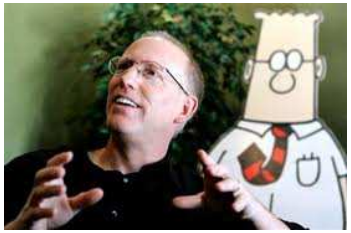
Eliciting Requirements
Specifying Requirements

Nicolas Sannier
* EDF R&D – STEP, 6 Quai Watier BP49
78401 Chatou, France
nicolas.sannier@edf.fr
** Inria, Campus Universitaire de Beaulieu,
35042, Rennes Cedex, France
nicolas.sannier@inria.fr

Where does this lecture come from?

- Make of work, experiences, slides, stuff borrowed from / inspired by / …
  - Scott Adams
  - Daniel Amyot
  - Benoit Baudry
  - Steve Easterbrook
  - Daniel Lucas-Hirtz
  - Martin Mahaux
  - Alistair Mavin
  - Gunter Mussbacher








  - And so many others

Summary from Last Session

- Key terms about RE
  - Is problem oriented and NOT solution oriented
    - Is about building the right product and not building it right (Software Engineering)
  - Is about both the system and its environment
    - Is hard
    - Is crucial
    - Needs social engineering and domain knowledge alignment
  - Requirements are of different nature, abstractions, concerns
    - Functional, non-functional, users, business, application domain, goals, etc.
  - Is everywhere in a lifecycle!
    - You shall not escape!
  - Is a set of Activities (Elicitation, Specification, Analysis, Validation, Management)
    - And concerns: elicitating, writing, modeling, tracing, managing variability and changes, etc.

What is the program for today?

- Requirements Elicitation
  - You said Requirements Elicitation ?
  - Goals and Challenges
  - Risks
  - Sources
  - Stakeholders
  - Elicitation tasks and techniques
- Requirements Specifications - Writing Better Requirements
  - Can’t write a better requirement because
  - Natural Language Requirements
  - Standards, Tips and Pitfalls toward better Requirements Writing
  - Writing Requirements using EARS Templates
- Requirements Specifications - Writing Requirements Documents
  - IEEE 830 Standard for Software Requirements Specifications
- Some Training Exercises

Introduction

- Elicitating things
- *I know that you believe that you understood what you think I said,*
  but I am not sure you realize that what you heard is not what I meant.
  Robert McCloskey, State Department spokesman (attributed)

- Writing clear stuff
- *I don't know half of you half as well as I should like, and I like less than half of you half as well as you deserve.*
  Bilbo Baggins, The fellowship of the Ring

REQUIREMENTS ELICITATION

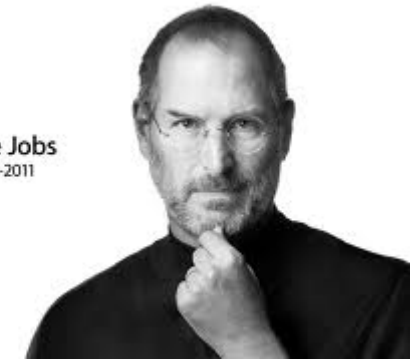
You said Requirements Elicitation?

- Requirements elicitation is “the process of discovering the requirements for a system by communicating with customers, system users and others who have a stake in the system development”

- Elicitation means “to bring out, to evoke, to call forth”
- Elicitation might even require one to provoke!
- Elicitation is not a spontaneous phenomenon

- Human activity involving interaction between a diverse array of human beings

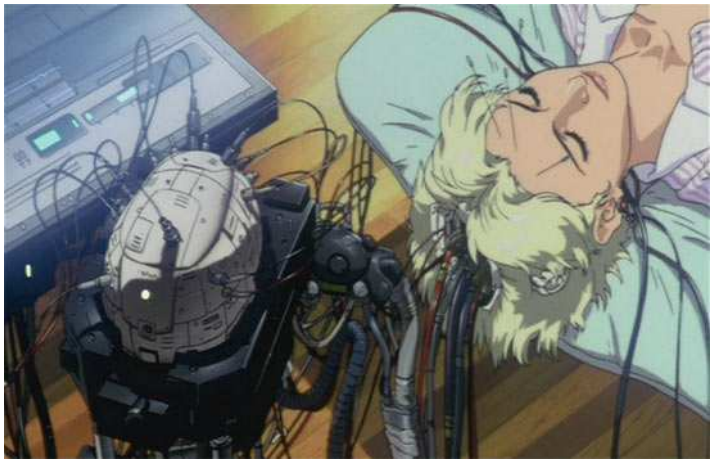
- *“It's really hard to design products by focus groups. A lot of times, people don't know what they want until you show it to them”*
  Steve Jobs - 1998



Requirements Elicitation
Goals and Challenges

- Determine sources of information & appropriate techniques
- Get information on domain, problem, constraints
- Produce a first document
  - Mainly user requirements and elicitation notes
  - Potentially incomplete, disorganized, inconsistent
    - But we must start somewhere!

- You need to extraction information from:
  - Available documentation (may increase with your own problem understanding)
  - The brain of your customer without damaging the customer, much less his brain!
  - Good technology and good tools
    - can help
    - cannot substitute for adequate social interaction!



Requirements Elicitation
Risks

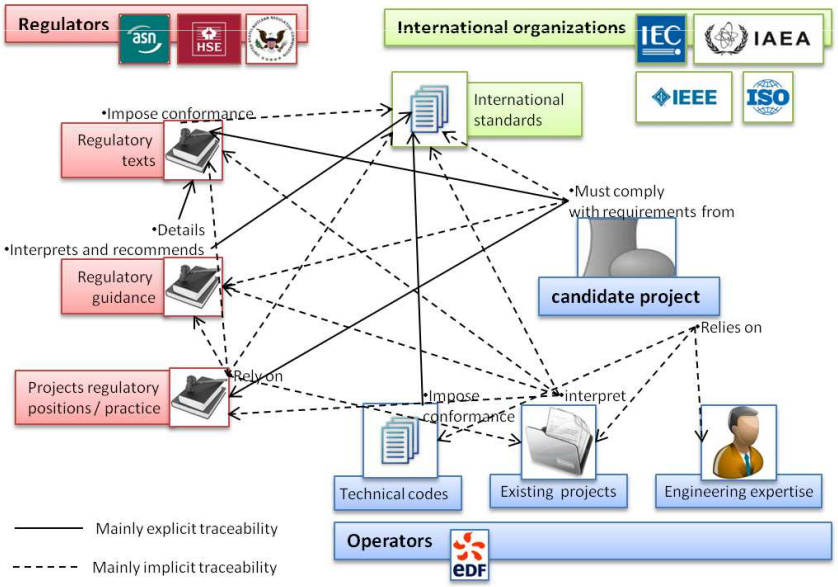
- Typical issues
  - Experts seldom available
  - Finding an adequate level of precision/detail
  - Common vocabulary often missing

- Requirements do not fall from the sky!
  - Sometimes hidden
  - Sometimes too obvious, implicit, ordinary…
    - *Assume == “ass” of “u” and “me”*

- Participants often lack motivation and resist to change
- We need much effort and discussion to come up with a common agreement and understanding!

Sources of Requirements

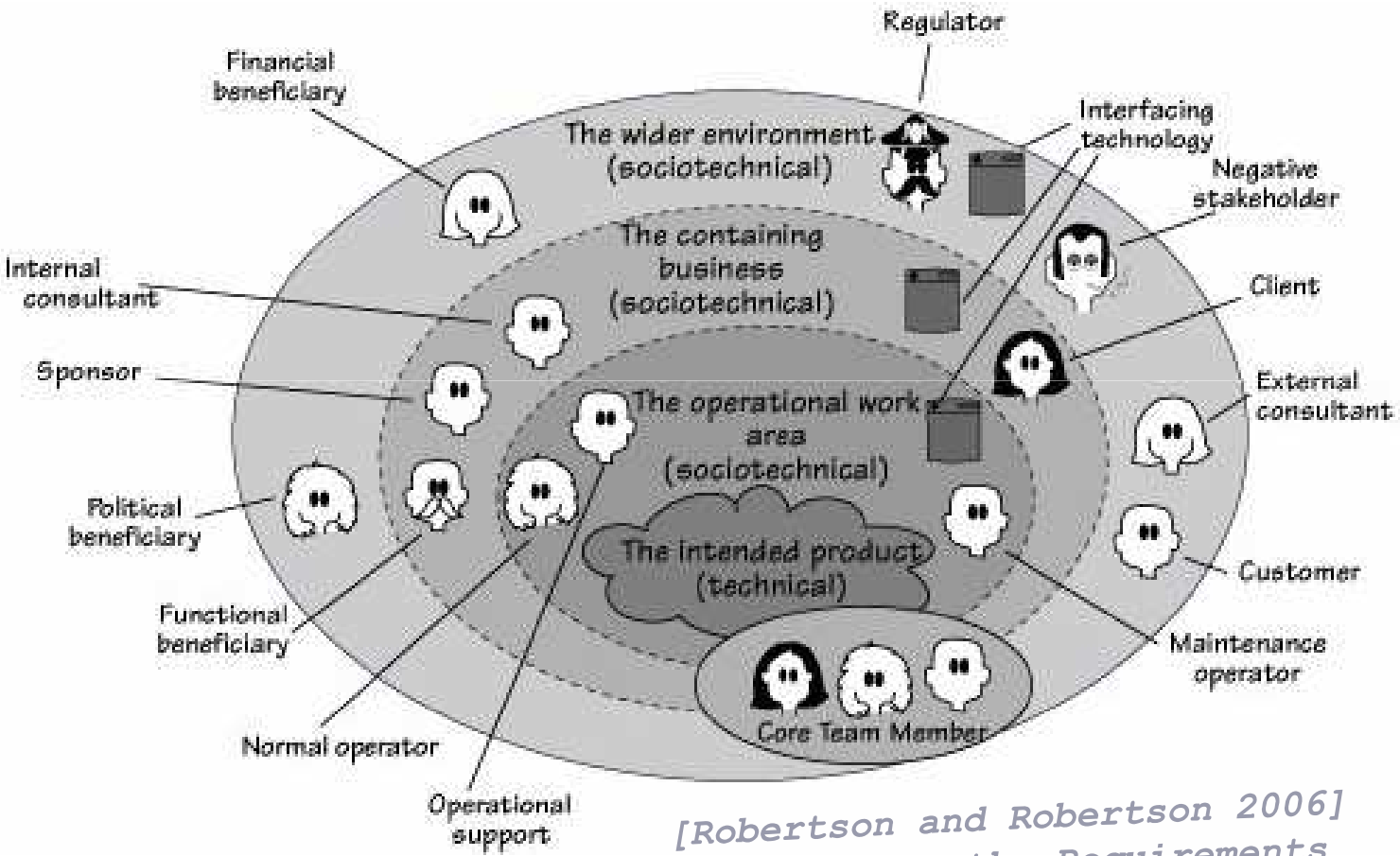
- Various stakeholders
  - Clients, customers, users (past and future), buyers, managers, domain experts, developers, marketing and QA people, lawyers, people involved in related systems, anyone who can bring added value!
  - Often restricted number of persons (and not the best ones)
- Pre-existing systems
  - Not necessarily software systems
- Pre-existing documentation
- Competing systems
- Documentation about interfacing systems
- Standards, policies, collective agreements,
- …



About stakeholders

- Client: Person who pays for the software development
  - Ultimately, has the final word on what will be the product
  - For an internal product, the client is probably a product manager
  - For the consumer market, the customer may be the marketing department
- User of the current system or future system
  - Experts of the current system: indicate which functions to maintain or improve
  - Experts of competitors' products: suggestions on designing a superior product
  - May have special needs or requirements
    - Usability, training, online help ...
  - Do not neglect interest groups
    - Expert users, or with disabilities or handicaps
  - Select users with care
    - Different seniority
    - Must speak with authority and be responsible and motivated

