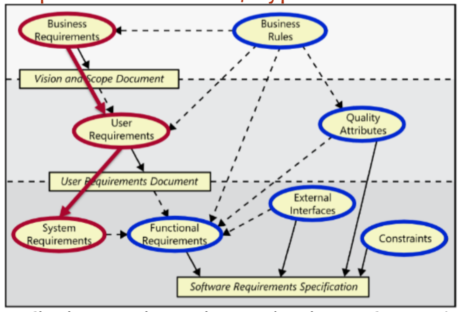
Fundamentals:

Requirements Definition/Importance

Requirement: - Captures purpose of system.
-An expr of ideas embodied in sys. or app under dev.
-a statement about sys. that SHs agree must be made true for customer problem to be solved. (**Short & concise, says something about system, all SHs agreed that is valid, helps solve customer's problem.**) IEEE 29148-2011: **A statement which translates or expresses a need and its associated constraints and conditions.**
Requirements Engineering: the interdisciplinary fn. that mediates between the domains of the acquirer and supplier to establ. and maint. The reqs. to be met by the sys., SW or service of interest. Concerned with discovering, eliciting, developing, analyzing, determining verif. methods, validating, communicating, documenting and managing requirements.
Dealing with Changing Requirements: Requirements baseline: Good enough to proceed to design with an acceptable level of risk. Formally reviewed and agreed on. Rough guide: limit changes to <0.5% per month.
Business requirements: Identify the primary benefits that a new system is expected to provide to its sponsors, buyers and users. **Project scope:** allow new requirements to be vetted in/out.

Requirements Levels / Types:



Airline wants to reduce counter staff costs by 25%

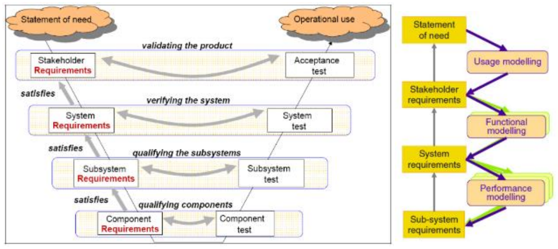
Passengers check in for a flight using an airport kiosk

Kiosk shall print boarding passes upon successful check-in

Application-Domain Requirement / Business Rule: Req. derived from bus. practices within a given indust. sector or in a given comp. or defined by gov. regul. or stand.
User Requirement: Desired goal or fn. that user and other stakeh. expect the system to achieve (doesn't nec. become sys. req.)
System Requirement: Requirement for the whole sys. to be.
Software Requirements are derived from System Requirements.
Goal: objective or concern that guides RE process → not yet req.
Functional Requirement: What the system should do.
Non-functional Requirement: Quality rather than function of sys.
Example Functional Requirement: The user shall be able to search a subset of the initial set of databases.
Example Non-Functional Requirement: The system development process and deliverable documents shall conform to the process and deliverables defined in MIL-STD-498.
Measurable Non-Functional Requirements: Speed – Processed transactions / second, Screen refresh time, **Size** – Kbytes, **Ease of Use** – Training Time, **Reliability** – Mean time to failure, **Robustness** – Time to restart after failure, **Portability** - % of target dependent statements.

Requirements Engineering Process:

V-Shaped Approach / Requirements Sandwich:



Requirements Activities:

Inception: Start process - identify business need, build business case, feasibility assessment.
IEEE 29148-2011: **Requirements Elicitation:** process through which the acquirer and the suppliers of a system discover, review, articulate, understand, and document the requirements on the system and the life cycle processes. **Validation:** *Right system is being built* - confirmation by examination that requirements (individually and as a set) define the right system as intended by the SHs. **Verification:** *Product is being built right* - confirmation by examination that requirements (individually and as a set) are well formed. **Management:** activities that ensure requirements are identified, documented, maintained, communicated and traced throughout the life cycle of a system, product or service. **Software Req. Specification:** structured collection of the requirements (functions, performance, design constraints, and attributes of the software and its external interfaces.)

Inception

Problem Analysis:

Gain better understanding of the problem being solved before development begins. **Outputs: Product Vision & Project Scope.**

Business Requirements:

Business Opportunity: Identify opportunity in market: e.g. Exploit the poor security record of a competing product.
Business Objective and Success Criteria: Business benef of product in quant. / measur. way (how success will be measured, factors impact. success): *e.g. Achieve pos. $ flow on prod. within 6 months.*
Business Risks: Major risks associated with developing or not developing the product (marketplace competition, timing, user acceptance, impl. issues). Can use GRL to model these.
We do this so that everyone is on the same page and to ensure further requirements are aligned with business goals.

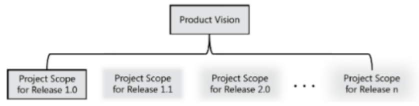
Root Cause Analysis:

Finding underlying causes that may not be immediately apparent, recursive determination of factors contributing to the problem.
e.g. Restaurant Management Application :
The problem of inefficiently running a restaurant including waiting and kitchen staff **affects** customers of a restaurant, **the impact of which is** reduced levels of service and lower food quality. **A successful solution would be** supporting waiting and kitchen staff with up-to-date information on pending orders and food inventory levels leading to improvements in customer experience at the restaurant.

