) Activity diagram

- Represent activity from start to end.
- Round rectangle = Action
- Diamond = Decision
- Bar = Start & end of concurrent activity
- Black circle = start node
- Encircled black circle = End node

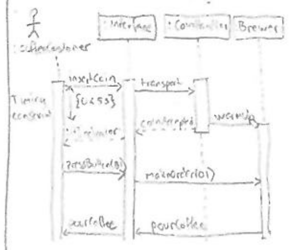
3) State-machine Diagram

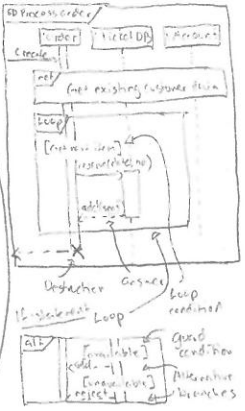
- Similar to activity diagram
- Represent current state and actions between these



4) Sequence diagram

- Interaction diagram that shows the operation of processes with one another over time





Design Patterns

- Goal is to enable reuse of info & componants
- Three common patterns are:
  1) Strategy (Policy)
  2) observer
  3) Façade

1) Strategy (Policy)

- Used when needed to select an algorithm at runtime
- Eg if you are validating input data Different input data demands different algorithms
- Use an interface for this
- Steps
  1) Define a family of algorithms
  2) Encapsulate each one of them
  3) Make them interchangeable

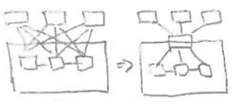
2) Observer

- Mainly used for event-handling systems
- When the main object (subject) gets updated, all its dependencies get informed about the update

3) Façade

- A unified interface to a set of interfaces in a subsystem
- Makes the subsystem easier to use. Reduce complexity
- Decouple the system

Upsides
- + Actors outside the system is not affected by changes in the system
- + Reduce no. of systems client interact with



Architecure styles

- 4 types of styles:
  1) Client-server
  2) Layered
  3) Pipe-and-filter
  4) Service-oriented architecture

1) Client server

- Describes how code & workload should be distributed between client & server

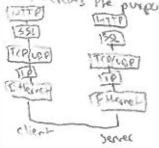
1) Two-tier, thin client
   Client: Presentation layer
   Server: Business layer + Data management
   } Heavy load on server

2) Two-tier, fat Client
   Client: Presentation layer + Business layer
   Server: Data management
   } - Excessive workload on client - System management problems

3) Three tier
   client: Presentation layer
   Middleware: Business layer
   Server: Data management
   } Possibility for load-balancing

2) Layers

- From high to low level interface
- Comms is only done between layers
- Layer bridging is possible though, but defeats the purpose of layering.

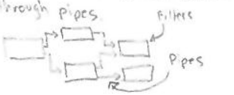


Pros
- Easy reuse of layers
- Support for standardization
- Dependencies kept local

Cons
- Could give performance penalties
- Layer bridging loses modularity

3) Pipes-and-filters

- Decomposing complex processes into filters that performs a process and then sends the data forward through pipes



4) Service-oriented Architecture

- Everything is broken down into services
- Service is a discrete unit of functionality that can be accessed remotely and acted upon and updated independently
- Components communicate via protocols over the network

Documentation of design

- When to document?
  - Initial design
  - Iterations
  - After implementation
- What to document?

1) System overview

Brief description
- who the user is
- main requirements constraints & quality factors
- Important background info

2) Different structural views
- Implementation view
- Execution view
- Deployment view
- Give overview of views
- Create view diagram for all views
- Describe each element
- Describe relations between elements
- Describe interfaces

3) Mapping between views
- Describe how views relates (they all describe same system, what is different?)
- Reduce confusion
- Extra important if not one-to-one

4) Behaviour
- Views state structural information
- Use e.g. text, sequence diagrams, statemachines etc

5) Rationale
- Why a decision was made
- What the implication is to change it

White Box testing

- Find input X and output Y so that Z is executed

Unit testing

Test a single unit of the source code before accepting it into the system

Testing & SCM

- Two central concepts
  1) Validation: Are we building the right system?
  2) Verification: Are we building the system right?
- Testing is an activity in which a program is executed under specific conditions, the results are observed & evaluated
- Methods for Validation & Verification:
  1) Formal verification (Z method)
  2) Model checking
  3) Prototyping
  4) Simulation
  5) Software review

Error: People make errors

Fault: Result of an error

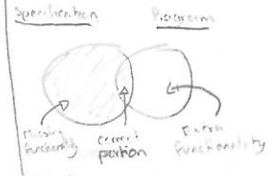
Failure: When faults execute

Fault of commission

Enter something into a representation that is incorrect

Fault of omission

Fail to enter correct information



Black Box testing

Given input X, software shall give me output Y

- Includes 3 types of testing:
  1) Exhaustive testing
  2) Equivalence class testing
  3) Boundary value analysis

1) Exhaustive testing
- Test with all possible inputs to the program

2) Equivalence class testing
- Test all equivalent sets of data
- Eg: At least 18 years old to burrow money, but more than 10000 and less than 100000

EC1: age < 18

EC2: age >= 18

EC3: sum < 10000

EC4: 10000 <= sum ≤ 100000

EC5: 100000 > sum

3) Boundary value analysis

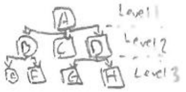
Focus on boundaries between equivalence classes
- Test one point on boundary, one above and one below

Integration test

- When integrating single componants into an integrated module
- 4 main types:
  1) Big bang
  2) Bottom-up
  3) Top-down
  4) Sandwich

1) Big bang
- Integrates all components in a big bang. Thereafter test complete module.
- Difficult to isolate faults
- Quick

2) Bottom-up



- Test from bottom-up (EFB) & (GHD)
- Driver

  Component that mimics higher level functionality
- Possible that it takes until the end to catch big interface defects
- Easy to use
- When to use?
  - If Bottom-up development
  - Complicated basic functions
- Problems
  - Complicated user interface testing is postpone
  - End-user feedback is postponed
  - Effort to write drivers

3) Top-down
- Test from top-down (ABCD)
- Stubs
  - Component that mimics lower level functionality (input/output)
  - Easier to write than drivers
- Pros
  - Defect in general design found early
  - Work well with incremental development
  - Testcases designed for functional req
- Cons
  - Technical details postponed
  - Many stubs are required
  - Many stubs with many conditions are hard to write

4) Sandwich
- Combine bottom-up & top-down
- Find testing target level
- (ABCD) (EFB) & (GHD) in parallell
- Good for systems with many layers
- Pros
  - Top-down/bottom-up done in parallel
- Cons
  - Requires more resources