About stakeholders



[Robertson and Robertson 2006] « Mastering the Requirements Process », fig 3.6 & D.1

About stakeholders

- Domain Expert
  - Expert who knows the work involved
  - Familiar with the problem that the software must solve. For example:
    - Financial expert for finance management software
    - Aeronautical engineers for air navigation systems
    - Meteorologist for weather forecasting system, etc…
  - Also knows the environment in which the product will be used
- Inspector
  - An expert in governmental rules and safety relevant to the project
  - Examples: safety inspectors, auditors, technical inspectors
- Lawyer
  - Familiar with laws, legal aspects, and/or standards relevant to the project
- Expert of systems that interact with the system to be built
  - Knows the interfaces of the interacting systems
  - May be interested in product features

About stakeholders

- Stakeholders are generally busy!
  - Have priorities other than you
  - Are rarely entirely disconnected from their daily routine and tasks
  - See their participation in the elicitation process as a supplementary task
- Hence, you must have the support and commitment of managers (especially theirs!)
- You must also avoid being perceived as a threat:
  - Loss of jobs caused by the improved system
  - Loss of autonomy, powers, or privileges
  - To the recognition and visibility of their work

  - *"Knowledge is power, guard it well."*
    Warhammer 40k – Blood Ravens battle cry

Requirements Elicitation Tasks

- Planning for the elicitation
  - Why? Who? When? How? Risks?
- During the elicitation
  - Confirm the viability of the project (is it worth it?)
  - Understand the problem from the perspective of each stakeholder
  - Extract the essence of stakeholders' requirements
  - Invent better ways to do the work of the user
  - Being innovative!
- Following the elicitation
  - Analyse results to understand obtained information
  - Negotiate a coherent set of requirements acceptable by all stakeholders and establish priorities
  - Record results in the requirements specification

Requirements Elicitation tasks

- Elicitation is incremental
  - Repeat as much as necessary
  - Driven by information obtained
  - You always do a bit of elicitation – analysis – specification – verification at the same time (even in a V-Cycle)

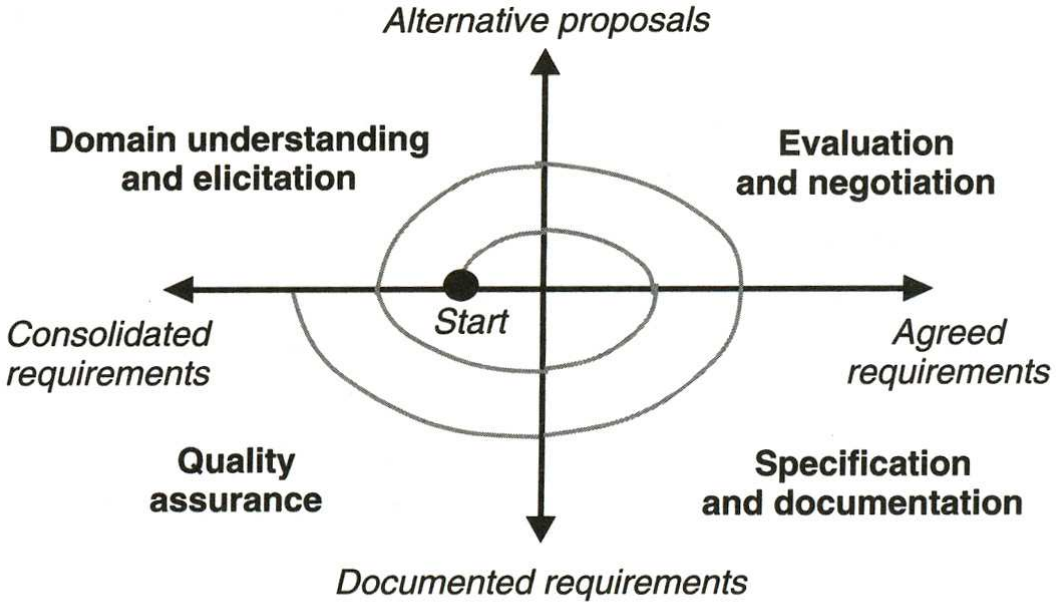


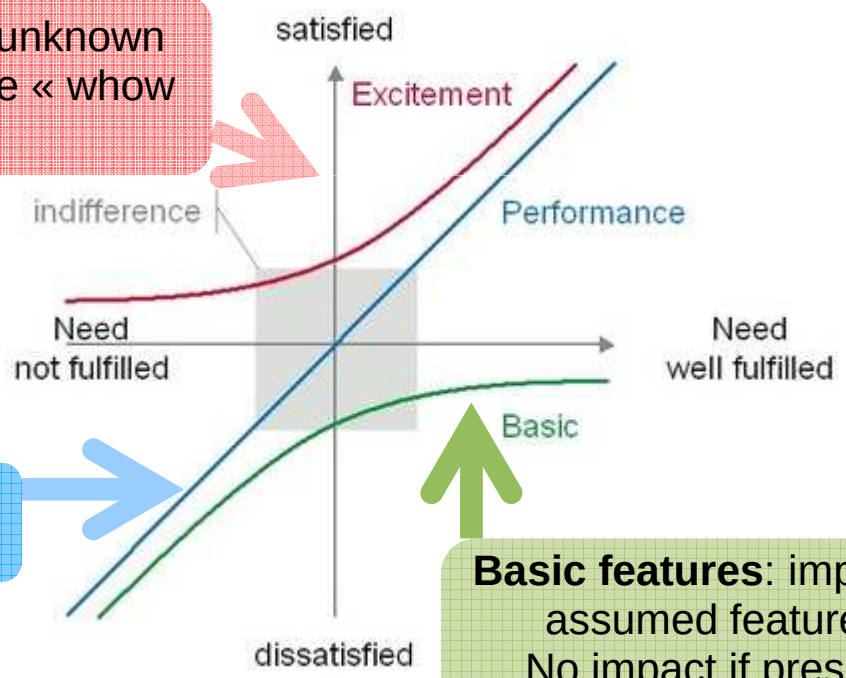
Figure 1.6 *The requirements engineering process*

Requirements Elicitation

Being Innovative & Attractive

- The Kano satisfaction mode[Kano et al. 1984]

Excitement: Unexpected/unknown feature, which produces the « whow effect ».



Performance : expected feature, explicit requirement

Basic features: implicit or assumed features.
No impact if present, unsatisfaction if missing

Requirements Elicitation Techniques

- Traditional techniques
  - Introspection
  - Reading existing documents
  - Analyzing hard data
  - Interviews
    - Open-ended
    - Structured
  - Surveys / Questionnaires
  - Meetings
- Collaborative techniques
  - Group techniques
  - Focus Groups
  - Brainstorming
  - JAD/RAD workshops
  - Prototyping
  - Participatory Design
- Cognitive techniques
  - Task analysis
  - Protocol analysis
  - Knowledge Acquisition Techniques
    - Card Sorting
    - Laddering
    - Repertory Grids
- Contextual approaches
  - Ethnographic techniques
    - Participant Observation
    - Enthnography
  - Discourse Analysis
    - Conversation Analysis
    - Speech Act Analysis
  - Sociotechnical Methods
    - Soft Systems Analysis

Requirements Elicitation

What Approach for what goal?

- Objectives: Why this elicitation?
  - Validate market data
  - Explore usage scenarios
  - Develop a set of requirements, etc..
- Set elicitation strategies and processes
  - Approaches used
  - Often a combination of approaches depending on the types and number of stakeholders

- Elicitation plans
  - Usually set of rough requirements
    - Written, audio, video notes
    - Documentation
  - Deliverables depend on objective and technique
  - Generally: un-organized, redundant, incomplete

Requirements Elicitation

Comparison of Some Elicitation Techniques

Technique	Good for	Kind of data	Plus	Minus
Questionnaires	Answering specific questions	Quantitative and qualitative data	Can reach many people with low resource	The design is crucial. Response rate may be low. Responses may not be what you want
Interviews	Exploring issues	Some quantitative but mostly qualitative data	Interviewer can guide interviewee. Encourages contact between developers and users	Time consuming. Artificiaenvironment may intimidate interviewee
Focus groups and workshops	Collecting multiple viewpoints	Some quantitative but mostly qualitative data	Highlights areas of consensus and conflict. Encourages contact between developers and users	Possibility of dominant characters
Naturalistic observation	Understanding context of user activity	Qualitative	Observing actuawork gives insight that other techniques cannot give	Very time consuming. Huge amounts of data
Studying documentation	Learning about procedures, regulations, and standards	Quantitative	No time commitment from users required	Day-to-day work wildiffer from documented procedures

Source: Preece, Rogers, and Sharp “Interaction Design: Beyond human-computer interaction”, p214

Requirements Elicitation
Analysis of Existing System

- Useful when building a new improved version of an existing system
- Important to know:
  - What is used, not used, or missing, What works well, what does not work
  - How the system is used (with frequency and importance) and it was supposed to be used, and how we would like to use it
- Why analyze an existing system?
  - Users may become disillusioned with new system or do not like the new system if it is too different or does not do what they want (risk of nostalgia for old system)
  - To appropriately take into account real usage patterns, human issues, common activities, relative importance of tasks/features
  - To catch obvious possible improvements (features that are missing or do not currently work well)
  - To find out which "legacy" features can/cannot be left out

Requirements Elicitation
Observation and Ethnography

- Observation
  - Get into the trenches and observe specialists “in the wild”
  - Shadow important potential users as they do their work
  - Initially observe silently (otherwise you may get biased information)
  - Ask user to explain everything he or she is doing
  - Session videotaping
- Ethnography also attempts to discover social, human, and political factors, which may also impact requirements
- Can be supplemented later with questionnaires
  - To answer questions that need comparison or corroboration (confirmation)
  - To obtain some statistics from a large number of users (statistical significance)
- Can be supplemented later with interviews
  - Some questions may require more detailed answers
  - You will not be wasting other people's time or your own
  - This is very labour intensive!

Observations & Etnography

- Experiment in the context of an air traffic control system
- Surprising observations
  - Controllers often put aircrafts on potentially conflicting headings with the intention of fixing them later
  - System generates an audible alarm when there is a possible conflict
  - The controllers close the alarms because they are annoyed by the constant warnings
- Incorrect conclusion
  - The controllers do not like audible alarms because they close them

