

Chapter 4: Software Requirements Elicitation

Dr. Michael F. Siok, PE, ESEP

UT Arlington, Computer Science and Engineering

CSE 5324: Software Engineering: Analysis, Design, and Testing



Key Takeaway Points



- Requirements are capabilities that the system must deliver
- *“The hardest single part of building a software system is deciding precisely what to build—i.e., the requirements.”*
 - Frederick P. Brooks, Jr., The Mythical Man-Month
- Software requirements elicitation aims to identify the **real requirements** for the system – **this may not be the same as what the customer asked for!**

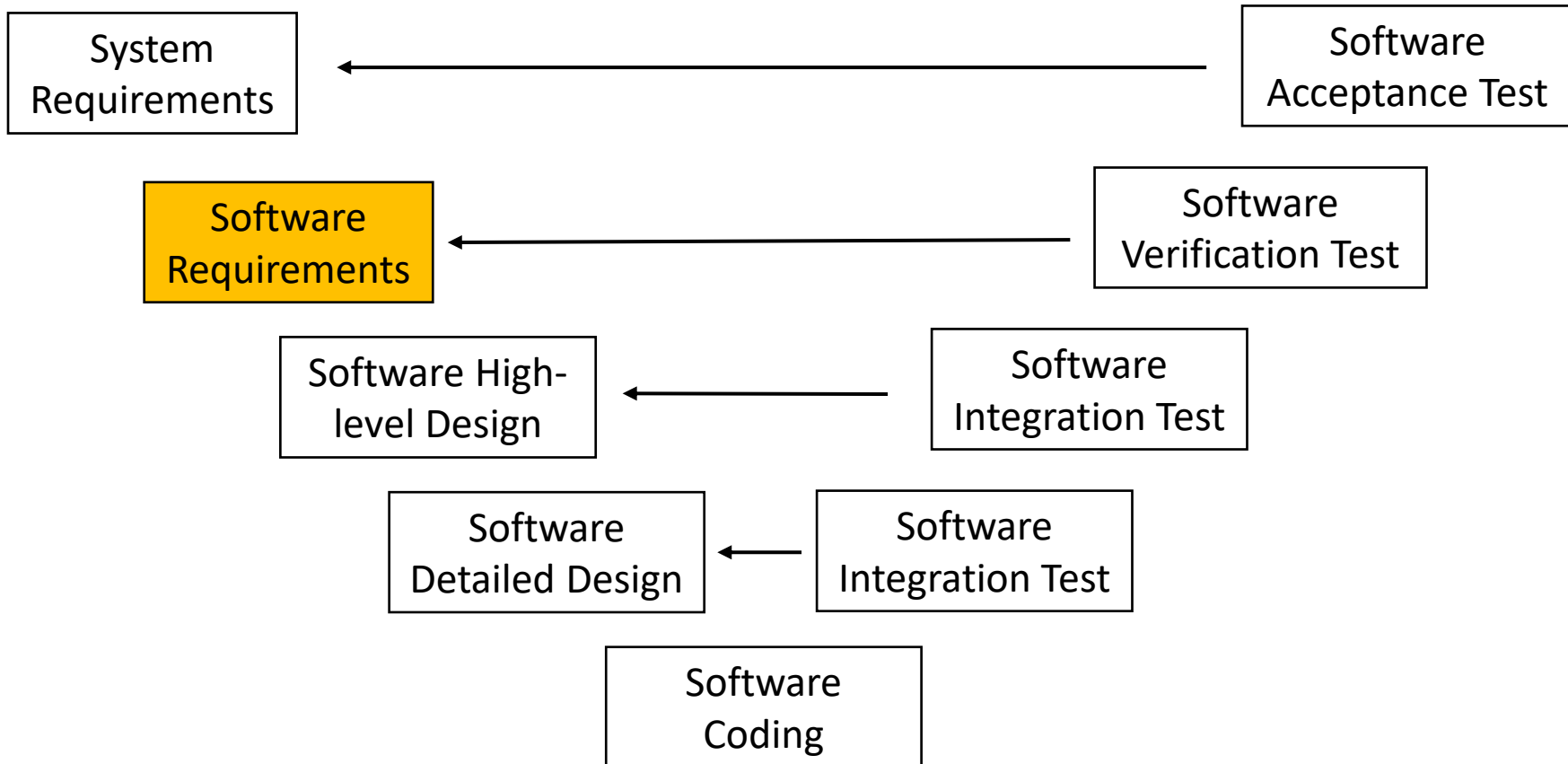
The Planning Phase

- The Planning Phase of our class Project SDLC has three (3) activities:
 - **Requirements Elicitation**
 - Deriving use cases from the requirements
 - Producing an iterative development plan
- This planning phase requires the software team to work closely with customer and users
 - Help them identify the needs of their business system
 - Help them determine the real requirements for the future system
 - Help them prioritize these requirements

Requirements Elicitation

- Requirements are *capabilities* that the system must deliver
 - Usually specified as functions, things the system must do
- Constraints are *restrictions on the solution space* of the software system
 - Typically reduces the number of design or implementation alternatives available
- Software Requirements are documented in a requirements specification
 - Software Requirements Specification (SRS) – *or equivalent*
 - Serves as part of ‘the contract’
 - *Software requirement analysis activities must assure that the specified requirements and constraints can be met and will satisfy the user needs*
 - Feasibility studies may be necessary
 - Modeling and/or simulation may be necessary
- Requirements elicitation is the process to identify and formulate the capabilities for the software system
 - Some may also call this *Requirements Analysis*