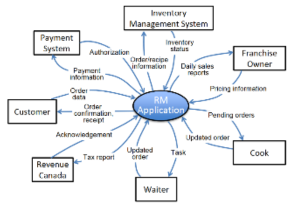
Vision & Scope:

Product Vision: describes what product is about and what it could eventually become, aligning SHs in common direction.
Project Scope: identifies what portion of the ultimate long-term product vision current proj. will address. **Per release: whats in/out.**

Vision


Product Vision
Project Scope for Release 1.0
Project Scope for Release 1.1
Project Scope for Release 2.0
. . .
Project Scope for Release n

Statement (Moore): For scientists [target customer] who need to request containers of chemicals [statement of need or opportunity], the Chemical Tracking System **is** an information system [product category] **that** will provide a single point of access to the chemical stockroom and vendors. The system will store the location of every chemical container within the company. the quantity of material remaining in it and the complete history of each container's location and usage. This system will save the company 25% […] [key benefit] **unlike** current manual ordering process [competition/alternative] **our product** will generate all reports req. [primary differentiation / advantages of product].
Context Diagram:



Elicitation

Goals Risks and Challenges:

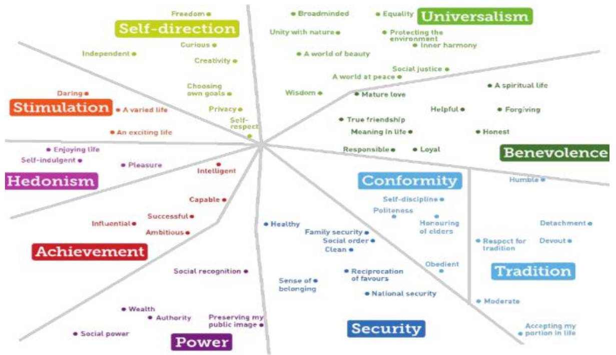
Problems of scope: Sys. bound. inadeq. defined/defined too soon.
Problems of understanding: SH not sure what is needed / has trouble communicating needs / doesn't understand capabilities and limitations of computing environment (reqs. limited by what SHs think is possible / SH does not have full understanding of domain / SHs state conflicting reqs.
Problems of volatility: SHs will not commit to a set of written requirements.

Sources of Requirements:

Various SHs: Clients. customers, users (past/future), buyers, managers, domain experts, developers, marketing and QA people, lawyers, people involved in related systems, anyone who can bring added value.
Pre-existing Systems (not necc. software), Pre-existing Documentation, Competing systems, Documentation of interfacing systems, standards/policies/collective agreements/legislation.

SHs:

Owner/Client/Champion: person paying for the software to be developed.
Customer/Buyer: person who buys software after it is developed (can be same as client)
Domain Expert: expert who knows the problem domain/work.
Software Engineer: expert who knows the technology and process (represents the rest of the development team).
User: of current system and future system
Inspector: expert in government and safety regulations aff project
Market Research Spec.: expert who has studied the market of potential customers.
Lawyer: familiar with laws and legal aspects
Interfacing System Experts, Negative SHs (do not want project to succeed).
Schwartz Theory of Values:



User Classes: Users can be categorized by their differences: security access / tasks performed / features used / frequency used / native language / unsafe (disfavored users) who shouldn't have access / software agents (bots).
Personas:
Fictitious representations of probable users. Vivid, Actionable, Real, Identifiable, Exact, Detailed. Remember: Name, Description of Background, Typical Activities and Frequency, Rank Domain/Technical Knowledge & Experience out of 10, Bullet points for Drivers/Goals/Pain Points.

Tasks Performed:

Planning for the elicitation: Why? Who? When? How? Risks?
During Elicitation: Confirm viab. of project (is it worth it?) Understand the problem from the persp. of each SH. Extract the essence of SHs' reqs Invent better ways to do the work of the user.
Following the elicitation Analyse results to understand obtained information. Negotiate a coherent set of requirements acceptable by all SHs and establish priorities. Record results in the requirements specification.

Elicitation Techniques

Artifact-Based: *Study documentation* - Document studies, similar companies, domain analysis, requirements taxonomies.
Stakeholder-Based: *Get prob. / SH spec. info.* – SH analysis, Questionnaires & Interviews, Observation & Ethnography, Task demo, Ask suppliers, Domain Workshop, Personas.
Model-Based: *Express in different language* – Modeling, Scenarios & Stories, Analysis Patterns, Mockups & Prototyping, Pilot Experiments
Creativity-Based: *Invent undreamed-of reqs.* – Brainstorming, Design thinking, Creativity workshop, Constrain relaxation.
Data-Based: *Use data:* Usage data analysis.

Interviews

Targeted, SH-specific questions. Phrase questions as **open questions.** Needs **Good Listening Skills** (listen to what SH actually saying / give them time to talk). **Interview as many SH as possible.** Ask **problem-oriented** questions but don't reference specific solutions (e.g. don't say "Want Word or Excel?" say "Want word processing or computations?"
Objectives: Record info, discover new info, reassure that interviewee has been listened to.
Stages: Planning/Prep →Interview Sess. → Consolid. →Follow Up
Planning/Prep: Set goals, acquire background info (domain and interviewee), prepare questions in advance, **tailor interview** to interviewee,

organize envir. (How will notes be taken?), limit duration and stick to limited duration. **Communicate agenda and list of attendees**
Session: Be polite and respectful, adjust to interviewee (be flexible), interview several people to create synergy, try to detect political aspects.
Consolid.: Revise and complete elicitation notes, identify inconsistencies and address in follow-up interview / email. Keep all diagrams, charts, models created during discussions. Use interviewee's terminology , create glossary if needed. **Thank participants by email, asking 2-3 short clarif. questions.**
Common Interviewing Mistakes: Not interviewing all the right people, asking direct questions too early, interviewing one at a time instead of small groups, assuming stated needs are exactly correct, trying to convince SH you are smart.
Start-Up Questions: Identify customers, goals and benefits. Ask when they need it by – identify constraints. Characterize problem and solution (whats good / bad solution) Ensure interviewee is the right person to answer your questions (is this official response?) Ask indirectly: are you opposed / against system / threatened by it?
Specific Questions: Functional Requirements, Design Constraints (Physical Environment, Interfaces, Standards / Laws), Performance, Usability and Human Factors (training etc.), Security, Reliability and Availability, Maintainability, Precision / Accurac.
Advantages: Rich collection of info / insights, can go in depth, can adjust format based on feedback, allows for follow-up questions, can assess nonverbal clues for opinions, feelings, motives etc.
Disadvantage: Large amount of qualitative data (hard to analyze), Responses can be hard to compare and analyze, relies on skillful interviewer. Ignorance of domain can be a good thing!