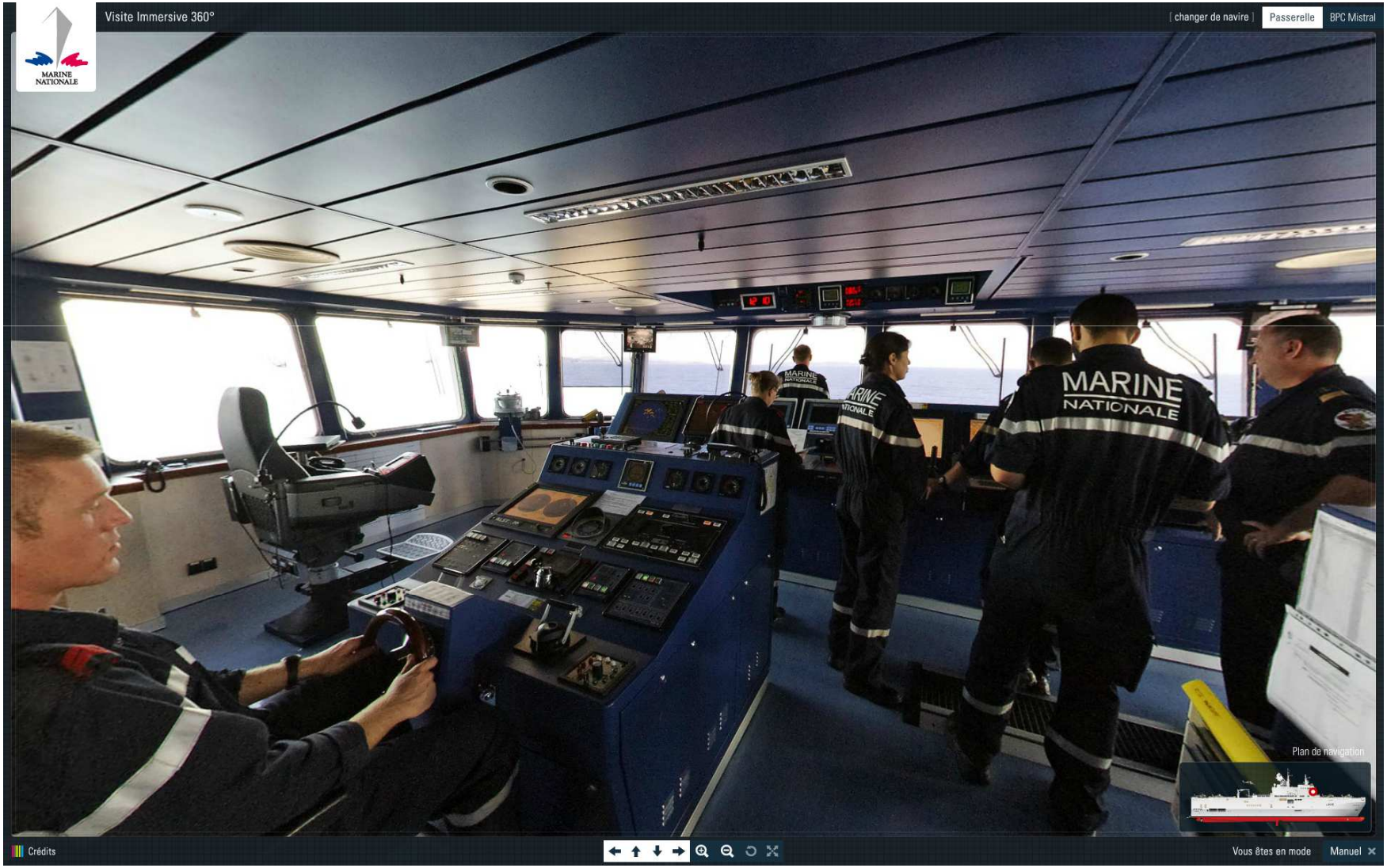
- More accurate observation
  - The controllers do not like being treated like idiots

Ethnography

- Comes from anthropology, literally means "writing the culture"
- Essentially seeks to explore the human factors and social organization of activities → understand work
  - Studies have shown that work is often richer and more complex than is suggested by simple models derived from interviews
- Social scientists are trained in observation and work analysis
- Discoveries are made by observation and analysis, workers are not asked to explain what they do
  - Collect what is ordinary/what is it that people do (make explicit what is implicit)
  - Study the context of work and watch work being done
- Useful to discover for example
  - What does a nuclear technician do during the day?
  - What does his workspace look like?
- Less useful to explore political factors
  - Workers are aware of the presence of an outside observer

Observations & Etnography

Example – Bridge of the French Navy Ship Mistral



Observations & Etnography

Example – Bridge of the French Navy Ship Forbin

Interviews

- Requires preparation and good communication management
  - Achieve interview objectives without preventing the exploration of promising leads
- Interview as many stakeholders as possible
  - Not just clients and users
- Ask problem-oriented questions

- Three main objectives:
  - Record information to be used as input to requirements analysis and modeling
  - Discover information from interviewee accurately and efficiently
  - Reassure interviewee that his/her understanding of the topic has been explored, listened to, and valued

Interviews

- Process consists of four important steps:
  - Planning and preparation (*failing to plan is planning to fail*)
    - Set goals and objectives for the interview
    - Acquire background knowledge of the subject matter to conduct an effective interview
      - About the domain (vocabulary, problems...) but also about interviewees
    - Prepare questions in advance, by topic
    - Organize the environment for conducting an effective interview
      - Determine how the notes will be taken (manually, audio, by whom…)
  - Interview session
    - Make the interviewee comfortable and confident (be polite ...)
    - Adjust to the interviewee (You have your goals – be persistent but flexible)
    - Interview several people at once to create synergy
    - Try to detect aspects as they may influence the said and the unsaid
  - Consolidation of information and follow-up

Interviews
Elicitation Notes

- Revise and complete the elicitation notes after the interview
  - Needs to be done soon after because one forgets the details (and everything else)
- Identify inconsistencies and address them in a follow-up interview or by email
- Keep all diagrams, charts, models created during the discussions
- You are learning, so be precise
  - Pay attention to terminology
  - Use the interviewee's terminology
  - Identify synonyms
  - Create a glossary if necessary
- **Thank the participants** (e.g., by email), and keep the door open

Interviews

Common mistakes

- Not interviewing all of the right people
  - Different points of view of stakeholders
- Asking direct questions too early
  - Keep being problem-oriented
- Interviewing one-at-a-time instead of in small groups
  - More people might help get juices flowing as in brainstorming
  - Users cannot think of everything they need
  - Reduces spotlight on individuals
  - Creates Synergy (the whole is better than the sum of each individuals)
- Assuming that stated needs are exactly correct
  - Often users do not know exactly what they want
  - Need to narrow what is asked for down to what is needed
- Trying to convince stakeholders that YOU are smart
  - This is not about you! This is about your stakeholder's needs!

Interviews

Starting questions - Context-free questions to narrow the scope a bit

- Identify customers, goals, and benefits
  - Who is (really) behind the request for the system?
  - Who will use the system? Willingly?
  - Are there several types of users?
  - What is the potential economic benefit from a successful solution?
  - Is a (pre-existing) solution available from another source?
- When do you need it by?
  - Can you prioritize your needs?
  - What are your constraints? Time - Budget - Resources (human or otherwise)
  - Expected milestones (deliverables and dates)?
- Try to characterize the problem and its solution
  - What problems is the system trying to address? “Good” solution?
  - In what environment? Performance issues? Special constraints?
  - What is (un)likely to change? Future evolution?

Interviews

Starting questions - Context-free questions to narrow the scope a bit

- Calibration and tracking questions
  - Are you the right person to answer these questions?
  - Are your answers "official"? If not, whose are?
  - Are these questions relevant to the problem as you see it?
  - Are there too many questions? Is this the correct level of detail?
  - Is there anyone else I should talk to?
  - Is there anything else I should be asking you? Have you told me everything you know about the problem?
  - **Do YOU have any questions**?
- Questions that cannot be asked directly (ask indirectly)
  - Are you opposed to the system?
  - Are you trying to obstruct/delay the system?
  - Are you trying to create a more important role for yourself?
  - Do you feel threatened by the proposed system?
  - Are you trying to protect your job? Is your job threatened by the new system?
  - Is anyone else's?

Interviews
Specific questions

- Functional requirements
  - What will the system do? When wilt he system do it?
  - Are there several modes of operations?
  - What kinds of computations or data transformations must be performed?
  - What are the appropriate reactions to possible stimuli?
  - For both input and output, what should be the format of the data?
  - Must any data be retained for any period of time?
- Design Constraints
  - Physical environment
    - Where is the equipment to be located? Is there one location or several?
    - Are there any environmental restrictions, such as temperature, humidity, or magnetic interference?
    - Are there any constraints on the size of the system?
    - Are there any constraints on power, heating, or air conditioning?
    - Are there constraints on the programming language because of existing software components?

Interviews
Specific questions

- Design Constraints
  - Interfaces
    - Is input coming from one or more other systems?
    - Is output going to one or more other systems?
    - Is there a prescribed way in which input/output need to be formatted?
    - Is there a prescribed way for storing data?
    - Is there a prescribed medium that the data must use?
  - Standards
    - Are there any standards relevant to the system?
  - Laws, policies, and regulations
    - Are there any laws, policies, or regulations applicable here?
- Performance
  - Are there constraints on execution speed, response time, or throughput?
  - What efficiency measure will apply to resource usage and response time?
  - How much data will flow through the system?

Interviews

Specific questions

- Usability and Human Factors
  - What kind of training will be required for each type of user?
  - How easy should it be for a user to understand and use the system?
  - How difficult should it be for a user to misuse the system?
- Security
  - Must access to the system or information be controlled?
  - Should each user's data be isolated from data of other users?
  - Should user programs be isolated from other programs and from the operating system?
  - Should precautions be taken against theft or vandalism?

Interviews

Specific questions

- Reliability and Availability
  - Must the system detect and isolate faults?
  - What is the prescribed mean time between failures?
  - Is there a maximum time allowed for restarting the system after failure?
  - How often will the system be backed up?
  - Should precautions be taken against fire or water damage?
- Maintainability
  - Will maintenance merely correct errors, or will it also include improving the system?
  - When and in what ways might the system be changed in the future?
  - How to add features to the system?
  - How easy should it be to port the system from one platform to another?
- Precision and Accuracy
  - How accurate must data calculations be?
  - To what degree of precision must calculations be made?

Interviews

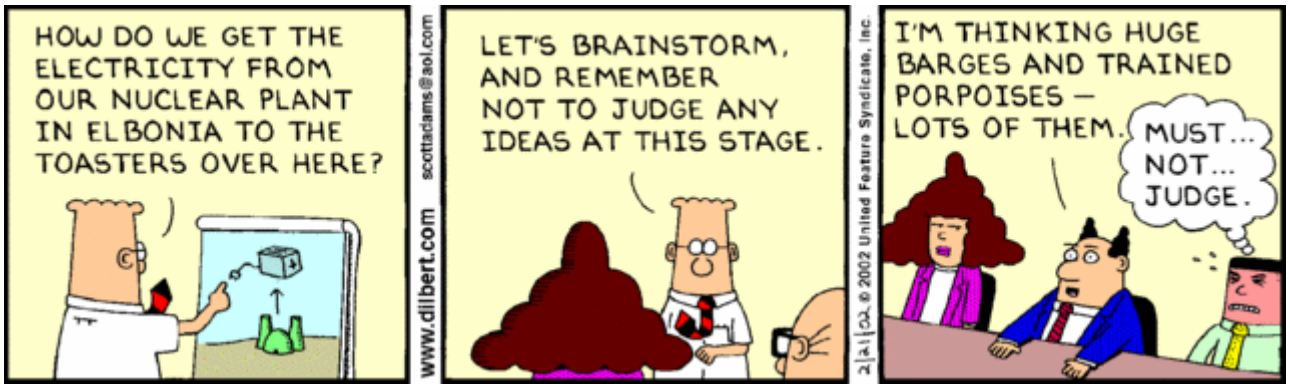
- *“Ignorance is Bliss”*
  Mr Reagan “Cypher” – The Matrix (1999)
- Ignorance is Bliss
  - At least for a short time
  - Ignorance (not stupidity!) allows one to expose hypotheses and some implicit facts

Brainstorming

- To invent new way of doing things or when much is unknown
  - When there are few or too many ideas
  - Early on in a project particularly when:
    - Terrain is uncertain
    - There is little expertise for the type of applications
    - Innovation is important (e.g., novel system)
- Two main activities:
  - The Storm: Generating as many ideas as possible (quantity, not quality) – wild is good!
  - The Calm: Filtering out of ideas (combine, clarify, prioritize, improve…) to keep the best one(s) – may require some voting strategy
- Roles: scribe, moderator (may also provoke), participants

Brainstorming
Objectives

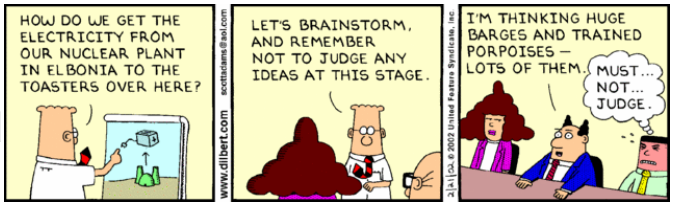
- Hear ideas from everyone, especially unconventional ideas
  - Keep the tone informal and non-judgemental
  - Keep the number of participants “reasonable”
    - if too many, consider a “playoff”-type filtering and invite back the most creative to multiple sessions
- Encourage creativity
  - Choose good, provocative project name / good, provocative problem statement
  - Get a room without distractions, but with good acoustics, whiteboards, coloured pens, provide coffee/donuts/pizza/beer
  - Provide appropriate props/mock-ups



