Requirements Quality Assurance

Mehrdad Sabetzadeh

April 7, 2010

What is a Requirement?

➜ No universally agreed definition

➥ But intuitively: A requirement is something that someone needs in order to solve a problem or achieve an objective

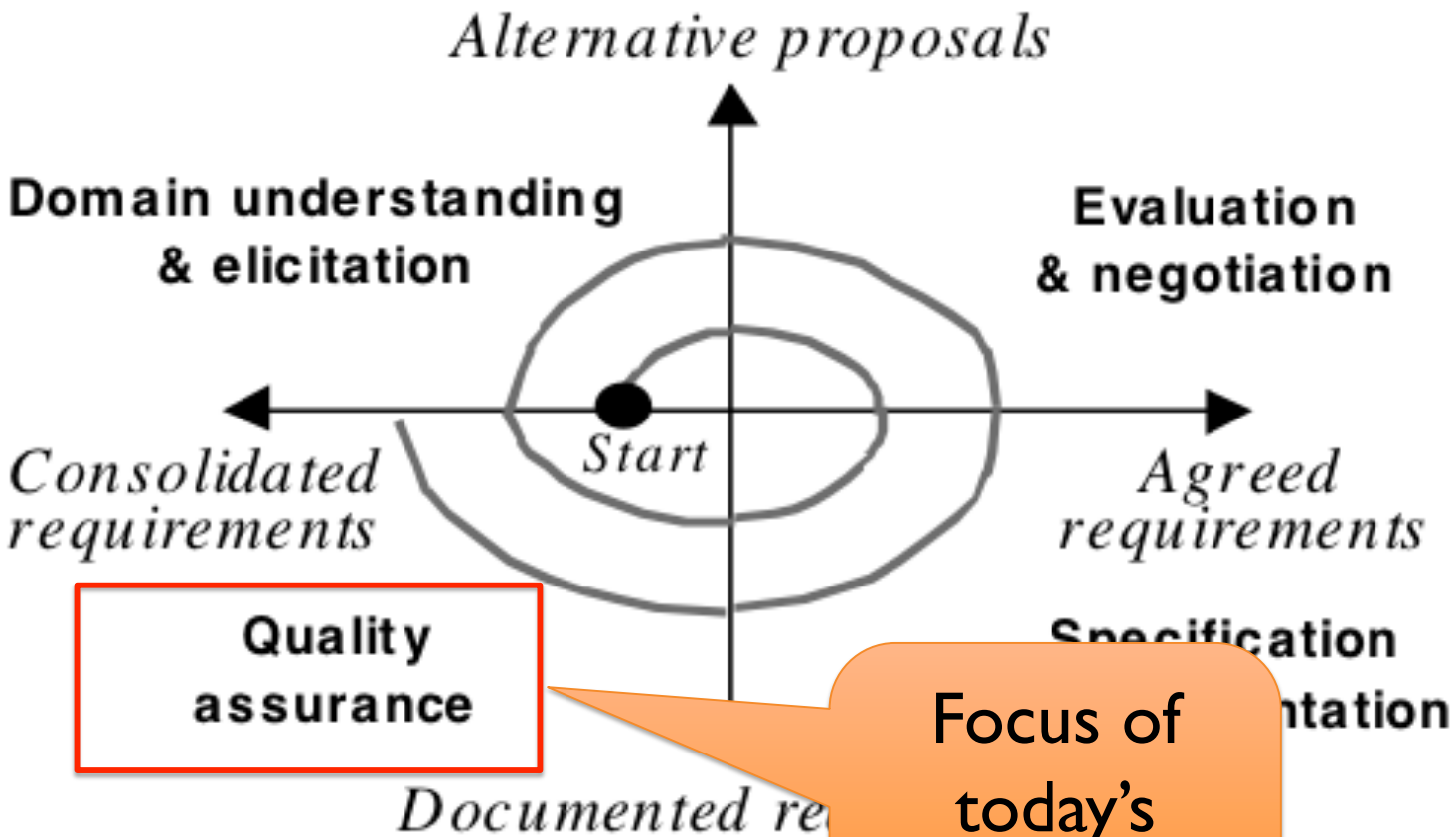
➜ IEEE Definition

"A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document. The set of all requirements forms the basis for subsequent development of the system or system component".
[IEEE Std 830-1998]