Brainstorming
Invent **new way** of doing things – early on project when you have no clue what to do / need to be innovative.
The Storm: Make lots of ideas, **The Calm:** Filter ideas.
Hear ideas from everyone, especially unconventional ideas – encourage creativity.
Scribe Role: Write down all ideas (can also contribute), can ask clarifying questions during storm but not criticize
Moderator / Leader: Can't be scribe. Either **traffic cop,** enforcing rules of order, not more. Or **Agent provocateur**, traffic cop plus more leadership, throwing out wild idea when discussion dies down.
Any stakeholder can participate in brainstorming.
The Storm: Go Crazy, The Calm: Review, Consolidate, Combine Clarify, Improve.
Rank list by priority somehow, choose winning ideas.
Eliminate ideas: combine similar ideas, 100$ per participant to spend on ideas, eliminate ideas that don't meet acceptance criteria, various ranking or scoring methods, threshold voting / campaign speeches.
Threshold Voting: each person allowed to vote up to n times, keep ideas with more than m votes, have multiple rounds with smaller n and m.
Campaign Voting: each person allowed to vote up to j < n times, keep ideas with at least one vote, have people who voted for an idea defend it for next round, multiple rounds with smaller j.
Joint Application Design: more structure / intensive brainstorming approach – three phases: Customization, Session, Synthesis – 6 roles: Session leader, Analyst, Executive Sponsor, User Representatives, Information System representatives, Technical expert on information systems, Specialists, JAD Sessions may last a few days.
Design Thinking:
Empathy Map: Persepctive of one stakeholder -related to persona. **Says, Does, Thinks, Feels**
As-is and To-Be Scenarios:
Steps, Doing, Saying, Feeling
Challenges to Brainstorming: Unnatural clusters of uncomfortable participants, "Groupthink", Superifical response to technical issues, Bias and dominance.

Analysis of Existing Systems:
Useful when building a new improved version of an existing system. Want to keep in mind: "What is used / not used / missing? What works well / doesn't work? How sys. is used (freq. and importance) / was supposed to be used / people would like to use it?"
Because: Users won't like too different new system, we want to take into account real usage patterns / human uses etc., catch obvious possible improvements, find out which legacy features can be left out.
Read available docs: User docs, Development docs (Bug reports, change histories,...) Req. documents, usage stats/ marketing data..
Observation: Observe Specialists "in the wild", shadow important potential users, silent approach OR ask user to explain everything, session videotaping. **Benefit:** Some user tasks are complex, highly skilled users do compelx things w/o having to think about steps, users may be unable to articulate their work so we watch & learn.
Challenge: Observed people can be conscious of the presence of external observers and change their behaviour (Hawthorne effect)
Apprenticeship: Learn the job with a master by observation, asking questions, doing some of the job under supervision.
Ethnography: discover social / human / political factors that impact requirements → take a long time. Anthropological approach.

Questionnaires / Surveys
Benefits: Can reach large no. people (anonymously), cheap, asynchronous, distributed and can be quick to answer.
Challenges: Preparation time, Open-ended vs close?, Avoid centralist / left-choice tendencies, Concrete questions so we gather data but not insights, Ensure no bias in questions, Determining suitable participants to invite, Making sure participants answer everything.
Demography Questions, Attitudinal Questions (Strongly Agree etc.), Supplementary Open Questions, Optional/Alternative Questions, by population, Redundant Questions

Prototyping
Sketch essence of a solution, bait stakeholders into providing new requirements details. (Mock-up or partial implementation of software system).
Evolutive (prototype that evolves as you show SH) vs **Throw-Away** (thrown away after shown)
Paper Prototypes, Screen mock-ups, Interactive, Models, Pilot Systems.
High Fidelity: Prototype that actually works: **Advantages:** provides understanding of functionality, reduce design risk, more precise verdicts about requirements. **Disadvantages:** takes time / more costly, false sense of security.
Low Fidelity: Not operated (static). **Advantages:** Quick to build, cheaper to develop, excellent for interfaces, can engage users before coding begins, encourage creativity. **Disadvantages:** May not cover all aspects of interfaces, not interactive, can seem non-professional in the eyes of some stakeholders.

Requirements Specifications
Clearly and accurately describes each of the essential requirements (functions, performance, design constraints and quality attributes) of the system / software and its external interfaces. Defines scope / boundaries of system/software.

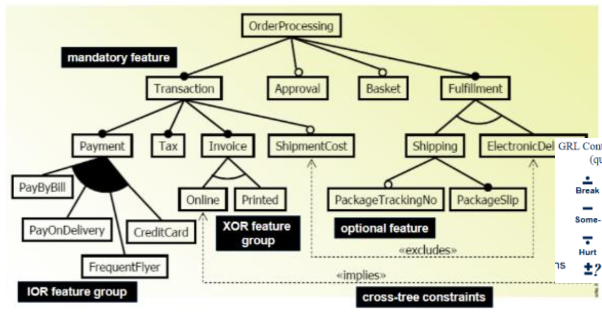
IEEE 830-1998 Standard
Title: "IEEE Recommended Practice for Software Requirements Specifications"
Objectives: Help customers to accurately describe what they wish to obtain. Help software suppliers to understand exactly what the customer wants. Help participants develop a template for software requirements specification in their organizations. Develop additional documents such as SRS quality checklists or an SRS writer's handbook.
Benefits: Establish the basis for **agreement** between the customers and the suppliers on what software product is meant to do. Reduce development effort, by reducing later redesign, recoding, retesting. Realistic estimates of costs and schedules. Basis for validation and verification. Facilitate transfer of software product to new users / machines. Basis for enhancement requests.
Considerations: Nature (goals): Functionality, interfaces, performance, qualities, design constraints. **Environment**: where does it fit in overall project hierarchy. **Characteristics of good SRS**: Generalized characteristics of good requirements to the document. **Evolution:** Implies a change management process. **Prototyping**: Helps elicit software requirements and reach closure of SRS. **Including design and project requirements in SRS**: focus on external behaviour and product (not design / production process).
Contents: Title. Table of Contents. 1 Introduction (1.1 Purpose, 1.2 Scope, 1.3 Definitions. Acronyms and Abbreviations, 1.4 References, 1.5 Overview) **2 Overall Description** (2.1 Product Perspective, 2.2 Product Functions, 2.3 User Characteristics, 2.4 Constraints, 2.5 Assumptions and Dependencies) **3 Specific Requirements** (3.1 External Interfaces, 3.2 Functions, 3.3 Performance Requirements, 3.4 Logical Database Requirements, 3.5 Design Constraints, 3.6 Software System Quality Attributes, 3.7 Object Oriented Models) **4 Appendices 5 Index**