Requirements in the V-Life Cycle Model



Requirements Elicitation Activities

- Identifying problems and business needs
- Constructing analysis models to help understand problems and needs
- Formulating system and software requirements and documenting them
- Conducting feasibility studies
 - To lower product development risk (e.g., risk of using new technology, real-time constraints, innovative features)
- Checking requirements and models for desired properties such as correctness, consistency
 - Requirements can be modeled, system simulated, model checked for requirements consistency
- Specifying acceptance tests
 - To verify requirements are met
 - To validate that user needs are met with implemented requirements
- Formulating an iterative development plan



Importance of Requirements Elicitation

- The requirements specification is part of the contract, which bears legal consequences
 - Numerous lawsuits are related to systems not satisfying the requirements and constraints
- 37% of all software development problems are related to requirements
 - Standish Group, 1994 (ref. 92 in textbook)
- Poorly identified requirements significantly increase development costs and has led to the failure of many software projects

"The hardest single part of building a software system is deciding precisely what to build.

No other part of the conceptual work is as difficult as establishing the detailed technical requirements, including all the interfaces to people, to machines, and to other software systems.

No other part of the work so cripples the resulting system if done wrong.

No other part is more difficult to rectify later."

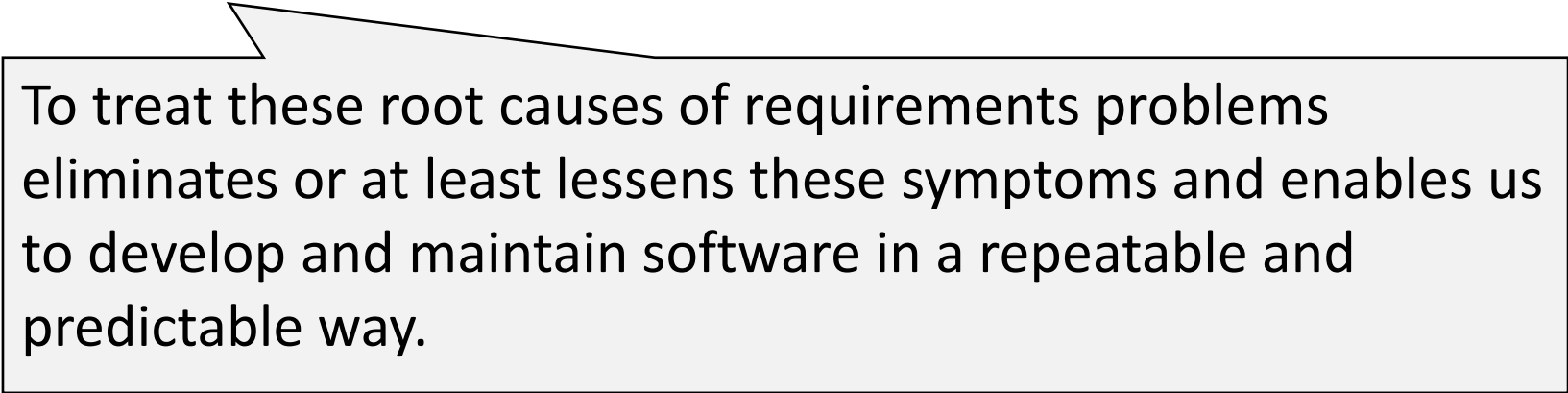
-- Frederick M. Brooks

General Problems with Requirements Elicitation

- Lack of the right expertise (i.e., software engineers, domain experts, etc.)
 - Often requires many different people within the business to assemble a complete picture of the needs, processes
 - People may only know (*or they think they know*) about “their part” of the process
 - Very incomplete picture
- Initial ideas of functions and features are often incomplete, wildly optimistic, and firmly entrenched in the minds of the people leading the acquisition process
 - Customers and users have difficulty expressing what is really needed
 - Developers try to capture these needs with written requirements (these are also difficult to express precisely)
- Non-functional requirements and constraints are often forgotten or worse . . . found out later!
- Difficulty of using complex tools and diverse methods associated with requirements gathering may negate the anticipated benefits of a complete and detailed approach
 - Lack of a common background between customers, users, and developers hampers communications

Root Causes of Problems

- An insufficient requirements specification and its '*ad hoc*' management
- Ambiguous and imprecise communication
- Brittle architectures
- Overwhelming complexity
- Undetected inconsistencies in requirements specification
- Poor and insufficient understanding of testing
- Subjective assessment of project status
- Failure to attack risks
- Uncontrolled change propagation



To treat these root causes of requirements problems eliminates or at least lessens these symptoms and enables us to develop and maintain software in a repeatable and predictable way.