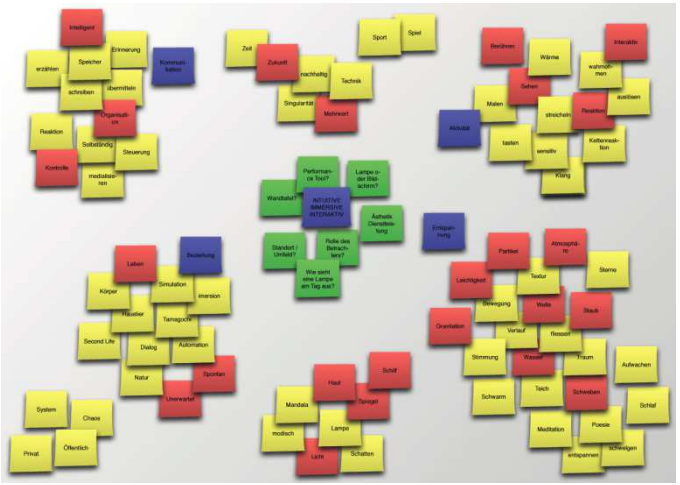
Brainstorming
Roles and Participants

- Scribe
  - Write down all ideas (may also contribute)
  - May ask clarifying questions during first phase but without criticizing
- Moderator/Leader
  - Cannot be the scribe
  - Two schools of thought: traffic cop or agent provocateur
  - Traffic cop – enforces "rules of order", but does not throw his/her weight around otherwise
  - Agent provocateur – traffic cop plus more of a leadership role, comes prepared with wild ideas and throws them out as discussion wanes
    - May also explicitly look for variations and combinations of other suggestions
- Virtually any stakeholder, e.g.
  - Developers, Domain experts, End-users, Clients, ...
- “Ideas-people” – a company may have a special team of people
  - Chair or participate in brainstorming sessions
  - Not necessarily further involved with the project

Brainstorming
Storm and Calm

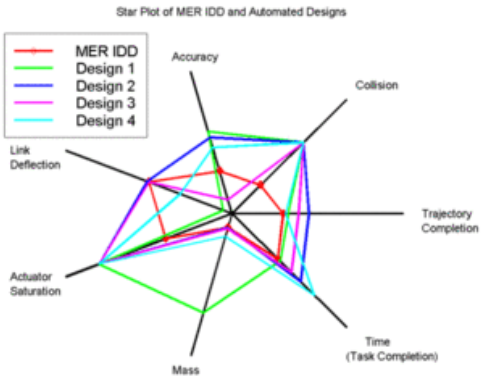


- The Storm
  - Goal is to generate as many ideas as possible
  - Look to combine or vary ideas already suggested
  - No criticism or debate is permitted
  - *Wild is good!*
    - Participants should NOT censor themselves – let yourself go!
- The Calm
  - Go over the list of ideas and explain them clearly
    - Review, consolidate, combine, clarify
  - Categorize into “yes” “maybe” and “no”
    - Rank the list by priority somehow
    - Informal consensus, 50% + 1 vote, veto?
  - Be careful about time and people
    - Long meetings tend to lose focus
      - after 90 to 120 minutes
    - Be careful not to offend participants

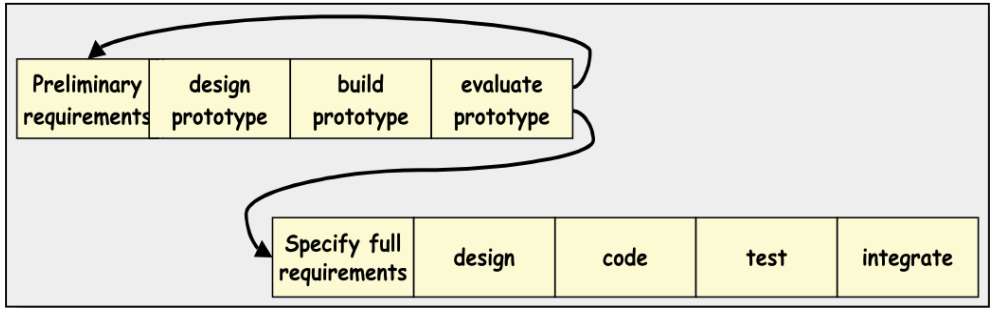


Brainstorming
Eliminating ideas

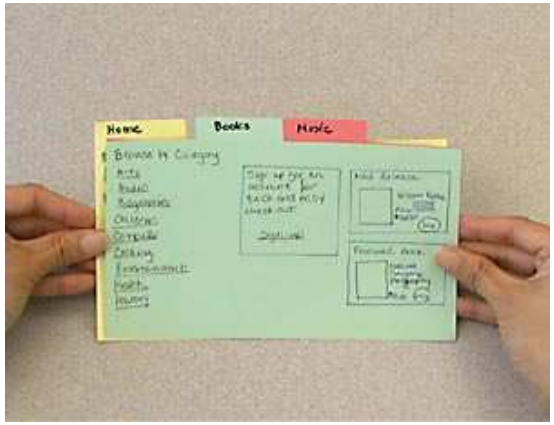
- Blending ideas
  - Unify similar ideas but be aware not to force fit everything into one idea
- Give each participant fake money to spend on the ideas
- Apply acceptance criteria prepared prior to meeting
  - Eliminate the ideas that do not meet the criteria
- Various ranking or scoring methods
  - Assign points for criteria met, possibly use a weighted formula
- Vote with threshold or campaign speeches
  - Possibly select top k for voting treatment



Prototyping



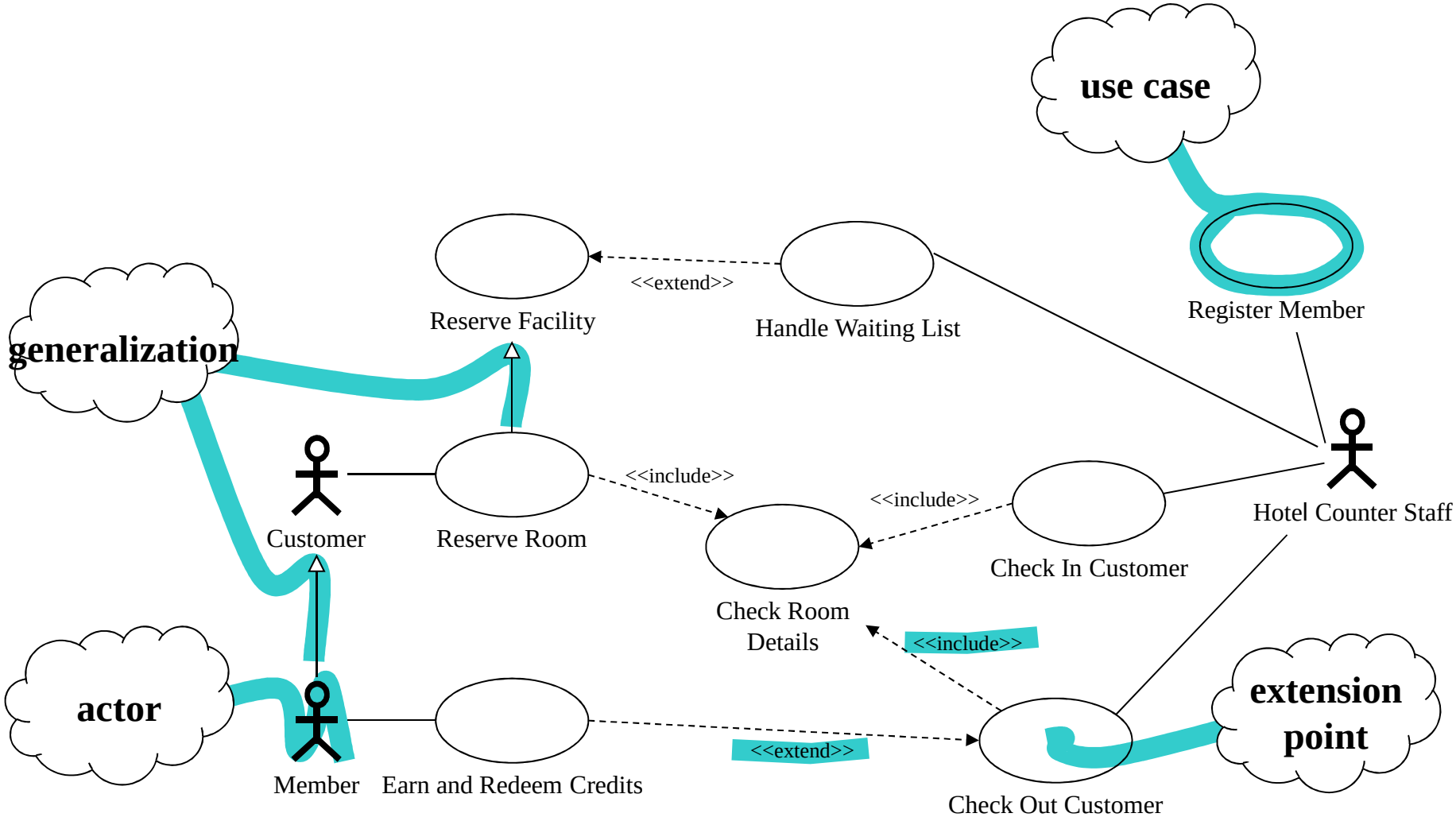
- A software requirements prototype is a mock-up or partial implementation of a software system
  - Helps developers, users, and customers better understand system requirements
  - Helps clarify and complete requirements
  - Provides early response to “I'll know it when I’ll see (or won’t see) it” attitude
  - Effective in addressing the “Yes, But” syndrome
  - Helps find new functionalities, discuss usability, and establish priorities
- Prototyping is effective in resolving uncertainties early in the development process
  - Focus prototype development on these uncertain parts
  - Encourages user participation and mutual understanding
- Prototypes can take many forms:
  - Paper prototypes (see http://www.paperprototyping.com/)
    - Prototype on index card, Storyboard
  - Screen mock-ups
  - Models (executables)
  - Etc.



Use cases

- Use case models
  - Description of a sequence of interactions between a system and external actors
  - Actors – any agent that interact with the system to achieve a useful goal
- Use case describes a typical sequence of actions that an actor performs in order to complete a given task
  - The objective of use case analysis is to model the system
    - … from the point of view of how actors interact with this system
    - … when trying to achieve their objectives
  - A use case model consists of
- A use case should describe the user's interaction with the system ...
  - Not the computations the system performs
- In general, a use case should cover the full sequence of steps from the beginning of a task until the end
- A use case should only include actions in which the actor interacts with the computer
  - Some views differ on this one!!!
- A use case should be written so as to be as independent as possible from any particular implementation / user interface design

Where are the use cases?