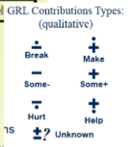
ISO/IEC 12207 (and IEEE 830-1998)
12207 is a common framework for Software Life Cycle Processes
Both 12207 and 830 place requirements on docs specifying software requirements. Annex B of 830 says how to satisfy both at the same time – sometimes customers ask for compliance w/ both.

IEEE/EIA 12207.1-1997 generic content	Corresponding clauses of IEEE Std 830-1998	Additions to requirements of IEEE Std 830-1998
a) Date of issue and status	—	Date of issue and status shall be provided.
b) Scope	5.1.1 Scope	—
c) Issuing organization	—	Issuing organization shall be identified.
d) References	5.1.4 References	—
e) Context	5.1.2 Scope	—
f) Notation for description	4.3 Characteristics of a good SRS	—
g) Body	5. The parts of an SRS	—
h) Summary	5.1.1. Overview	—
i) Glossary	5.1.3 Definitions	—
j) Change history	—	Change history for the SRD shall be provided or referenced.

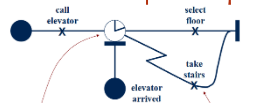
IEEE 29148-2011
ISO/IEC/IEEE 29148:2011: System and software engineering – Life cycle processes – Requirements engineering. Provides unified treatment of process and products involved in engineering requirements throughout the life cycle of systems and software.
More emphasis on characteristics of good requirements, RE activities and process operations (and operation context) / different info. items, such as spec of reqs. for SH, systems and software.

URN Overview
URN is for modeling and analyzing reqs. for elicitation , analysis, specification and validation.
Start with GRL to Model goals, stakeholders' priorities, alt. solutions, indicators, rationale and decisions then make UCM to Model and test use cases, investigate high level architectures which we can eventually transform into more detailed models. Often iterative (i.e. edit UCM → go back to edit GRL → go back to edit UCM → go back to edit GRL) before generating more detailed models.

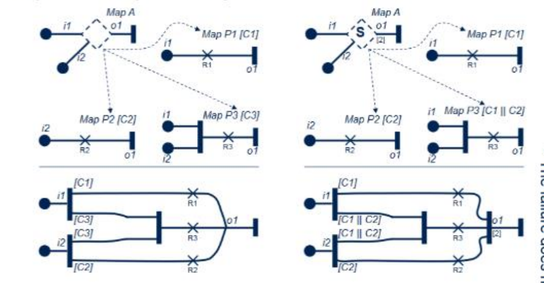
Feature Models:


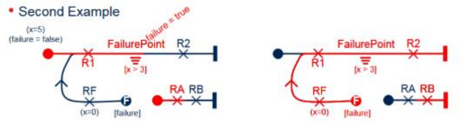


Validity: Do features selected conform to OR / XOR / AND and additionally specified constraints (implied/excludes)?
Completeness: Are all mandatory features selected?
When to use OR / XOR : Think about system as a whole, not a specific use case (i.e. if there's a case where system can do A and another case where system can do B, select OR). **Note however:** if you have a system that switches modes: it's a XOR because you need to model a single mode of the system for each feature model configuration.

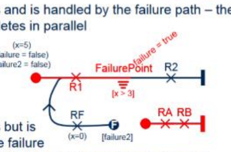
URN
Use Case Map Examples


Wait until elevator arrived. Zig-zag = timeout path.

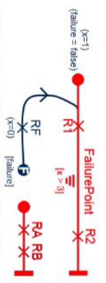
• Dynamic and synchronizing stub:


• Second Example


• The failure occurs and is handled by the failure path – the path with RA and RB completes in parallel

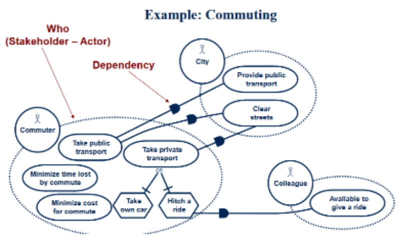
• Third Example


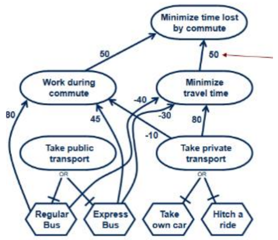
• The failure occurs but is not handled by the failure path – the path with RA and RB completes in parallel



• The failure does not occur and both paths complete in parallel

GRL Examples:





Softgoal: Qualitative, not measurable. **Goal:** Quantifiable

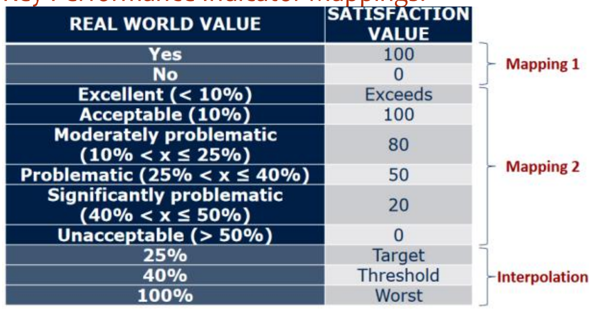
GRL Execution:

Contribution numbers (on arrows) represent percentages. i.e. if regular bus is selected with (numerical evaluation) 100 in above GRL model, it would contribute 80 to Work during commute.
AND-Decomposition: Take minimum of values of child elements.
IOR/XOR-Decomposition: Take maximum of values of child elements.