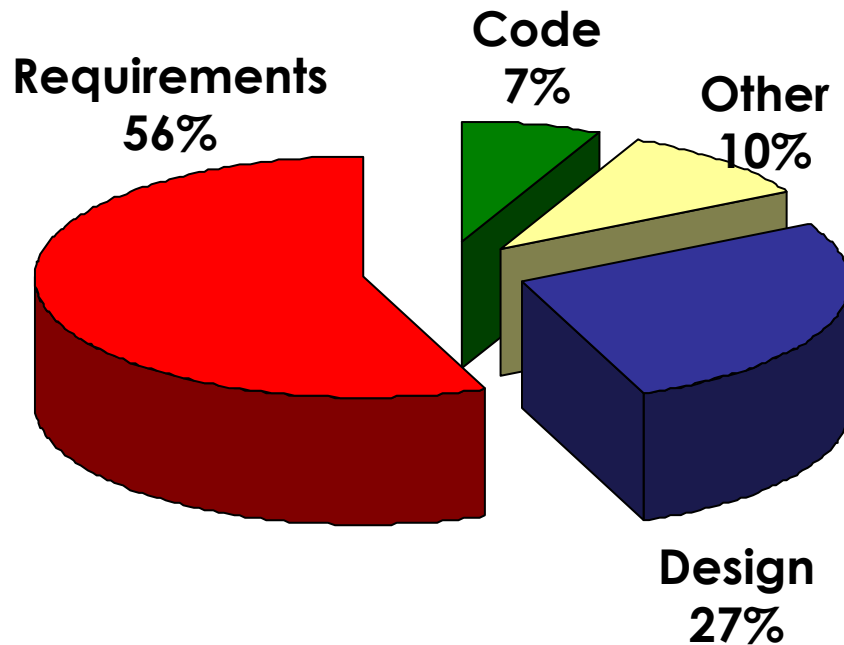
Statistics from NIST Report

- NIST (**National Institute of Standards and Technology**) has published a comprehensive (309 pages) report on project statistics and experiences based on data from a large number of software projects¹
 - 70% of the defects are introduced in the **specification** phase
 - 30% of defects are introduced **later** in the technical solution process
 - Only 5% of the specification inadequacies are corrected in the specification phase
 - 95% of the defects are detected later in the project or even after delivery where the cost for correction on average is 22 times higher compared to a correction directly during the specification effort
 - The NIST report concludes that extensive testing is essential, however testing detects the dominating specification errors very late in the process
 - *More expensive to fix the later defect is found*

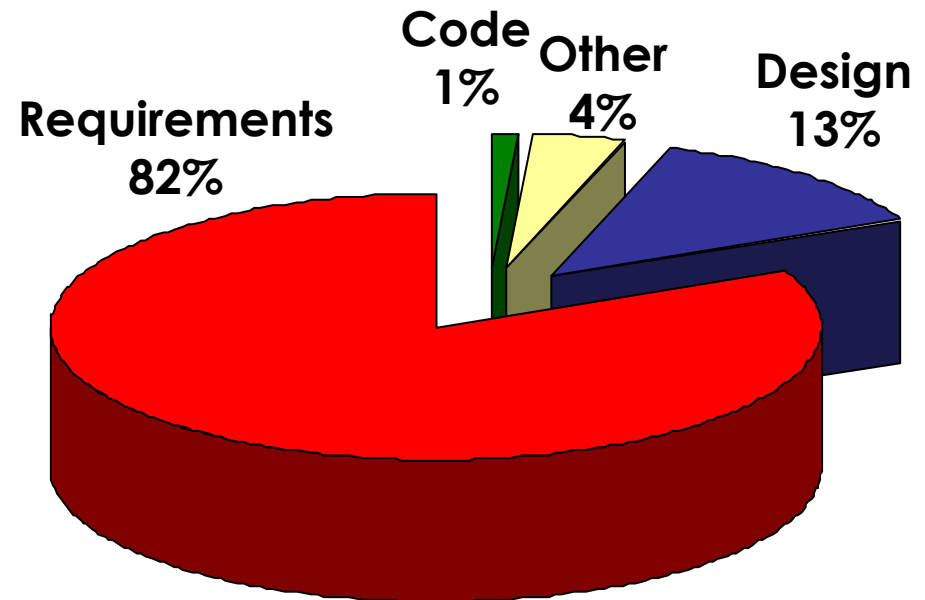
[1] <https://www.nist.gov/system/files/documents/director/planning/report02-3.pdf>

Why Focus on Requirements ?

- Distribution of Defects



- Distribution of Effort to Fix Defects



Challenges of Requirements Elicitation

"The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is as difficult as establishing that detailed technical requirements, including all the interfaces to people, to machines, and to other software systems. No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later."

Frederick Brooks 1987

Challenges of Requirements Elicitation

As an analyst, I need to know . . . what do you want?



I want you to design the software for me.



But what do you want to do with the software?



I don't know until you tell me what the software can do.



Well, I can design the software to do anything!



Can you design the software to tell you my requirements?!



Funny, but really not so funny!