Use Case Diagrams

- To define system boundary (subject), actors, and use cases
  - Subject could be: a physical system, a component, a subsystem, a class
- To structure and relate use cases
  - Associate actors with use cases
  - Include relation
  - Extend relation
  - Generalization (of actors and use cases)
- Inclusions allow one to express commonality between several different use cases
- Inclusions are included in other use cases
  - Even very different use cases can share a sequence of actions (reuse)
  - Enable you to avoid repeating details in many use cases (consistency)
- An inclusion represents the execution of a lower-level task with a lower-level goal
- Extensions used to make optional interactions explicit or to handle exceptional cases by creating separate use case extensions, the description of the basic use case remains simple
- Use sparingly: there is disagreement over the semantics

Scenarios

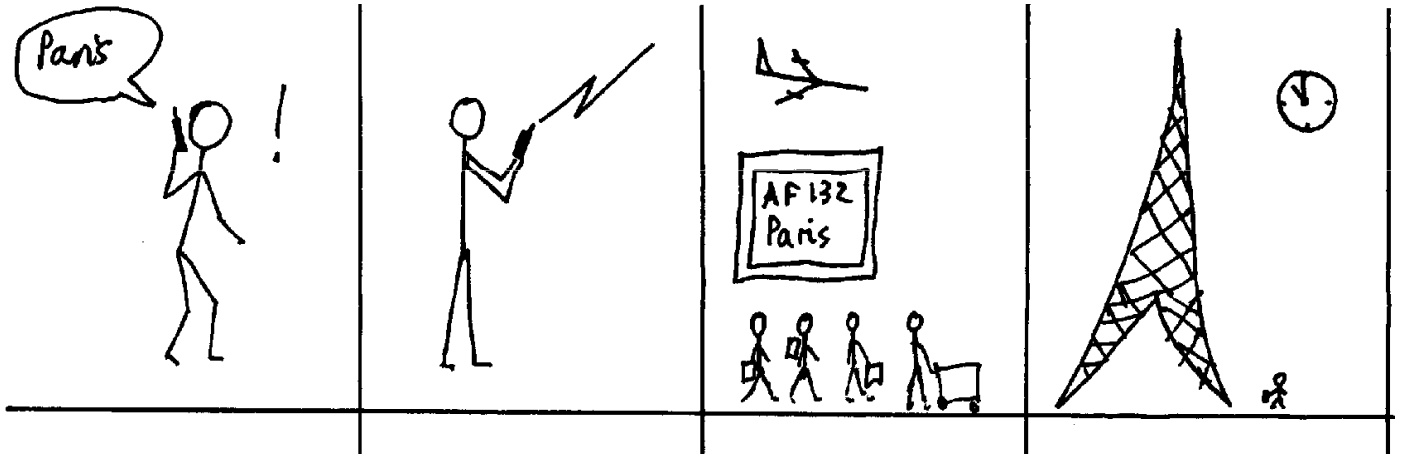
- A scenario (according to the UML/UC community) is an instance of a use case
  - It expresses a specific occurrence of the use case (a specific path through the use case)
    - A specific actor, at a specific time, with specific data …
  - Many scenarios may be generated from a single use case description
  - Each scenario may require many test cases
- A use case includes primary and secondary scenarios
- 1 primary scenario for the normal course of events
- 0 or more secondary scenarios
  - Alternative/exceptional course of events, variations of primary scenario
  - An alternative scenario meets the intent of the use case but with a different sequence of steps
  - An exceptional scenario addresses the conditions of main case and alternative cases that differ from the norm and cases already covered

Types of Scenarios

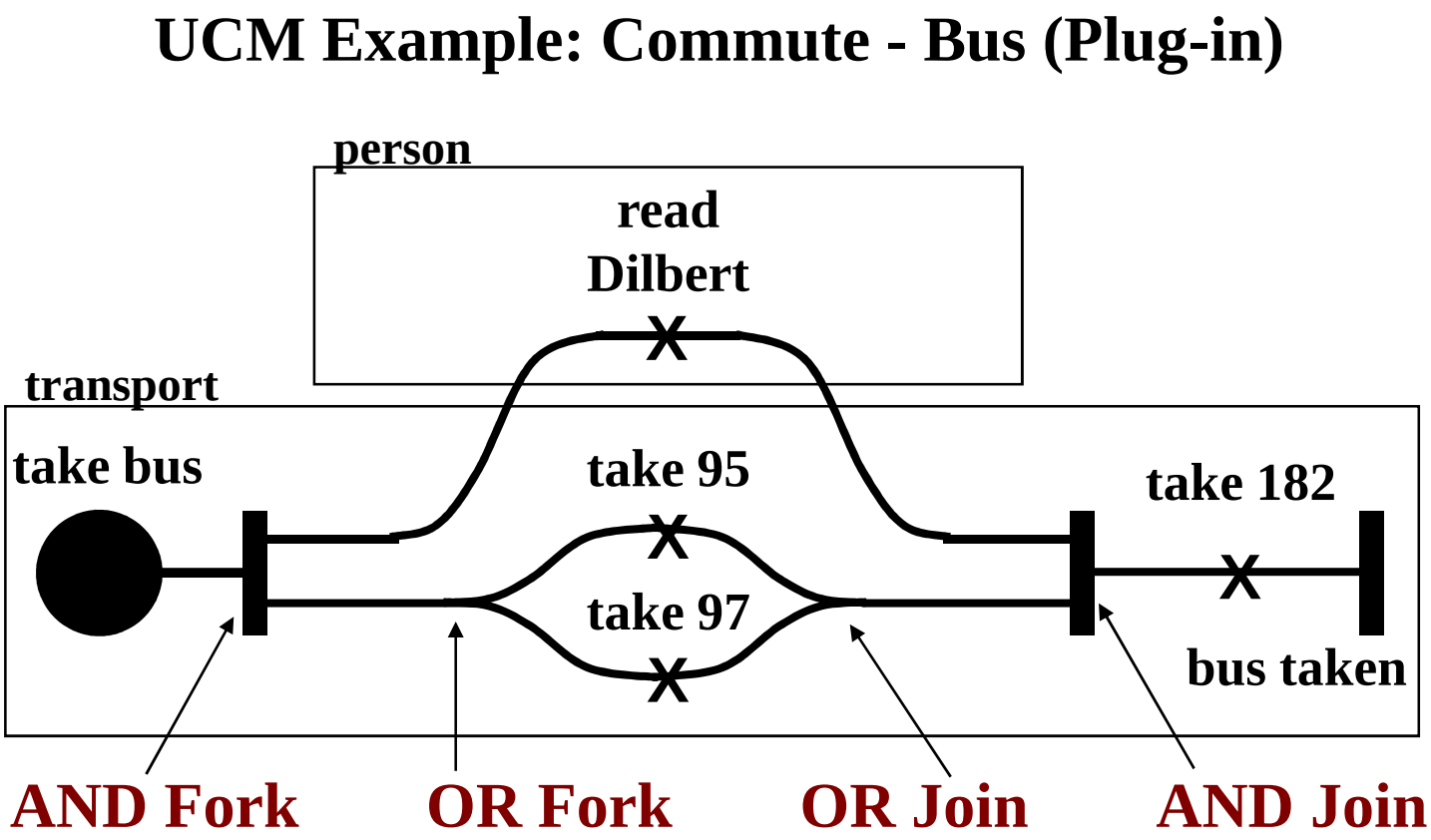
- As-is scenario
  - Used in describing a current situation, usually used in re-engineering projects, the user describes the system
- Visionary scenario
  - Used to describe a future system, usually used in greenfield engineering and reengineering projects
  - Can often not be done by the user or developer alone
- Evaluation scenario
  - User tasks against which the system is to be evaluated
- Training scenario
  - Step by step instructions that guide a novice user through a system

Scenarios Representations

- Different representations may be useful in specific situations
  - For example, storyboards, often used in the film industry, can describe situations, roles, and sequences of tasks in a fast, compact, and polyglot way[1]



Scenarios with URN & Use Case Maps

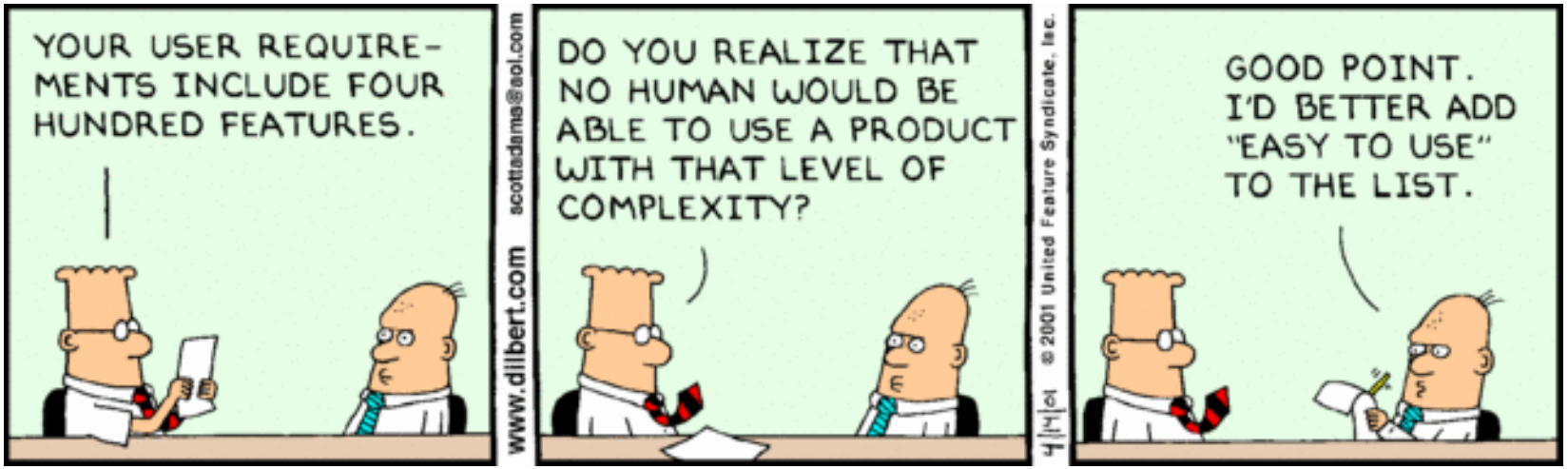


REQUIREMENTS SPECIFICATION

WRITING BETTER REQUIREMENTS

Introduction

- Writing clear stuff
- *I don't know half of you half as well as I should like, and I like less than half of you half as well as you deserve.*
  Bilbo Baggins, The fellowship of the Ring



Alice and Bob cannot write Requirements because…

- She/He doesn't know what to do!
  - She/He was not taught at school
  - She/He doesn't know how to write
  - She/He doesn't understand the process
  - She/He doesn't have the necessary data
- She/He doesn't understand why!
  - She/He doesn't understand the impact / changes
  - She/He thinks this is "just a document"
- She/He'd rather do something else!
  - She/He'd rather design – she sees no reward
  - She/He doesn't have enough time
  - She/He thinks the review process will catch the errors

Natural Language Requirements

- **Universal** : independent from any domain
- **Flexible** : abstractions and refinements without constraints
- **Understandable** : without training or tools
  - The sole prerequisite is to know how to read in the written language

BUT

- **Ambiguous by nature**
- Depending of the **cultural context** of the reader
- Ambiguity is very **hard to detect**

5 kinds of ambiguity [Pohl]:

- **Lexical ambiguity** (meaning of a word)
- **Syntactical ambiguity** (structure of a sentence)
- **Semantic ambiguity** (meaning of a sentence)
- **Referential ambiguity** (target or domain)
- Use of **unclear** or **vague** terms