Types of Requirements

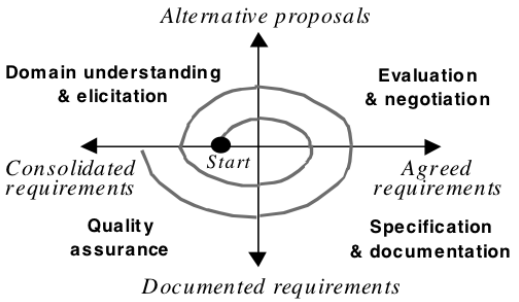
- Functional requirements:
  - What the system does: the interactions between the system and its environment
  - Should be as independent as possible from implementation
- Non-functional requirements:
  - Observable aspects of the system that are not directly related to functional behavior
    - e.g. performance, reliability
- Safety/security requirements ("shall not" properties)
  - A kind of nonfunctional requirement
  - Behavior the system must never exhibit
    - e.g. it must be impossible to apply reverse thrust in mid-flight
- Constraints (Pseudo requirements)
  - Imposed by the client or environment in which the system operates
  - Often concern the technology to be used (language, operating system, middleware etc.)

What do Requirements Engineers do?



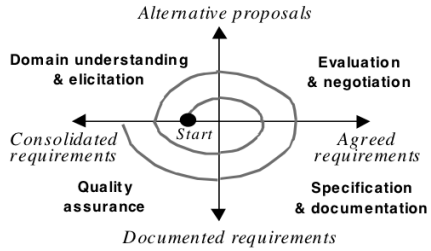
Focus of today's lecture

Domain Understanding and Elicitation



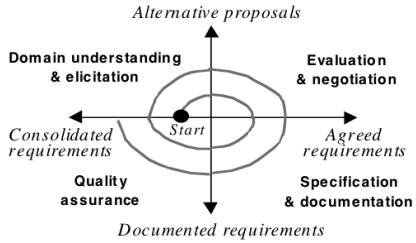
- Domain Understanding
  - Identify the problem / opportunity
  - Understand the organizational context
  - Identify stakeholders
  - Specify the scope of the software system
- Requirements Elicitation
  - Elaborate stakeholders' goals
  - Identify alternative options for satisfying these goals
  - Identify organizational / technical constraints and assumptions
  - Define typical scenarios illustrating desired interactions between the system and its environment
  - Articulate the requirements the system should meet in order to conform to all of the above

Evaluation and Negotiation



- Evaluation Tasks
  - Identify conflicting concerns and resolve them
    - Conflicts often arise from multiple viewpoints and different expectations.
  - Identify and assess risks associated with the system
    - Budget, schedule, etc.
  - Compare the alternative options identified during elicitation and select best options
  - Prioritize the requirements
    - give weight to requirements that are essential to the system
    - drop lower-priority requirements that would together exceed budgets and deadlines
- All the above involve some kind of negotiation to arrive at a consensus
- Output is a preliminary requirements document

Specification and Documentation



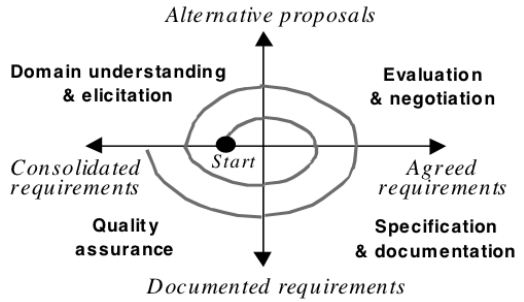
- Goal
  - Detailing, structuring, and documenting the agreed characteristics of the system as they emerge from the evaluation and negotiation activity
- Contents
  - A detailed and precise description of the following:
    - Objectives
    - Concept definitions
    - Relevant domain properties
    - Responsibilities
    - System requirements
    - Software requirements
    - Environment assumptions

Requirements Quality Assurance (QA)

➜ Goal:

➥ Quality assurance is aimed at checking that the items specified in a Requirements Document (RD) meet the desired qualities attributes

➢ Completeness, consistency, adequacy



➜ Why is Requirements QA important?