Communication Barrier - Misunderstanding



Challenges of Requirements Elicitation

- Software development, in general, is a wicked problem
- There are communication barriers between the team and the customer and users
 - The team does not typically 'know' the application domain or the needs of the customer and users
 - The customer and users do not know what the software can do for them
- The importance and challenges of 'requirements elicitation' are often under-estimated
 - Takes time and effort; must be planned, budgeted, and managed
- Nonfunctional requirements are not identified or are under-stated
- Requirements change throughout the software lifecycle and must be accommodated and managed



The Importance of Requirements

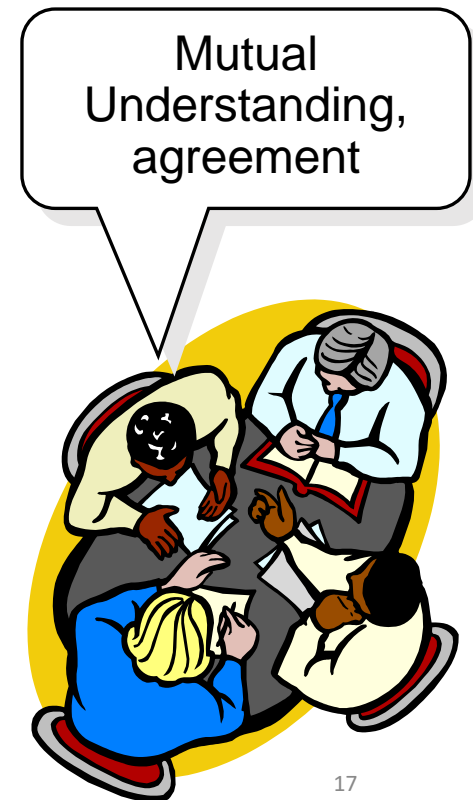
- Requirements problems are the single biggest cause of project problems
- Requirements define what is to be done, how well, and under what constraints
 - Get the requirements wrong and the design, hardware, and software will be wrong
- Requirements drive . . .
 - Cost - Design - Schedules - Skills required - Verification plans - Operational procedures - - - - basically . . . everything.
- *It is amazing how many teams begin to solve a problem before there is agreement on what the problem is !*
 - Requirements and their associated constraints and assumptions quantify the problem to be solved
 - ***Requirements establish how project success will be determined***

What is a Requirement?

- Statement of some THING wanted and needed
OR
A characteristic of some THING wanted and needed

A function . . .
A capability . . .

- A requirement is also...
 - A *Contractually Binding* Statement
 - Documentation of the *Problem Space*
 - The *Means* We Use to Communicate
- Usually expressed formally using the verb, “*Shall*”
 - The ‘x system’ **shall** provide
- Use of the words “must” or “will” express only desired features which may not necessarily be required nor verified during acceptance test
 - These are “*Desirements*,” not requirements



Types of Requirements

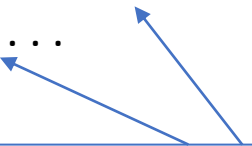
- Functional requirements – statements of information processing capabilities that the software system must possess
- Non-functional requirements – define criteria used to specify and/or evaluate ‘how well’ or ‘to what extent’ the system operates or performs. These criteria may include:
 - Performance requirements (these may be functional and/or non-functional and specify how much, how fast . . . e.g., throughput, response time)
 - Quality requirements (how good, defect density limits, expected fielded defect rate)
 - Safety requirements (these may be functional and/or non-functional and specify ‘under what conditions,’ unwanted states, unwanted consequences)
 - Security requirements (these may be functional and/or non-functional)
 - Interface requirements (how entities communicate, constraints (i.e., how much, how fast, how often, how much, etc.), protocols, APIs)
 - Software Reliability requirements (how well and under what conditions, defect density limits, expected fielded defect rate)
 - Dependability requirements (how well and under what conditions)

Examples of Functional Requirements

- For a car rental system:
 - The system *shall* allow a potential customer to query information and availability of rental cars using various combinations of search criteria including make, model, from date, to date, price range, and class (small size, medium size, large size, and luxury cars).
- For a study abroad system:
 - The system *shall* provide interactive as well as batch-processing means for an OIE (Office of International Education) staff to enter exchange program information into the database.

Examples of Non-Functional Requirements

- Workload:
 - The system shall be capable of handling a typical workload of 10,000 (ten thousand) inquiries at the same time
- Response time:
 - The system's response time shall not exceed 3 (three) seconds under the typical workload
 - Typical workload defined . . .
- Development Methodology (sometimes specified by the customer)
 - The System/software shall use the Object Oriented Methodology
 - The Software shall be coded in Java



These requirements are also constraints as they represent “restrictions on the design and/or solution space alternatives”.