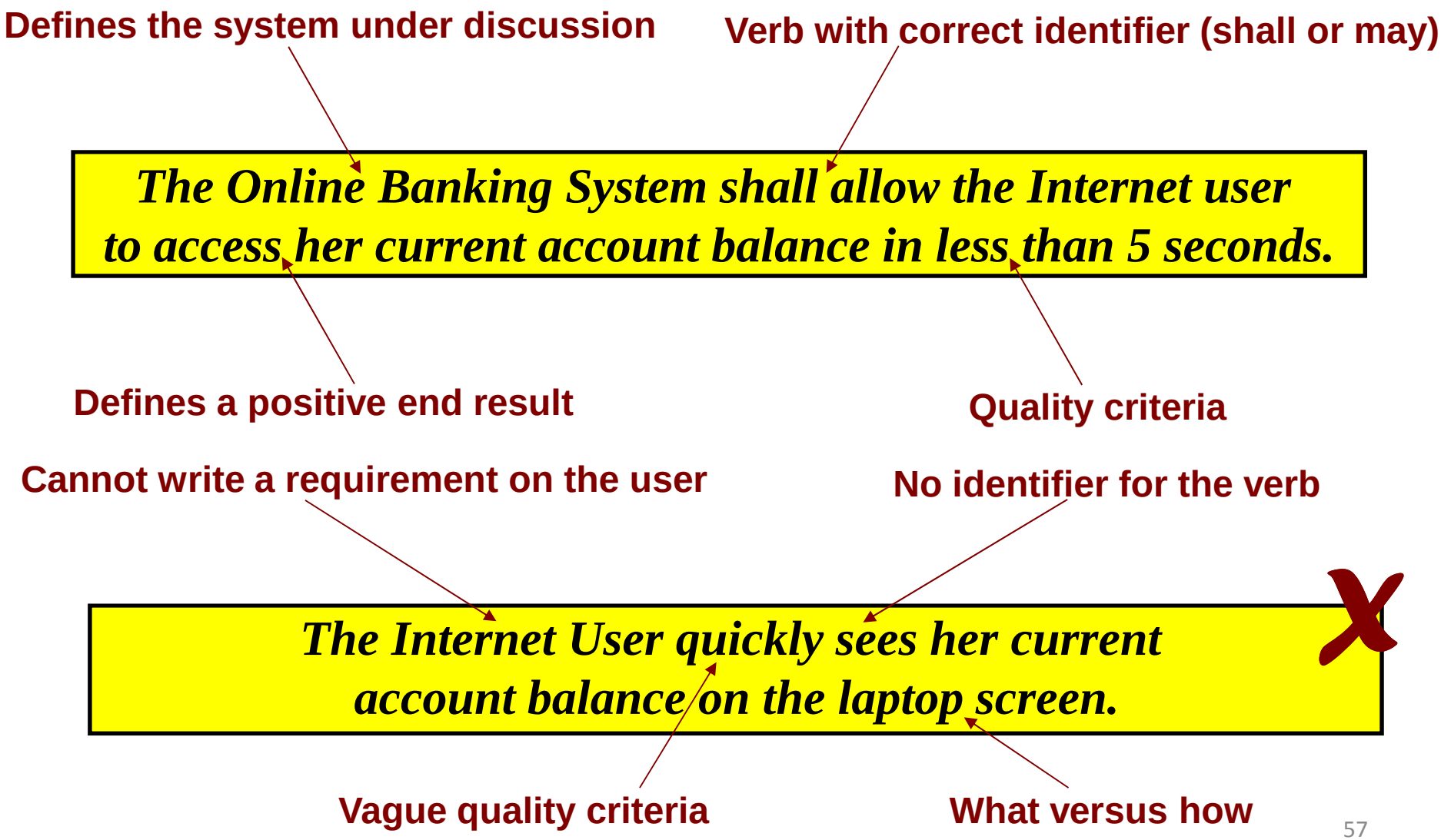
Natural Language ambiguity

- Lexical ambiguity
  - “The fisherman went to the bank”
    Bank is a as well a financial institution and the ground along the edge of a river
- Syntactical ambiguity
  - “Alice saw a man with a telescope.”
    Alice, using a telescope, saw a man.
    Alice saw a man holding a telescope in his hands.
- Semantic ambiguity
  - "There was not a single man at the party."
    No men at all at the party
    No men who were single at the party
- Referential ambiguity

Why is it ambiguous?

- Requirements may be ambiguous both intentionally or not [Breaux]
  - Standards are voluntary ambiguous to fit maximum concerns over time
- Requirements are ambiguous because:
  - xxx does not know what he/she wants
  - xxx has his/her own perspective on it
  - A 1:1 map is still not the reality – (writing on something is abstracting this thing)

Anatomies of Good and Bad Requirements



Standards for Writing

- Each requirement must first form a complete sentence
  - Not a bullet list of buzzwords, list of acronyms, or sound bites on a slide

- Each requirement contains a subject and predicate
  - Subject: a user type (watch out!) or the system under discussion
  - Predicate: a condition, action, or intended result
  - Verb in predicate: "shall" / "will" / "must" to show mandatory nature; "may" / "should" to show optionality
    - **MUST**, **REQUIRED** or **SHALL**: mean that the definition is an absolute requirement of the spec.
    - **MUST NOT** or **SHALL NOT**: absolute prohibition
    - **SHOULD** or **RECOMMENDED**: think twice about not doing it!
    - **SHOULD NOT** or **NOT RECOMMENDED**: think twice about doing it!
    - **MAY** or **OPTIONAL**: truly optional
- The whole requirement provides the specifics of a desired end goal or result
- Contains a success criterion or other measurable indication of the quality

Standards for Writing

- Look for the following characteristics in each requirement (Amyot)
  - Feasible (not wishful thinking)
  - Needed (provides the specifics of a desired end goal or result)
  - Testable (contains a success criterion/other measurable indication of quality)
  - Clear, unambiguous, precise, one thought
  - Prioritized
  - ID
- Another set of criteria (Pohl, Lucas-Hirz)
  - Complete: No missing information
  - Atomic: Express one and only one requirement
  - Traceable: Source, evolution, impact, effective use
  - Correct: needed
  - Unambiguous: exactly one meaning
  - Consistent: consistent with respect to the terminology
  - Verifiable: testable
  - Up to date: reflect the actual status

Tips for Writing

- Tips from Karl Wiegers – “Writing Quality Requirements”
  - Write short sentences. Use the active voice.
  - To know whether your requirement is explicit enough, read it from the developer and the tester points of view.
  - granularity: Write requirements that can be unitary tested. Avoid “and/or” statements that compose multiple requirements.
  - Keep a consistent and homogeneous level of details.
  - Avoid redundancy. May ease the reading but not maintenance. May lead to inconsistencies.

Writing Pitfalls to Avoid

- Never describe how the system is going to achieve something (over-specification), always describe what the system is supposed to do
  - Refrain from designing the system
    - Danger signs: using names of components, materials, software objects, fields & records in the user or system requirements
  - Designing the system too early may possibly increase system costs
  - Do no mix different kinds of requirements (e.g., requirements for users, system, and how the system should be designed, tested, or installed)
  - Do not mix different requirements levels (e.g., the system and subsystems)
    - Danger signs: high level requirements mixed in with database design, software terms, or very technical terms

The system shall use Microsoft Outlook to send an emaito the customer with the purchase confirmation.

The system shall inform the customer that the purchase is confirmed.

Writing Pitfalls to Avoid

- Never build in let-out or escape clauses
  - Requirements with let-outs or escapes are dangerous because of problems that arise in testing
  - Danger signs: if, but, when, except, unless, although
    - These terms may however be useful when the description of a general case with exceptions is much clearer and complete that an enumeration of specific cases
- Avoid ambiguity
  - Write as clearly and explicitly as possible
  - Ambiguities can be caused by:
    - The word or to create a compound requirement
    - Poor definitions (giving only examples or special cases)
    - The words etc., …and so on (imprecise definition)
- Do not use vague indefinable terms
  - Many words used informally to indicate quality are too vague to be verified
  - Danger signs: user-friendly, flexible, approximately, easy, as much as possible

Writing Pitfalls to Avoid

- Do not make multiple requirements
  - Keep each requirement as a single sentence
  - Conjunctions are danger signs: and, or, with, also
- Do not ramble
  - Long sentences with arcane language
  - References to unreachable documents (traceability issues)
- Do not speculate
  - There is no room for “wish lists” – Things that somebody probably wants
  - Danger signs: vague subject type and generalization words such as usually, generally, often, normally, typically
- Do not express suggestions or possibilities
  - Suggestions that are not explicitly stated as requirements are invariably ignored
  - Danger signs: may, might, should, could, perhaps, probably
- Avoid wishful thinking
  - Wishful thinking means asking for the impossible (e.g., 100% reliable, safe, handle all failures, fully upgradeable)

Writing Requirements Using EARS Templates

A. Mavin et al., Easy Approach to Requirements Syntax, RE2009

- You said EARS? Easy Approach to Requirements Syntax
  - Alistair Mavin et al., Easy Approach to Requirements Syntax, RE2009
- Two classes of requirement
  - Normal operation
    - Define the required system behavior during *sunny day operation*
    - All users and all interacting systems behave as expected to meet the goals of the user
  - Unwanted behavior
    - A general term used to cover all deviations from *sunny day operation*
    - Define the *required response of the* system to
      - Failures and disturbances
      - Deviations from desired user behavior
      - Unexpected behavior of interacting systems

Writing with EARS

- Generic syntax is
  - ***<optional preconditions> <optional trigger> the <system name> shall <system response>***
  - Simple structure adds rigor & clarity
  - System response describes what the system must actually do that is *visible* at the boundary of the system

- Ubiquitous: Requirement is always active
- Event-driven (keyword *When):* Required response to a triggering event
- State-driven (keyword *While):* Required response in a specified state
- Option (keyword *Where):* Applicable only if *feature is included*

- *(can use combinations of When, While and Where for requirements with complex conditional clauses)*

EARS Normal Operation Templates

- Ubiquitous Requirement
  - ***The <system name> shall<system response>***
  - Used to define system behavior that must be active at all times →*"continuous"*
  - No preconditions or trigger → *"unconditional"*
- Event-driven Requirement
  - ***When <trigger> the <system name> shall <system response>***
  - Initiated only when a triggering event is detected at the system boundary
  - The trigger must be something that the system itself can detect
- State-driven Requirement
  - ***While <in a specific state> the <system name> shall <system response>***
  - Requirement is active while the system is in a defined state
  - *Requirement is "continuous", but only while the system is in the specified state*
- Option
  - ***Where <feature is included> the <system name> shall <system response>***
  - Applicable only in systems that include a particular *feature*
  - *The requirement will often be "ubiquitous", but only for systems that include the specified feature*

EARS Unwanted Behavior Template

- A variation of e*vent-driven requirement.*
  - ***If <optional preconditions> <trigger>, then the <system name> shall <system response>***
  - This format forces the separation of Circumstances in which the requirement can be invoked (preconditions)
  - The initiating event (trigger)
  - The expected system behavior (response)

Examples of EARS Requirements

- The laptop shall have a minimum battery life of XXX hours
- When the laptop is running and the laptop is closed, the laptop shall enter "powersave" mode
- While the laptop is running on the battery and the battery is below XXX % charge, the laptop shall display "low battery“
- Where the laptop is a "lightweight” model, the laptop shall have a mass of no more than XXX grams
- If the incorrect password is entered, then the laptop shall display XXX warning message

Complex Requirements Syntax

- Requirements with complex conditional clauses are defined using combinations
- of *When, While, Where and If-Then*
- The keywords can be built into more complex expressions to specify richer system behaviors
- For instance, the same event may trigger different system behavior depending on the state of the system when the event is detected

- "While the laptop is running on mains electrical power, if the power cable is disconnected, then the laptop shall display a warning message."