Exercise Solutions:

Two aspects that characterize a family of products? – Commonality and Variability.
Feature models created or used in domain eng or application eng? – Created in domain eng., used in application eng.
Types of links in Feature Model? Mandatory, Optional, XOR/IOR feature group, includes, excludes.
Configuration? – Selection of features (in/out)
How are feature models represented using URN? Special case of goal model (AND/OR Graph → AND/OR Tree): mand./optional mapped to contributions, XOR/IOR feature groups mapped to decomps, includes/excludes mapped to OCL constraints.
How are configurations expressed with URN? Using strategies: selected feature = 100, not selected = 0).
Combine URN with FM, how do we know that valid config greater than another valid config? Model impact of features on stakeholder goals, see evaluation.

Key Performance Indicator Mappings:



REAL WORLD VALUE	SATISFACTION VALUE
Yes	100
No	0
Excellent (< 10%)	Exceeds
Acceptable (10%)	100
Moderately problematic (10% < x ≤ 25%)	80
Problematic (25% < x ≤ 40%)	50
Significantly problematic (40% < x ≤ 50%)	20
Unacceptable (> 50%)	0
25%	Target
40%	Threshold
100%	Worst

Mapping 1
Mapping 2
Interpolation

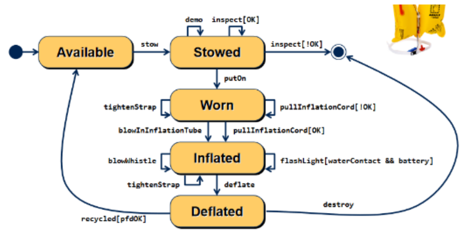
Requirements Modeling with UML

Qualities of a Good Model: Abstract, Understandable, Accurate, Predictive, Inexpensive.
Modelling Formalisms:
Natural Language: + No special training required, - ambiguous, verbose, vague, obscure, - no automation. **Ad hoc (Bubbles and Arrows):** + No special training required, -ambiguous, unclear, - no automation. **Semi-Formal (URN, UML):** +Syntax well defined, +Partial common understanding, pretty easy to learn, + partial automation **Formal:** +Syntax well defined, +Great automation, -More difficult to learn & understand.

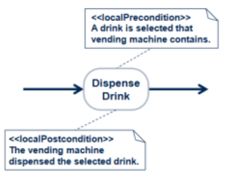
Class Diagrams:

Class Diagrams for Precise Domain Modeling: No operations, Attributes with types, Precise multiplicities on associations, names for association ends must be used! Enumerations are useful, care when generalizing. Specify additional constraints in English or OCL.
But: Requirements don't align perfectly with Object Oriented design (i.e. a single requirement can be satisfied in multiple classes in the domain, single domain class can satisfy many requirements).
Separation of Concerns: Composition to specify crosscutting relationship e.g. if a system requires a record to be stored for multiple different actions – the storage of records **concern** is composed in multiple actions (it crosscuts multiple actions).

State Machine:

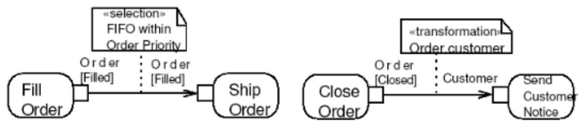


UML Activity Diagram:

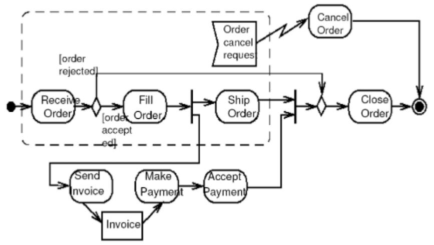

<<localPrecondition>>
A drink is selected that vending machine contains.
Dispense Drink
<<localPostcondition>>
The vending machine dispensed the selected drink.

Terminal node with circle in middle = end whole activity, return to parent (if it exists)
Terminal node with cross in the middle = end flow, continue activity if unfinished parallel flows.

Pins to specify passed objects:



Region Interruption:

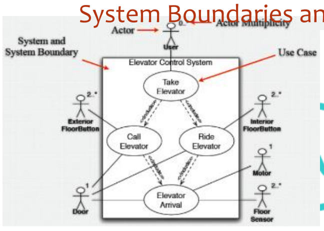


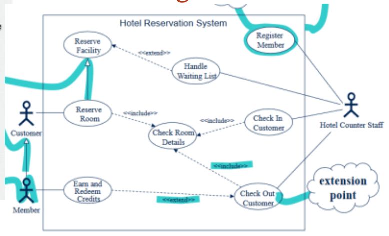
Elicitation Techniques

Use Cases:

Title:, Identifier: ,Actor:, Intention: [the intention of <actor> is to <action>. <additional info>], Precondition:, Main Scenario:, Alternatives:, Postcondition:

System Boundaries and Use Case Diagrams:



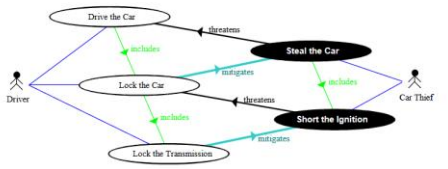


User Stories:

Connextra, 2001: As a <role>, I want <goal/desire> so that <benefit>.
Chris Matts: In order to <receive benefit> as a <role>, I want <goal/desire>
Capital One 2004: As a <role>, I can <action with system> so that <external benefit>
INVEST Characteristics: Independent, Negotiable, Valuable, Estimable (Understandable), Small, Testable.