Examples of Quality Requirements

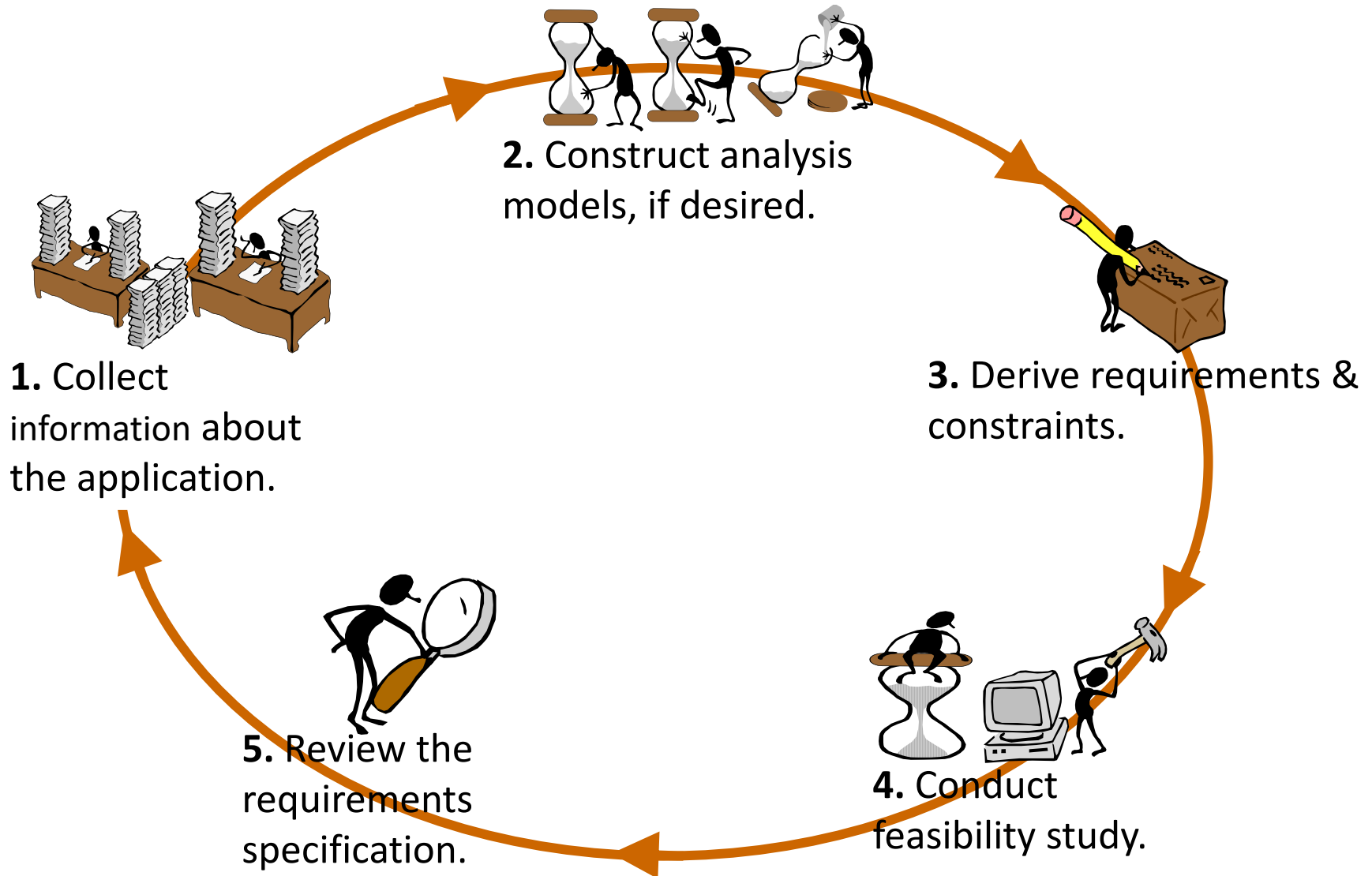
- Security:
 - The system must protect the contents of the website from malicious attacks and protect the privacy of its users.
- Platform independence:
 - The system must run on Windows 2000 and later, Unix, Linux, and Mac and support popular relational database management systems (RDBMS) including Oracle, SQL Server, Access, and MySQL.



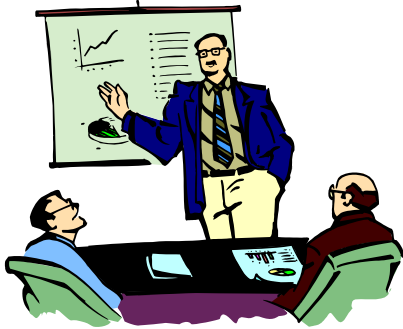
Examples of Quality/Interface Requirements

- User friendliness:
 - The system must provide a user interface that conforms to commonly used web-application user interface look-and-feel and man-machine interaction conventions.
 - *This requirement would usually include a reference to a specific User Interface (UI) standard or guide*
- User Interface (UI):
 - The system must support the following user interfaces:
 - web-based
 - stand-alone
 - voice
 - cellular phone

Requirements Elicitation Steps



(1) Information Collection Techniques



Customer presentation



Business forms



Operating Procedures



Regulations & Standards

Literature survey



Stakeholder survey



User interviewing



Writing user stories

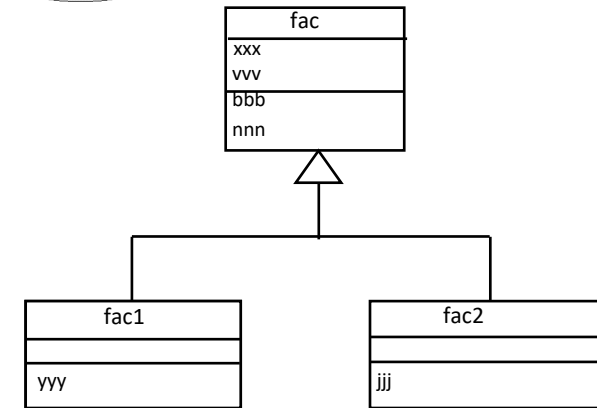
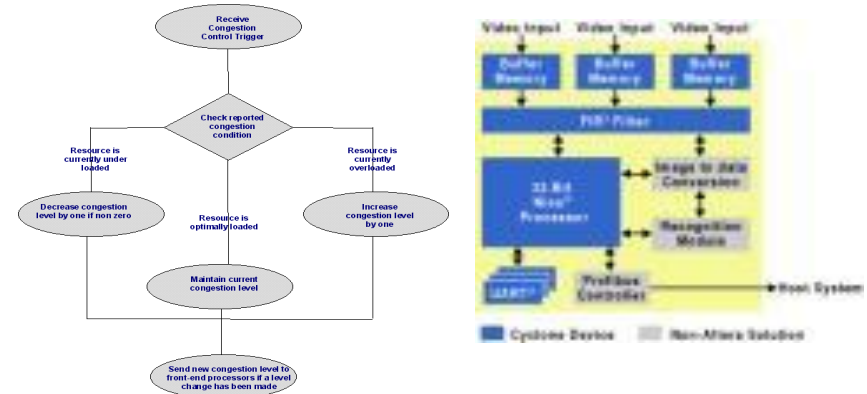
Focuses of Information Collection Activities