➥ Because cost of requirements failures is very high

"Finding and fixing a software problem after delivery is often 100 times more expensive than finding and fixing it during the requirements and design phase." [Boehm and Basili]

➜ Remainder of lecture is about Requirements QA

Requirements Quality Attributes

- Completeness
  - The requirements must be sufficient to ensure that the system will satisfy all its objectives
- Consistency
  - The requirements must be satisfiable when taken together, i.e. they must be compatible
- Unambiguity
  - The requirements must be specified in a way that precludes different interpretations.
- Measurability
  - The requirements must be formulated at a level of precision that enables analysts to evaluate alternative options, and to test or verify whether an implementation satisfies them.
- Relevance
  - The requirements must each contribute to the satisfaction of one or several objectives underpinning the system

Requirements Quality Attributes (continued)

- Feasibility
  - The requirements must be realizable in view of the budget, schedule, and technology constraints
- Comprehensibility
  - The formulation of requirements must be comprehensible by the people who need to use them.
- Good structuring
  - The requirements document should be organized in a way that highlights the structural links among its elements
- Modifiability
  - It should be possible to revise, adapt, extend, or contract the requirements document through modifications that are as local as possible
- Traceability
  - The context in which an item of the requirements document was created, modified, or used should be easy to retrieve. Such context should include the rationale for creation, modification, or use.

Common Requirement Defects

➜ Major Issues

Omission	No requirement about expected state of train doors in case of emergency stop.
Contradiction	... "train doors must always be kept closed between stations" ... *and elswhere:* ... "train doors must be opened once a train is stopped after emergency signal" ...
Inadequacy	"If a book copy has not been returned one week after the third reminder has been issued, the negligent borrower shall be notified that she has to pay a fine of $*X*" *rather than* "If a book copy has not been returned one week after the third reminder has been issued, a fine of $*x* shall be retained on the borrower's registration deposit and a notification will be sent to the borrower."
Ambiguity	"Train doors shall be opened as soon as the train is stopped at some platform" *(possible interpretations:)* "... the front of train is (stopped) at platform" *or* "... the whole train is (stopped) at platform" *?*
Unmeasurability	"Information panels inside trains shall be user-friendly."

Common Requirements Defects

➜ Minor Issues

Noise	"Every train car will be equiped with a software-controlled information panel together with non-smoking signs posted on every window."
Overspecification	"The setAlarm method must be invoked upon receipt of a stopAlarm message."
Unfeasibility	"The meeting scheduler will also make travel arrangements such as flight, car, and hotel reservations for every participant who needs to travel to attend the meeting."
Unintelligibility	A requirement statement containing five acronyms.
Poor structuring	Intertwining of book acquisition and loan management aspects.
Forward reference	Multiple uses of the concept of "participating in a meeting" in the requirements document and then, several pages further, the definition: "A person *participates* in a meeting if she attends that meeting from beginning to end."
Remorse	After multiple uses of the undefined concept of "participating in a meeting", the last one directly followed by an incidental definition between brackets such as: (a person *participates* in a meeting if she attends that meeting from beginning to end)
Poor modifiability	Use of fixed numerical values for quantities throughout the requirements document (e.g., for *maximum loan period*, *meeting notification deadline* or *train speed thresholds*), when such values are subject to change over time or from one variant to another.
Opacity	A requirement such as: "the commanded speed of a train must always be at least 7 mph above its physical speed" without any contextual information about the origin and rationale of this requirement, and its impact on other requirements.

Techniques for Requirements QA

- Prototyping
  - Partial implementation for validation with customers
- Inspections, reviews, walkthroughs
  - Independent inspectors search for defects and recommend appropriate actions on agreed defects
- Simulation
  - Simulates the environment of a system and checks the appropriateness of specified behaviors
- Formal checks
  - Completeness and sanity checks
  - Model checking and theorem proving

Prototyping

- Definitions
  - "A software prototype is a partial implementation constructed primarily to enable customers, users, or developers to learn more about a problem or its solution." [Davis 1990]
  - "Prototyping is the process of building a working model of the system" [Agresti 1986]
- Approaches to prototyping
  - Presentation Prototypes
    - explain, demonstrate and inform – then throw away
    - e.g. used for proof of concept; explaining design features; etc.
  - Exploratory Prototypes
    - used to determine problems, elicit needs, clarify goals, compare design options
    - informal, unstructured and thrown away
  - Breadboards or Experimental Prototypes
    - explore technical feasibility; test suitability of a technology
    - Typically no user/customer involvement
  - Evolutionary (e.g. "operational prototypes", "pilot systems"):
    - development seen as continuous process of adapting the system
    - prototype is an early deliverable, to be continually improved.