Group together User Stories with Epic Stories.

Misuse Cases:



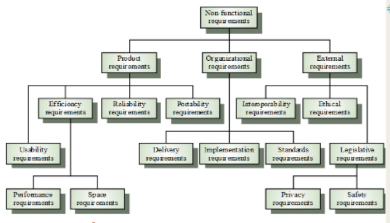
Use Cases for 'Car Security'

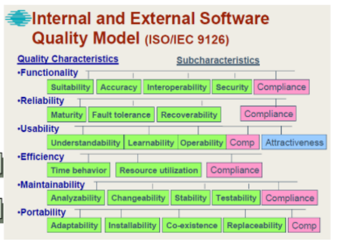
NF Requirements & Quality Measures

Software Quality

"Conformance to **explicitly** stated functional and performance requirements, explicitly documented development standards and **implicit** characteristics that are expected of all professionally developed software"
➔ Need to be able to quantify requirements to verify solution meets them. We need **measures.**
Measurable objectives are usually achieved!

Classification of Non-Functional Requirements




Internal and External Software Quality Model (ISO/IEC 9126)
Quality Characteristics
Subcharacteristics
Functionality
Suitability
Accuracy
Interoperability
Security
Compliance
Reliability
Maturity
Fault tolerance
Recoverability
Compliance
Usability
Understandability
Learnability
Operability
Comp
Attractiveness
Efficiency
Time behavior
Resource utilization
Compliance
Maintainability
Analyzability
Changeability
Stability
Testability
Compliance
Portability
Adaptability
Installability
Co-existence
Replaceability
Comp

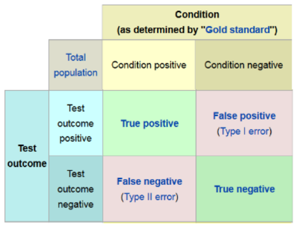
Nowadays, replaced by: ISO/IEC 25010:2011
Adds compatibility and security as main quality characteristics

Quality Measures

Fit criterion quantifies the **extent** to which a quality requirement must be met.
Values must have a rationale, SH must understand tradeoffs.

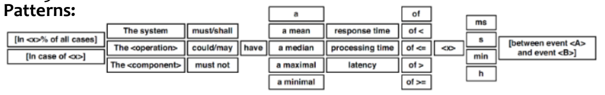
Requirement	Outstanding	Target	Minimum
Resp. Time	0.1s	0.5s	1s
CPU Util.	20%	25%	30%
Usabil.	40 tasks/hr	30 tasks/hr	20 tasks/hr

Precise numbers unlikely to be known at beginning of req. process. **Don't slow down initial elicitation process – ensure quality attributes identified and negotiate values later.**
n.b. Metric: mean time between failures
Measure: 23 days between failures (specific observation)
Confusion Matrix:

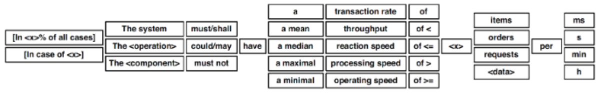


		Condition (as determined by "Gold standard")	
	Total population	Condition positive	Condition negative
Test outcome	Test outcome positive	True positive	False positive (Type I error)
	Test outcome negative	False negative (Type II error)	True negative

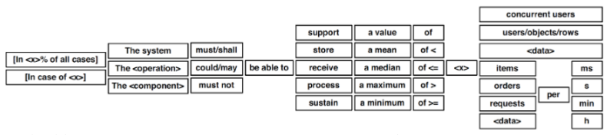
Where Condition Positive / Condition Negative refers to whether the condition is met for the population/system as a whole.
Lots of measures exist; many of them come with probabilities / confidence intervals.
Can model and simulate performance measures (mainly at architectural level) – this is known as performance prediction (Queueing Model, process algebra, stochastic petri nets, arrival rates, distributions of service requests, sensitivity analysis, scalability analysis.
Recall Performance Requirements are a subset of NFR:
e.g. "The system shall be able to process 100 payment transactiosn per second in peak load." "In standard workload, the CPU usage shall be less than 50%" etc.
Patterns:



(a) Sentence Patterns for Time Behavior Requirements



(b) Sentence Patterns for Throughput Requirements

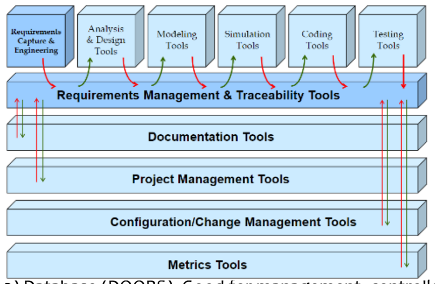


(c) Sentence Patterns for Capacity Requirements

Reliability Measures: degree to which system performs during request. Measured with defect rate / degree of precision for computations. E.g. Precision of calculations shall be at least 1E-6, System defect rate shall be less than 1 failure per 1000 hours of operation.
Availability Measures: percentage of time that the system is up and running correctly / probability system is up when needed.
Mean-time betw. failures (MTBF) /Mean-time to Repair(MTTR):
Availability = MTBF / (MTBF+MTTR) Redundant components in architecture improves MBTF, modifiable components improve MTTR but reduce MTBF.
"The system shall meet or exceed 99.99% uptime" 90% Availability → 36.5 downtime days / year.
Security Measures: Ability to resist unauthorized attempts at usage and continue providing service to legit. Users while under DoS attacks. "The system shall identify a client application before allowing it to use system capabilities."
Usability Measures: concerns ease of use and ease of training.
Learnability: proportion of fn / tasks masters after x train. Time. **Efficiency:** acceptable response time or mouse clicks need to get to information or functionality. **Memorability:** ratio of learned tasks that can still be performed after not using system for given time period. **Error Avoidance:** Number of error per time period & user class. **Error Handling:** Mean time to recover from error and be able to continue tasks. **User satisfaction:** satisfaction ratio per user class.
Maintainability Measures: ability to make changes quickly and cost effectively. Mean time to fix defect or add new functionality.
Testability Measures: ability to detect isolate and fix defects. Time to run tests / setup testing environment probability of visible failure in presence of a defect.
Portability Measures: Measure ability of system to run under different computing environments: Hardware / software / OS etc. e.g. no. targeted platforms "No more than 5% of the system implementation shall be specific to the operating system."
Integrability Measures: Ability to make separated components work together (Mean time to integrate with new interfacing system).
Reusability Measures: Ability that existing components can be reused in new applications (e.g. percentage of reused requirements, design elements, code, tests.
Robustness: Ability to cope with unexpected (percentage of failures on invalid inputs).
Most appropriate quality measures vary from one application domain to another (e.g. difference between web-app and video game).
Monte Carlo Techniques: Estimate unknown quantity using known quantity. Evaluating criteria that cannot be evaluated before final product is delivered. E.g. Plant a number of known (seeded) errors into program which testing team doesn't know about: **no. detected seeded errors / no. seeded errors = no. detected erros / no. erors in program.**