- What is the business, the current business situation, and how does it operate?
- What is the system's environment or context?
- What are existing business processes, their input and output, and how do they relate to each other?
- What are the problems with the current system?
- What are the business or product goals?
- What do the customer and users want, and what are their business priorities?
- What are the quality, performance, and security considerations?
- Who are the users of the current and future systems, respectively?

(2) Constructing Analysis Models



Intuitive models



Formal or informal models

Purpose: to aid in understanding the application, requirements, and constraints.

Businesses in Different Domains



Finance



Insurance



Health Care



Transportation



Defense



Telecommunication



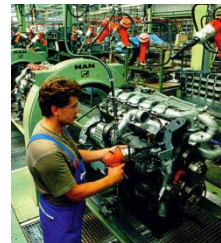
Retailing



Energy



government



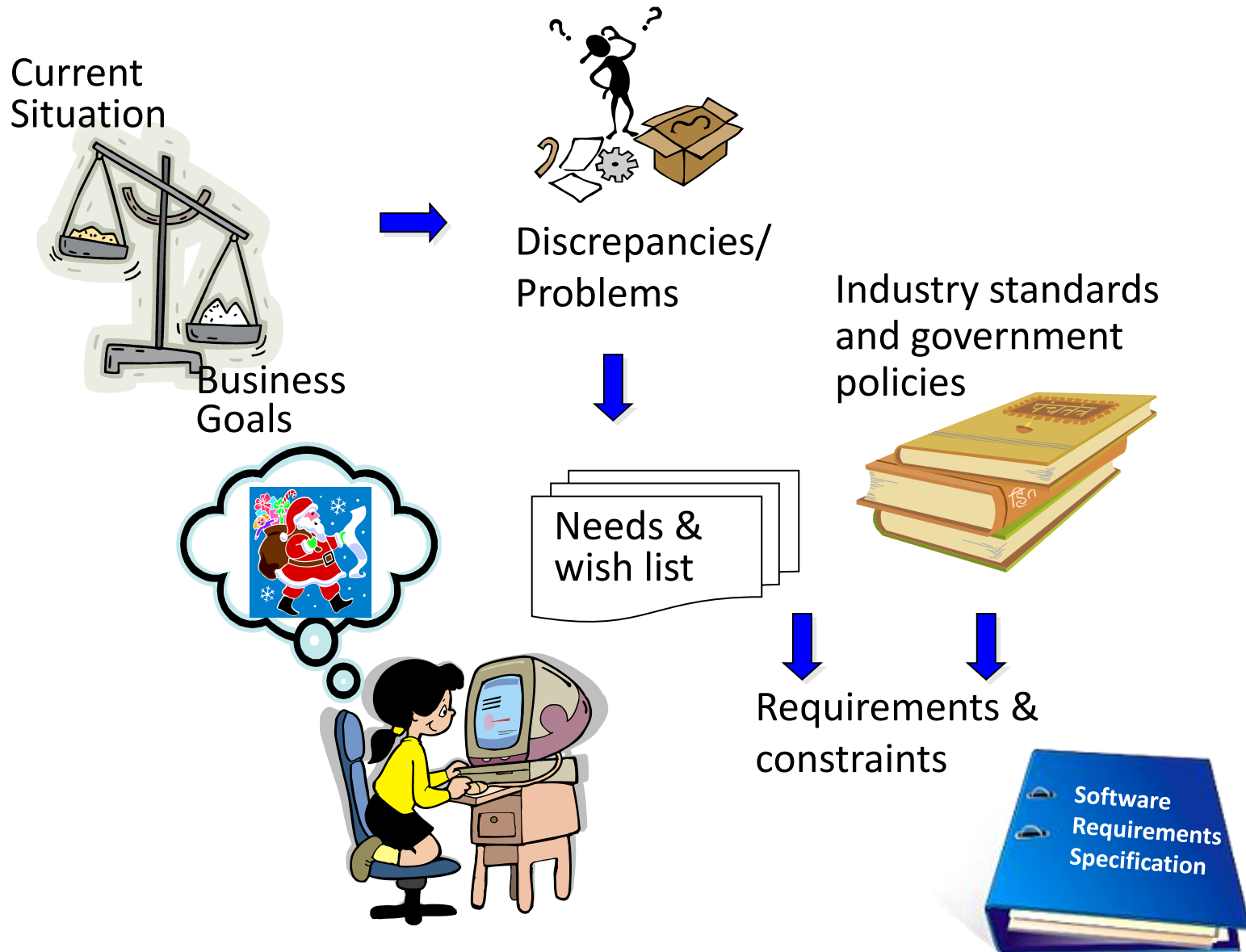
Manufacturing



Education

and more

(3) Deriving Requirements and Constraints



Requirements Specification

1. Introduction to Document

- 1.1 Purpose of Product
- 1.2 Scope of Product
- 1.3 Acronyms, Abbreviations, Definitions
- 1.4 References
- 1.5 Outline of the Rest of the SRS

2. General Description of Product

- 2.1 Context of Product
- 2.2 Product Function
- 2.3 User Characteristics
- 2.4 Constraints
- 2.5 Assumptions and Dependencies

3. Specific Requirements

- 3.1 External Interface Requirements

3.1.1 User Interfaces

3.1.2 Hardware Interfaces

3.1.3 Software Interfaces

3.1.4 Communication Interfaces

3.2 Functional Requirements

3.2.1 Class 1

3.2.2 Class 2

3.2.3 ...