Inspections, Reviews, Walkthroughs

- Note: these terms are not widely agreed
  - Formality
    - informal: from meetings over coffee, to team get-togethers
    - formal: scheduled meetings, prepared participants, defined agenda, specific format, documented output
  - Management reviews
    - E.g. preliminary design review, critical design review, …
    - Used to provide confidence that the design is sound
    - Attended by management and sponsors (customers)
    - Usually a "dog-and-pony show"
  - Walkthroughs
    - developer technique (usually informal)
    - used by development teams to improve quality of product
    - focus is on finding defects
  - (Fagan) Inspections
    - a process management tool (always formal)
    - used to improve quality of the development process
    - collect defect data to analyze the quality of the process
    - written output is important
    - major role in training junior staff and transferring expertise

Benefits of Formal Inspections

- Formal inspection works well for programming:
  - For applications programming:
    - more effective than testing
    - most reviewed programs run correctly first time
      - compare: 10-50 attempts for test/debug approach
  - Data from large projects
    - error reduction by a factor of 5; (10 in some reported cases)
    - improvement in productivity: 14% to 25%
    - percentage of errors found by inspection: 58% to 82%
    - cost reduction of 50%-80% for V&V (even including cost of inspection)
  - Effects on staff competence:
    - increased morale, reduced turnover
    - better estimation and scheduling (more knowledge about defect profiles)
    - better management recognition of staff ability
- These benefits have been shown to apply to requirements inspections too

Inspection Constraints

- **Size**
  - enough people so that all the relevant expertise is available
  - min: 3 (4 if author is present)
  - max: 7 (less if leader is inexperienced)
- **Duration**
  - never more than 2 hours; concentration will flag if longer
- **Output**
  - all reviewers must agree on the result: accept; re-work; re-inspect;
  - all findings should be documented
    - summary report (for management); detailed list of issues
- **Scope**
  - focus on small part of a design, not the whole thing
- **Timing**
  - Examines a product once its author has finished it
    - not too soon: product not ready - find problems the author is already aware of
    - not too late: product in use - errors are now very costly to fix
- **Purpose**
  - Remember the biggest gains come from fixing the process
  - collect data to help you not to make the same errors next time

Inspection Guidelines

- Prior to the review
  - schedule Formal Reviews into the project planning
  - train all reviewers
  - ensure all attendees prepare in advance
- During the review
  - review the product, not its author
  - keep comments constructive, professional and task-focused
  - stick to the agenda
  - leader must prevent drift
  - limit debate and rebuttal
  - record issues for later discussion/resolution
  - identify problems but don't try to solve them
  - take written notes
- After the review
  - review the review process!

Inspection Guidelines

- Possibilities
  - specialists in reviewing (e.g. QA people)
  - people from the same team as the author
  - people invited for specialist expertise
  - people with an interest in the product
  - visitors who have something to contribute
  - people from other parts of the organization
- Exclude
  - anyone responsible for reviewing the author
    - i.e. line manager, appraiser, etc.
  - anyone with known personality clashes with other reviewers
  - anyone who is not qualified to contribute
  - all management
  - anyone whose presence creates a conflict of interest

Structuring the Inspection

- Can structure the inspection in different ways
  - Free mode
    - Rely on expertise of the reviewers
  - Checklists
    - uses a checklist of questions/issues
    - checklists tailored to the kind of document
  - Active reviews (perspective-based reading)
    - each reviewer is given a specific process to follow for defect search and reads for a specific purpose
    - effectively different reviewers take different perspectives
- The differences may matter
  - Some studies indicate that:
    - active reviews find more faults than free mode or checklist methods
    - no effective difference between free mode and checklist methods
    - the inspection meeting might be superfluous!