Requirements Management & Traceability

Requirements Volatility: no. of requirements changes in a specified time interval.
Stable requirements are concerned with essence of a system and its application domain. Derived from the client's principal **business activities** or the **domain model.** E.g. a hospital will always have doctors, nurses, patients to deal with…
Volatile Requirements are specific to the instantiation of the system of a particular environment for a particular customer at a particular time. E.g. in a hospital, a nurse may want the system to support the input of specific information about a patient.
Requirements/Feature Creep: Requirements changing toward the end of development without any impact assessment.
Req Management: systematic approach to eliciting, organizing, and documenting the requirements of the system. Requires **traceability** and **baselines** (to compare). **Requirement ID: Dynamic renumbering:** Word processing systems allow for automatic renumbering of paragraphs and the inclusion of cross references. The system keeps track of the cross references and automatically renumbers your requirements depending on its chapter, section, and position within the section. **Database record identification:** When a requirement is identified, it is entered in a requirements database and a database record identifier is assigned which is then used for all subsequent references to the requirement. **Symbolic identification:** Requirements can be identified by giving them a symbolic name which is associated with the requirement itself (e.g., SEC1, SEC2, SEC3… may be used for requirements which relate to system security). **Confidence:** distinguish requirements that are fully certain from the others which can be delayed.
Business intelligence improves decision making, cuts costs, and is used to identify new business opportunities. It leverages data mining, analytical processing, querying, and reporting**. PROJECT intelligence** is based on the data produced – and decisions made, during the Software Development Life Cycle. **Traceability:** ability to describe and follow the life of a requirement from its origins, through its development and specification, to its subsequent deployment and use, and through all periods of refinement
Traceability Factors: 1) Number of requirements 2) Expected system lifetime 3) Maturity level of organization 4) Size of project and team 5) Additional constraints from customer. **Traceability Matrix:** Shows the pairwise connection for each requirement to a goal, use case or requirement.
Benefits: 1) Impact analysis 2) Change control 3) Process monitoring Improved software quality 4) Reengineering) Risk reduction 6) Supports the verification process **Baseline for Requirements:** Represents the set of functional and non-functional requirements that the development team has committed to implement in a specific release.



Req Management Tools: 1) Document (Word) Easy to use, easy to access, simple training. Requirements are all stored in the requirements document. It is easy to produce the final requirements document. Traceability harder.
2) Database (DOORS). Good for management, controlled access, links, analysis, reports. Good query and navigation facilities. Support for change and version management. But: hard, costly to configure, manage, and use; link b/w the database and the req document must be maintained.

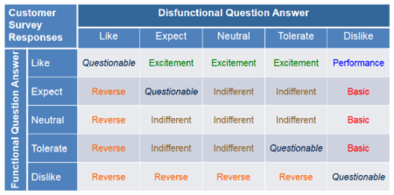
Requirements Negotiation and Prioritization

Requirements Negotiation

Possible **conflicts** to be resolved among stakeholders (related to subjects , interests , values, relationships, structures) or (between supplier an customers about costs, benefits, risks) etc.
Potential Resolution strategies: Agreement, Compromise, Voting, Definition of variants, Overruling, Mediation (involves neutral third party that facilitates), Arbitration (involves neutral third party that judges), Goal Analysis (GRL!). → Conflict Resolution involves **negotiation.**
Let Project Schedule Drive Requirements: You know you have X months, negotiate which requirements should be satisfied in those months.

Requirements Triage and Prioritization

Too many requirements to be implemented – prioritize / filter them! But different SH have different/conflicting goals and priorities and their priorities may change with time! Determine what is needed (**triage**) for **minimum viable product.** Make **acceptable** tradeoffs among benefits, cost, time-to-market.
80-20 Rule: 20% of functionalities provide 80% of revenues.
Prioritization based on Cost and Value: where **Value** is req's potential contribution to customer satisfaction and **Cost** is the cost of implementing req. **Prioritize based on cost-value ratios.** But its **hard to calculate absolute value/cost, relative value/cost figures are easier to obtain, interdependent requirements difficult to treat individually, inconsistencies or conflicts in priorities assigned by individual stakeholders.**
Priority Ranking: Each requirement is assigned a **unique rank**. Its not possible to see the relative difference between ranked reqs
100-Dollar Test (Cumulative Voting): Distribute 100 dollars to distribute among reqs. Reqs with most points win.
Prioritization Scales: Determine criteria, granularity, scale dimensions (**group requirements**). Often: **Urgency –** (High, Medium, Low), **Importance –** (Essential, Conditional, Optional).
Kano Surveys; Group requirements based on customer perception. (Basic – reqs customer takes for granted, Performance – reqs customer specifically asked for, Excitement – reqs that the customer does not request or expect, Indifferent – reqs customer doesn't care about, Reverse – reqs the customer hates. **Ask customers reaction if requirements were included (functional question) and were not included (disfunctional questions).** Possible answers**: I like, I expect, I am neutral, I can tolerate, I dislike.**