EARS Strengths and weaknesses

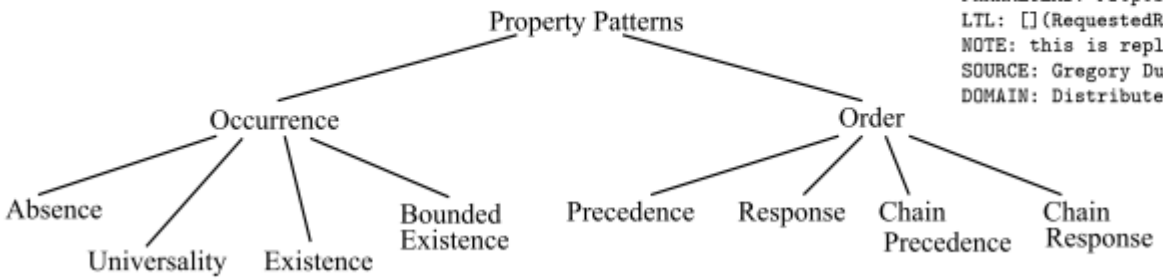
- Strengths
  - Provides rigor and consistency
  - Easy to learn and apply
  - No tools needed
  - Common form of requirements
  - communication
- Weaknesses
  - Limited inter-requirement coupling
  - Unsuitable for very complex requirements
    - (consider using truth tables or other non-textual notation)

Specifying Requirements using Dwyer's Patterns

M. B. Dwyer et al., "Patterns in Property Specifications for Finite-State Verification", ICSE'99

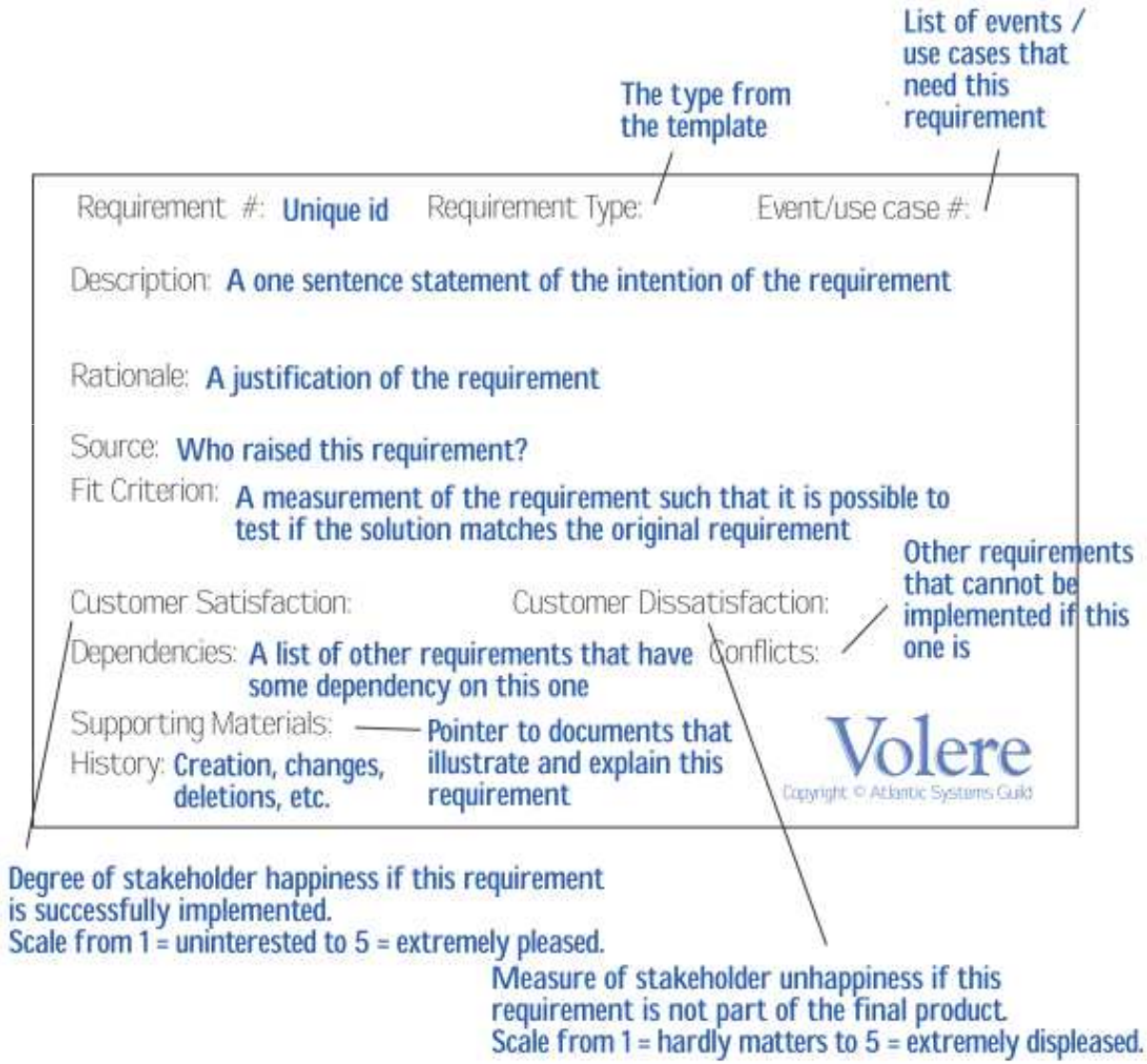
- Requirement: A prose description of the requirement
- Pattern: the pattern of which we determined this requirements is an instance (if any)
- Scope: the scope of the pattern
- Parameters: notes on the parameters provided to the template (formulae, arrays of propositions)
- Mappings: mappings of the property to formal specirfication languages (LTL, CTL, INCA …)
- Source: the source of the example
- Domain: the application domain the example is from
- Note: any additional information

```
REQUIREMENT: When a server requests its registration in
             the ROT, it will eventually be registered.
PATTERN: Response
SCOPE: Global
PARAMETERS: Propositions (boolean vector)
LTL: [](RequestedRegisterImpl[i] -> <>ServerRegistered[i])
NOTE: this is replicated for all modeled servers
SOURCE: Gregory Duval \cite{duval:98}, RI, pp. 46
DOMAIN: Distributed Object System
```



Volere Requirement Shells

Suzanne and James Robertson, "Mastering the Requirements Process ", Addison-Wesley, London, 1999.



Toward Good Requirements Specifications

- Valid (or “correct”)
  - Expresses actual requirements
- Complete
  - Specifies all the things the system must do (including contingencies)
  - ...and all the things it must not do!
  - Conceptual Completeness
    (e.g., responses to all classes of input)
  - Structural Completeness
    (e.g., no TBDs!!!)
- Consistent
  - Doesn’t contradict itself (satisfiable)
  - Uses all terms consistently
  - Formal modeling can help
- Beneficial
  - Has benefits that outweigh the costs of development

Toward Good Requirements Specification

- Necessary
  - Doesn't contain anything that isn't "required"
- Unambiguous
  - Every statement can be read in exactly one way
  - Clearly defines confusing terms (e.g., in a glossary)
- Uniquely identifiable
  - For traceability and version control
- Verifiable
  - A process exists to test satisfaction of each requirement
  - "every requirement is specified behaviorally"
- Understandable (clear)
  - E.g., by non-computer specialists
- Modifiable
  - Must be kept up to date!

Typical Mistakes

- Noise = the presence of text that carries no relevant information to any feature
- Silence = a feature that is not covered by any text
- Over-specification = describing a feature of the solution, rather than the problem
- Contradiction = text that defines a single feature in a number of incompatible ways
- Ambiguity = text that can be interpreted in 2 or more different ways
- Forward reference = text that refers to a feature yet to be defined
- Wishful thinking = text that defines a feature that cannot possibly be validated
- Jigsaw puzzles = distributing requirements across a spec and then cross-referencing
- Inconsistent terminology = inventing and then changing terminology
- Delegating = i.e. making the reader work hard to decipher the intent
- Writing for the hostile reader (fewer of these exist than friendly ones)

Some Training …

Rate these requirements

The Order Entry system provides for quick, user-friendly and efficient entry and processing of all orders.

Invoices, acknowledgments, and shipping notices shall be automatically faxed during the night, so customers can get them first thing in the morning.

Changing report layouts, invoices, labels, and form letters shall be accomplished.

The system shall be upgraded in one whack.

The system has a goal that as much of the IS data as possible be pulled directly from the T&M estimate.

REQUIREMENTS SPECIFICATION

WRITING REQUIREMENTS DOCUMENTS

You said Requirements Document?

- 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 the scope and boundaries of the system / software
- Each requirement must be described in such a way that it is feasible and objectively verifiable by a prescribed method (e.g., by inspection, demonstration, analysis, or test)
- Basis for contractual agreements between contractors or suppliers and customers
- Elaborated from elicitation notes
- Specifications are intended to a diverse audience
  - Customers and users for validation, contract, ...
  - Systems (requirements) analysts
  - Developers, programmers to implement the system
  - Testers to check that the requirements have been met
  - Project Managers to measure and control the project
- Different levels of detail and formality is needed for each audience

Templates for Requirements Specification Documents?

- Different templates for requirements specifications
  - e.g. IEEE 830-1998 Standard for *Software* Requirements Specifications
  - e.g. IEEE 1233-1998 Standard for *Systems* Requirements Specifications
- Describes the content and qualities of a good software requirements specification (SRS)
  - *An SRS should be:*
  - *a) Correct;*
  - *b) Unambiguous;*
  - *c) Complete;*
  - *d) Consistent;*
  - *e) Ranked for importance and/or stability;*
  - *f) Verifiable;*
  - *g) Modifiable;*
  - *h) Traceable.*
- Presents several sample SRS outlines

IEEE830 Objectives and Benefits

- Help software customers to accurately describe what they wish to obtain
- Help software suppliers to understand exactly what the customer wants
- Help participants to:
  - Develop a template (format and content) for the software requirements specification (SRS) in their own organizations
  - Develop additional documents such as SRS quality checklists or an SRS writer's handbook

- Establish the basis for agreement between the customers and the suppliers on what the software product is to do
- Reduce the development effort
  - Forced to consider requirements early → reduces later redesign, recoding, retesting
- Provide a basis for realistic estimates of costs and schedules
- Provide a basis for validation and verification
- Facilitate transfer of the software product to new users or new machines
- Serve as a basis for enhancement requests

How to produce a good SRS?

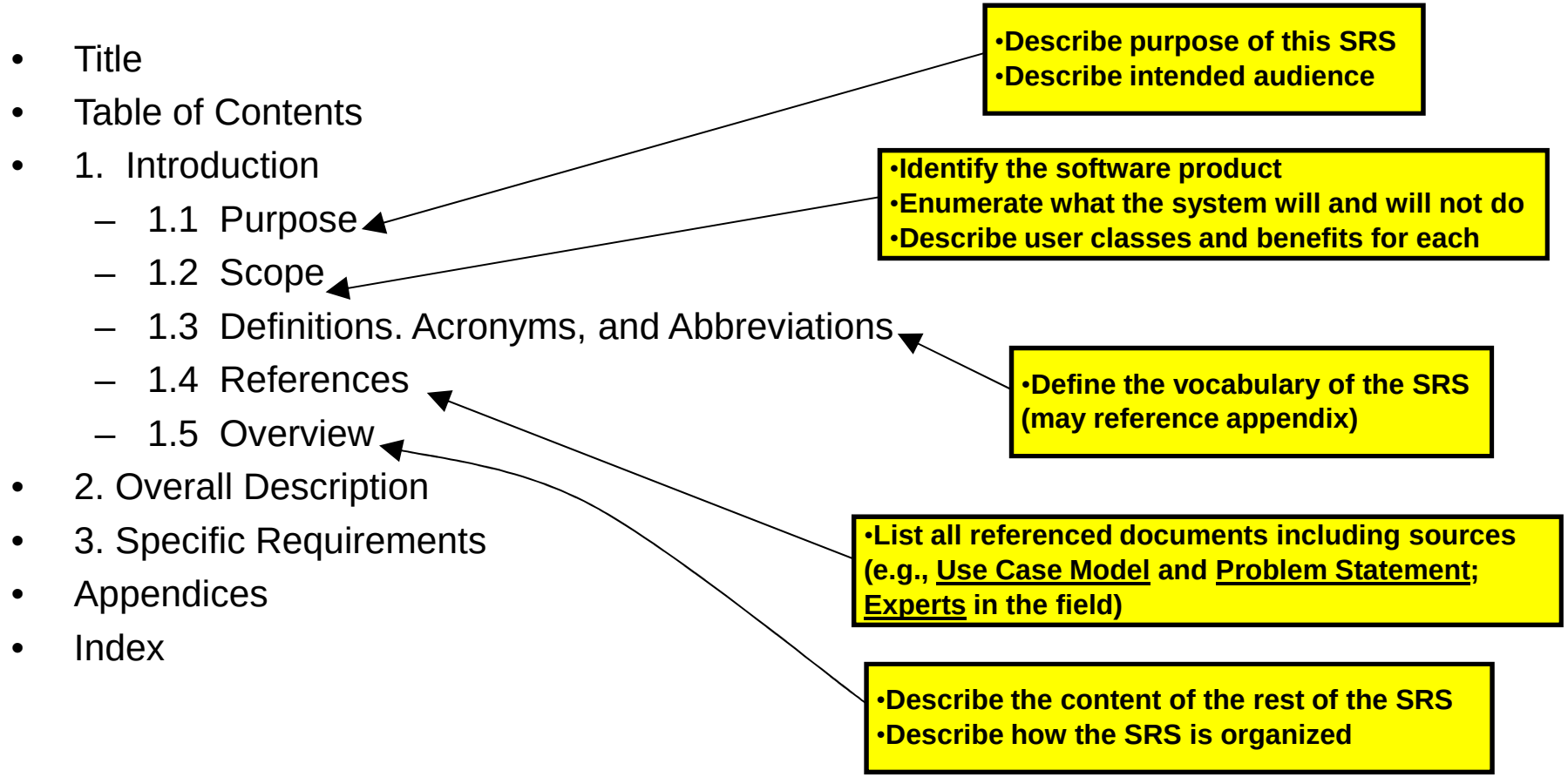
- Section 4 of IEEE 830
  - Nature (goals) of SRS
    - Functionality, interfaces, performance, qualities, design constraints
  - Environment of the SRS
    - Where does it fit in the overall project hierarchy
  - Characteristics of a good SRS
    - Generalization of the characteristics of good requirements to the document
  - Evolution of the SRS
    - Implies a change management process
  - Prototyping
    - Helps elicit software requirements and reach closure on the SRS
  - Including design and project requirements in the SRS
    - Focus on external behavior and the product, not the design and the production process (describe in a separate document)