3.3 Performance Requirements

3.4 Design Constraints

3.5 Quality Requirements

3.6 Other Requirements

4. Appendices

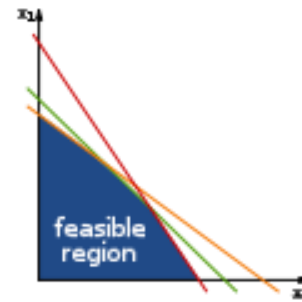
IEEE SRS Standard by Objects, 1998

Requirements Specification (cont'd)

- Want to derive requirements and constraints from information collected
 - Identify “needs” – these are “must haves”
 - Identify “wants” -- these are “would like to have” (usually for free)
 - These are not really requirements!
 - *Customer is not paying for these, but would like to have them*
- Derive or reword to specify system capabilities (i.e., requirements)
 - ##### The xyz system *shall* . . .
 - ##### The xyz system *shall* . . .
 -
- Each requirement has a unique number, for tracking
- Once list of requirements (capabilities) are specified, need to prioritize
 - Typically according to business needs or business value
 - Could be prioritized by when needed, if using agile development
 - Usually prioritization activity includes customer and users
- Requirements are typically specified using an identified and defined standard
 - Industry or company standard
 - Document or database

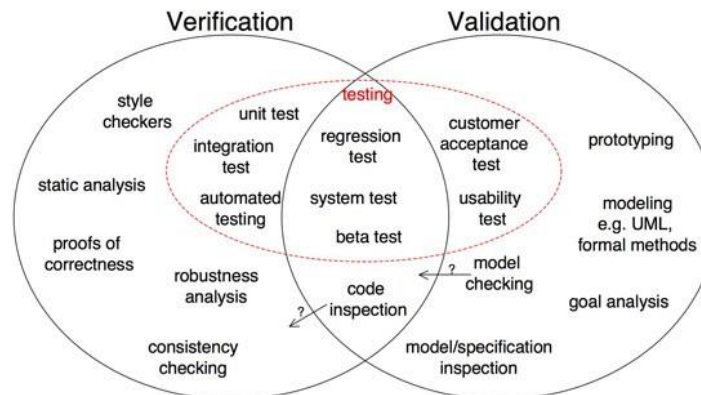
(4) Conducting Feasibility Study

- Not all projects are '*practically do-able*' with technology, time, and/or resource constraints
- A Feasibility study aims at determining if the project is do-able under a set of given/stated constraints
- Feasibility study in Requirements Engineering (RE) is concerned with:
 - Feasibility of the functional, performance, nonfunctional, and/or quality constraints
 - Adequacy of the technology
 - Timing and cost constraints
 - Constraints imposed by customer, industry, company, and/or government agencies



Requirements Verification and Validation

- Verification: Are we building the **product right**?
 - Are we building the product in the right way?
 - It concerns the product building process to result in a product that implements the requirements
- Validation: Are we building the **right product**?
 - Are we building the product that the customer wants/needs to solve their problem (or implement their business need)?
 - It concerns the correctness of the product being built to solve the customer's specific problem



What is the right system to build ?