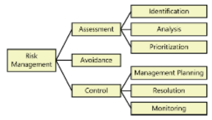


Customer Survey Responses		Disfunctional Question Answer				
		Like	Expect	Neutral	Tolerate	Dislike
Functional Question Answer	Like	*Questionable*	Excitement	Excitement	Excitement	Performance
	Expect	Reverse	*Questionable*	Indifferent	Indifferent	Basic
	Neutral	Reverse	Indifferent	Indifferent	Indifferent	Basic
	Tolerate	Reverse	Indifferent	Indifferent	*Questionable*	Basic
	Dislike	Reverse	Reverse	Reverse	Reverse	*Questionable*

Wiegers' Prioritization: Relies on estimation of **relative priorities of reqs.** (Relative benefit, Relative penalty to SH if not incl., Relative cost to impl, Relative risk) – Each of these dimensions given a value on a given scale (0..9) Priority. Each dimension given a weight. **Priority = (value %) / ((cost% * cost weight) + (risk% * risk weight)).** Where % is % of total of all reqs and value = benefit weight * benefit ranking + penalty weight * penalty tanking
Volere Prioritization: Editable Excel Document, similar to Wieger's.
Analytical Hierarchy Process: SH's pairwise comparisons of requirements → relative ranking of requirements. Use **cost-value** diagrams. For (x,y) pair: 1 – Equal Value, 3 – x slightly preferred, 5 – x strongly preferred, 7 – x very strongly preferred, 9 – extremely preferred over the other. Use intermediate values (2,4,6,8) when compromise needed. (x,y) has value n → (y,x) has value 1/n. **Steps:** 1. Divide each entry by sum of its column, 2. Sum each row, 3. Divide each sum by sum of sums, 4. Report relative values (%). Step 3 gives priority vector.
Further steps: 1. Multiply comparison matrix by priority vector, 2. Divide each element by corresponding element in priority vector, 3. Compute average of results (gives **principle eigenvalue**), 4. Calculate consistency index = (principle eigenvalue – n) / (n-1), 5. Compare against consistency index of random matrix (<0.10) CR = result from 4. / 1 – index of random matrix.
Tools: TeamBLUE Focal Point – Pairwise comparisons of **features,** Integration with DOORS

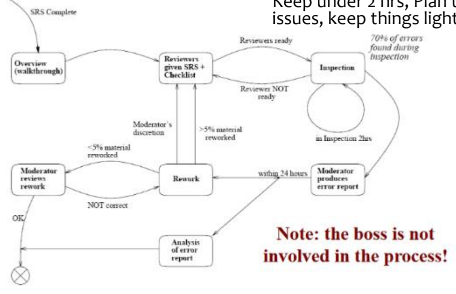
Risk Management

ISO 31000:2009: No longer "chance or probability of loss – now risk refers to positive or negative possibilities". Identify / characterize **uncertainty** (threats / opportunities). How susceptible are critical assets to threats / opportunities. Determine risks (potential damage / gain & likelihood of occurrence). Identify ways to **mitigate** negative risks.
Risk Exposure: RiEx = prob(occurrence) * cost(consequences)



Requirements Verification and Validation

See previous def for Verification and Validation.
Requirements Analysis: focused on making Software Requirements Specification document – **"We have the right requirements"**
Requirements V&V: After requirements specified in SRS doc, we check these specs are accurate – **"We have the right requirements well done"**
Simple Checks: Various checks done using **traceability techniques.** Given requirements doc, verify that all elicitation notes are covered adequately. Tracing between different levels of requirements, checking goals against tasks, features, requirements. Completeness and consistency checks.
Develop traceability matrix. Verify that the requirements are **well written.**
Prototyping: Excellent for **validation** by users and customers. Important to choose right scenarios or use cases for elicitation session.
Functional Test Design: Tests should be **derived** from the requirements specifications. Designing these tests **may reveal erros** in he specification (even before designing and building the system).
Formal V&V: Simulation: random or interactive, **Testing:** predetermined traces have to be accepted (or rejected), **Completeness Checking:** According to certain syntax rules, **Consistency Checking:** Given Model M, show that M doesn't imply a contradiction / any other undesirable general property (deadlocks, unhandled exceptions etc.), **Refine checking:** given two models M and M', show that the properties of M imply the properties of M', **Model checking:** given a model M and some properties P, show that any system implementation satisfying M will have the properties P, **Theorem proving:** prove the correctness and validity of theorems against a model, for all situations.
Reviews and Inspections: Reading and approval (sign-off): Encourages the reader to be more careful (and responsible!), **Walkthroughs:** can be led by author/expert to educate others on his/her work. Audience reviews work based on author's presentation, **Active Review** Author asks reviewer questions which can only be answered with the help of the document to be reviewed. **Formal Inspections:** Structured and detailed review with defined roles, preparation etc.
Fagan Inspection: Not more than 2 hours at a time to keep participants focues. 3-5 participants. Author presents document but is silent otherwise. Metrics are collected **authors supervisor doesn't participate or access data.** Moderator responsible for initiating inspection, leading meeting and ensuring issues found are fixed. All reviewers need to prepare themselves using checklists. Issues are recorded in special forms. **Do:** Critique artifact, Keep under 2 hrs, Plan time for reviews, prioritize review of important issues, keep things light **.Don't:** Attack the person, go on for too long without adequate breaks, skip reviews, FIFO queue reviews, use sarcasm or exaggeration.



Typical Problem Categorization: Requirements clarification - req badly expressed or have accidentally omitted information which has been collected during elicitation, **missing information –** information missing from requirements document.
Req conflict – there is a significant conflict between requirements, SH's involved need to negotiate to resolve conflict, **Unrealistic Requirement –** The requirement does not appear to be implementable w/ technology available / sys. constr.