How to structure a SRS?

- Section 5 of IEEE 830
- Contents of SRS
  - Introduction
  - General description of the software product
  - Specific requirements (detailed)
  - Additional information such as appendixes and index, if necessary

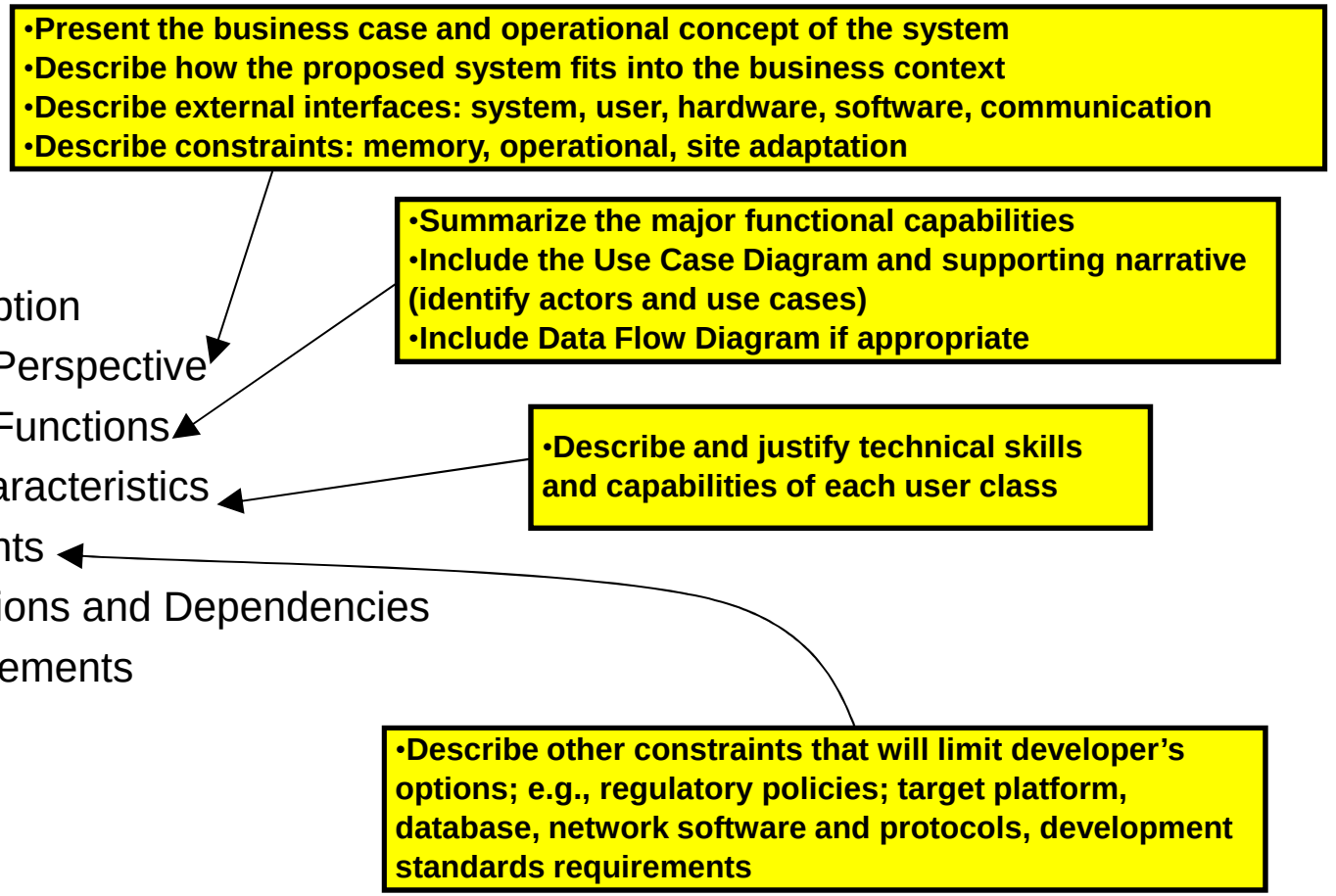
SRS – Section 1

- 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
- 3. Specific Requirements
- Appendices
- Index



SRS Section 2

- Title
- Table of Contents
- 1. Introduction
- 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
- 4. Appendices
- 5. Index



SRS Section 3

- …
- 1. Introduction
- 2. Overall Description
- 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

Specify software requirements in sufficient detail to enable designers to design a system to satisfy those requirements and testers to verify requirements

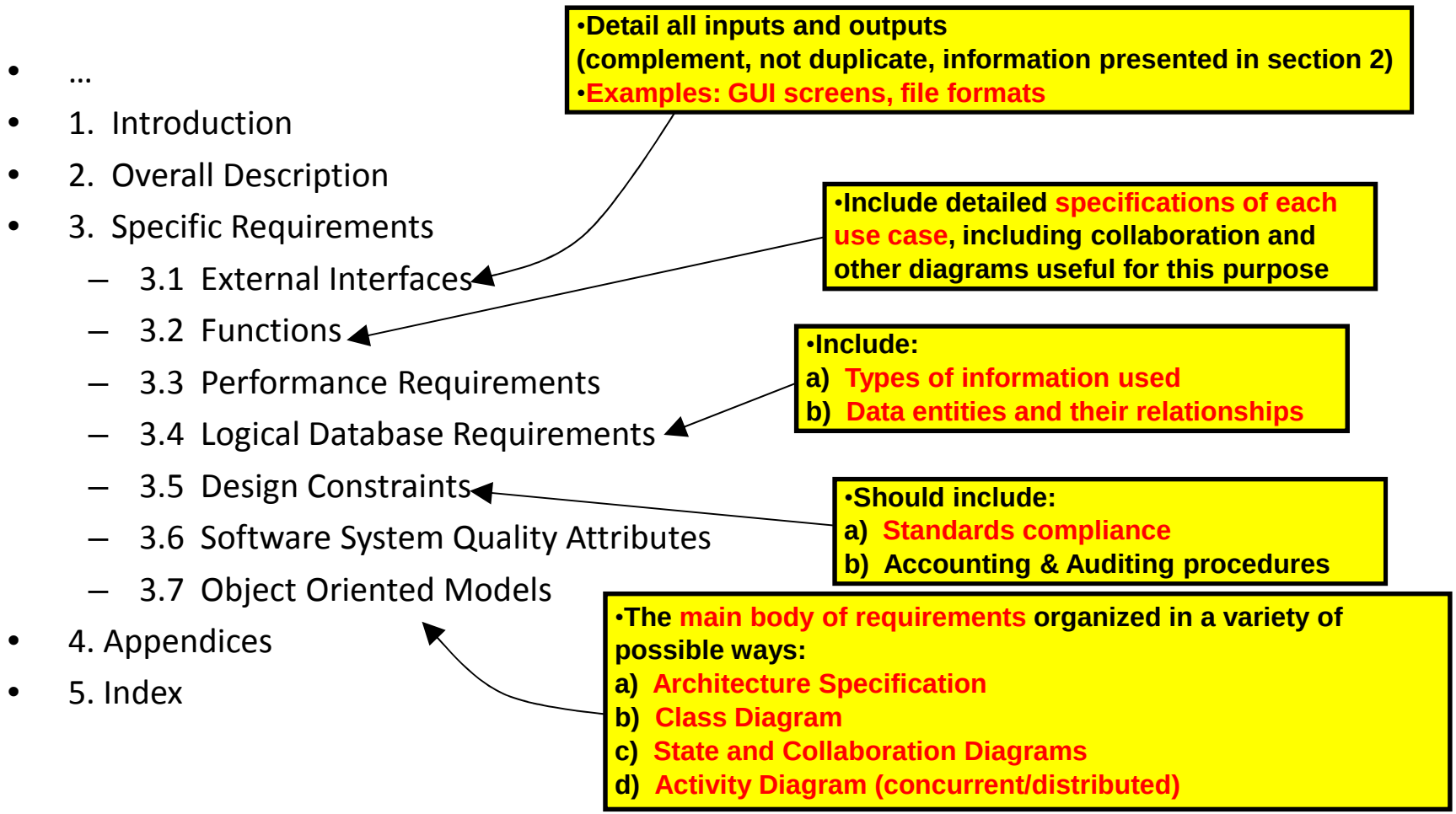
State requirements that are externally perceivable by users, operators, or externally connected systems

Requirements should include, at a minimum, a description of every input (stimulus) into the system, every output (response) from the system, and all functions performed by the system in response to an input or in support of an output

(a) Requirements should have characteristics of high quality requirements
(b) Requirements should be cross-referenced to their source.
(c) Requirements should be uniquely identifiable
(d) Requirements should be organized to maximize readability

SRS Section 3

- …
- 1. Introduction
- 2. Overall Description
- 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



SRS Section 3 - alternatives

- Section 3 (Specific Requirements) may be organized in many different ways based on
  - Modes
  - User classes
  - Concepts (object/class)
  - Features
  - Stimuli
  - Organizations

WHAT NEXT?

What Next?

- What next?
  - RE activities in details
    - (3) Modeling Requirements
    - (3) Requirements Verification and Validation
    - (4) Requirements Management
    - (4) Requirements Traceability
    - (4) Requirements Variability and Software Product Lines
    - (5) Test
    - (5) Requirements Management in Practice with Polarion

Some Training …

Rate these requirements

The Order Entry system provides for quick, user-friendly and efficient entry and processing of all orders. ✗

Invoices, acknowledgments, and shipping notices shall be automatically faxed during the night, so customers can get them first thing in the morning. ✗

Changing report layouts, invoices, labels, and form letters shall be accomplished. ✗

The system shall be upgraded in one whack. ✗

The system has a goal that as much of the IS data as possible be pulled directly from the T&M estimate. ✗