Inspection Checklists

➜ Example

Omission	*Is this concept precisely defined somewhere? Is this acronym defined? Are these definitions summarized in the glossary of terms?* *Is this objective operationalized through specific requirements? Are those requirements sufficient to ensure this objective? Is there any hidden assumption required in addition for this?* *Is the rationale for this requirement (or assumption) made explicit somewhere ?* *If this requirement or assumption relates to another, is the latter specified somewhere?*
Contradiction	*Is this statement consistent with the system objectives and constraints?* *Is this statement consistent with other related statements?*
Inadequacy	*Does this RD item formulate what stakeholders really expect?*
Ambiguity	*Can this statement be interpreted differently in different relevant contexts or by readers from different backgrounds? What are the possible interpretations then ?* *Are there other statements using this term with different meanings?*
Unmeasurability	*Is there a fit criterion associated with this quality requirement?* *Is this fit criterion stated in terms of measurable quantities and measurement protocol?* *Can test data be derived from this statement to test that the implementation meets it?* *Is this statement stated in a way that discriminates it from alternative options?*

Inspection Checklists

➜ Example (continued)

Noise	*Is this statement relevant to system objectives and constraints?* *Does the negation of this statement make any sense? (Otherwise the statement is a tautology.)* *Has this already been said without any reason for redundancy?* *Are there any other statements using this concept under different terms?*
Overspecification	*Does this statement entail a premature design choice? Would there be alternative sensible choices?*
Unfeasibility	*Is this RD item implementable in view of infrastructure, budget, or timing constraints?*
Unintelligibility	*Will this statement be comprehensible by anyone who should use it? If not, why?*
Poor structuring	*Is the structuring rule for organizing those RD sections apparent? Is there any other structuring rule that would make them easier to understand?* *Is there any RD item related to this and described in some unrelated part of the RD ?* *Is this RD item covering unrelated requirements?* *Is this RD item mixing requirements and assumptions altogether?* *Is this RD item mixing requirements and domain properties?* *Is there any unnecessary intermixing of functional and non-functional requirements?*
Forward reference	*Is this concept, so far undefined, defined somewhere after?*
Remorse	*Has this concept been used already before this definition?*
Poor modifiability	*Would any change to this RD item need to be propagated throughout major portions of the RD?*
Opacity	*Are there any inderdependent RD items whose dependencies are not made visible?*

Simulation

- Definition
  - Mimicking possible behaviors of the environment and executing a model of the software system to respond to these events
  - Often accompanied by animation
    - a visual representation that shows how the system evolves as the model is being executed
- Prerequisite
  - An (abstract) executable model of the system needs to be built
- Types of visualization
  - Textual: input events are entered as textual commands; model reactions are displayed as execution traces.
  - Diagrammatic: input events are entered by event selection among those applicable in the current state; model reactions are displayed as tokens progressing along the model diagrams
  - Domain-specific visualization: input events are entered through domain -specific control devices displayed on the screen; model reactions are displayed as new values on domain-specific control panels

Formal Checks

➜ A wide range of mathematically defined checks that can be automated by tools

➥ The specification must be formal.

➜ Example 1

➥ *Checking disjointness of input cases*

➢ *For every two cases C1, C2 we must have* $C1 \wedge C2 = \text{false}$

Mode	Conditions	
Stopped	AtPlatform OR Emergency	NOT AtPlatform
MovingOK, TooFast	False	True
DoorsState	Open	Closed

(AtPlatform OR Emergency) AND NOT AtPlatform = (AtPlatform AND NOT AtPlatform) OR (Emergency AND NOT AtPlatform)
= **false** OR Emergency AND NOT AtPlatform
= Emergency AND NOT AtPlatform
≠ **false**

Formal Checks (continued)

- Example 2
  - *Checking completeness of input cases*
    - *For C1, C2, … Cn we must have:* $C1 \vee C2 \vee \cdots \vee Cn = \text{true}$

Mode	Conditions	
Stopped	AtPlatform OR Emergency	NOT AtPlatform AND NOT Emergency
MovingOK, TooFast	False	True
DoorsState	Open	Closed

(AtPlatform OR Emergency) OR (NOT AtPlatform AND NOT Emergency) = AtPlatform OR Emergency OR NOT AtPlatform
= **true** OR Emergency
= **true**

- *A wide variety of techniques use model checking and theorem proving*
  - *Model checking will be covered in a future lecture*

References

- Lecture contents adapted from:
  - Requirements Engineering From System Goals to UML Models to Software Specifications by Axel van Lamsweerde
    - Chapter 5 Requirements Quality Assurance
  - Course slides from the Requirements Engineering course at the University of Toronto by Steve Easterbrook
    - http://www.cs.toronto.edu/~sme/CSC2106S/

- More Resources
  - [IEEE Std 830-1998] IEEE Recommended Practice for Software Requirements Specifications -Description
  - [Boehm&Basili] Software Defect Reduction Top 10 List
  - [Blum] Software Engineering: A Holistic View