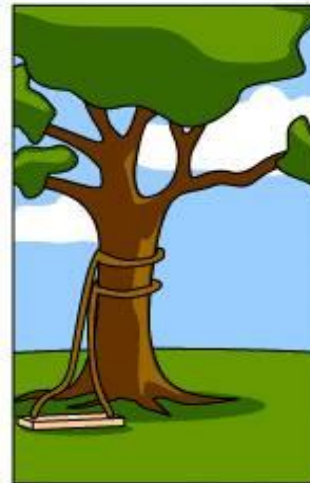
How the customer explained it



How the Project Leader understood it



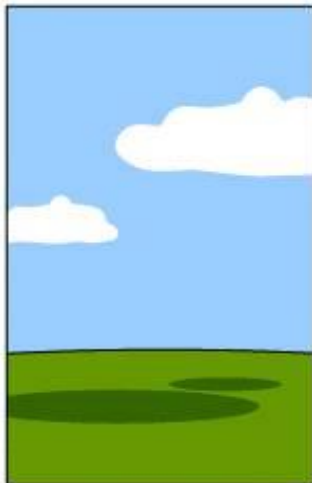
How the Analyst designed it



How the Programmer wrote it



How the Business Consultant described it



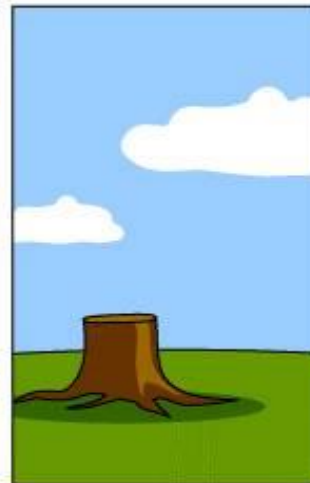
How the project was documented



What operations installed



How the customer was billed



How it was supported



What the customer really needed

(5) Review Requirements

- Technical review is a review performed by the technical team
- Review techniques include:
 - **Peer review** - peers or colleagues perform informal “desktop reviews” sometimes guided by a review questionnaire
 - **Walkthrough** - the analyst explains each requirement while the reviewers examine it and raise any doubts/questions within the technical meeting
 - **Inspection** - inspector is guided by a checklist of commonly encountered problems in SRS during desk review (e.g., incompleteness, duplicate definition, inconsistency, etc.). Follow-up formal meeting is scheduled to collect defects from all the inspectors using a reading technique during the meeting
- In every case, there is a requirement for a period of time between which the review materials are available and the review meeting is held – to allow for the pre-review
 - ... where most of the errors are found

(5) Review Requirements (cont'd)

- Additional reviews (these are validation activities)
 - **Expert review** -- means review of the requirements specification by domain experts, gray-beards. These people know the domain rules, guidelines, behaviors, policies, standards, and regulations and can offer detailed review relative to this product knowledge gained over their years of experience.
 - **Customer/user reviews** -- are performed by involving the customer and/or users of the system. These people can offer review of the functions, constraints, and identify unstated or forgotten requirements or constraints, including company and industry policies and procedures, using a directed formal or informal review technique.



Requirements Defects: The Problem

- Many Software Requirements Specifications (SRS) are filled with badly written requirements
 - Poor requirements cannot lead to excellent software
- Few software developers are educated about writing quality requirements
 - Not many examples of good requirements available to learn from
 - Few projects have good requirements to share
 - Few companies are willing to place their product specifications in the public domain as examples
- No general formulaic way to write good requirements – largely a matter of experience
 - However, the International Council on Systems Engineering ([INCOSE](#)) provides a guide for writing requirements reference
 - Available to INCOSE members
- It can be vastly more expensive to fix defective requirements in later stages of development or . . . when the customer calls

Quality Requirement Characteristics

- Correct
- Feasible
- Necessary
- Prioritized
- Unambiguous
- Complete
- Consistent
- Traceable



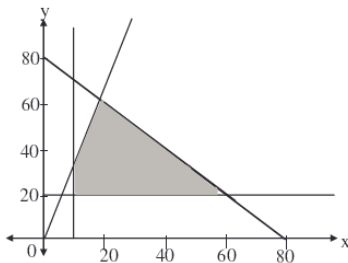
Correct Requirement

- Accurately describes functionality to be delivered
- Reference for correctness – source of requirement
 - Actual customer
 - Higher-level system requirements specification
- Need user representatives to determine correctness of user requirements
 - Inspection of requirements
 - Prevent developers from “*guessing*”



Feasible Requirement

- Can successfully implement requirement within known capabilities and limitations of the system environment
- Developer should work with requirements analysts or marketing personnel throughout elicitation
 - Provides reality check on technical feasibility of requirement
 - What can and cannot be done
 - What can be done only at excessive cost
 - What can be done only with other tradeoffs
- Example: *“The product shall switch between displaying and hiding non-printing characters instantaneously.”*
 - Not feasible – computers cannot do anything *instantaneously*



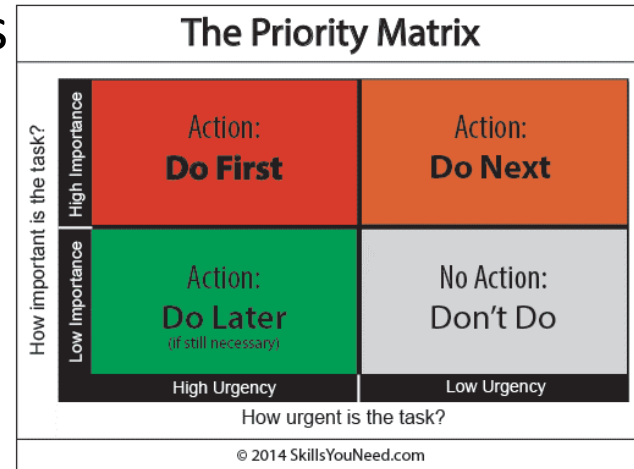
Necessary Requirement

- Something the customer **really needs**
- Something required for conformance to an external requirement, external interface, or standard
- Should be traceable back to its specific source
 - Source should be someone authorized to specify requirements
 - Untraceable requirements may not really be necessary requirements
- Estimates are that over 30% of most software requirements are capabilities that the user never asked for !
 - "Gold plating"



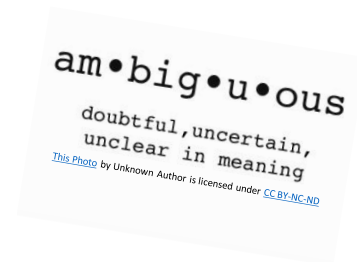
Prioritized Requirement

- Assign an implementation priority for a particular product release. E.g.,
 - Value provided to customer
 - Relative cost of implementation
 - Relative technical risk associated with implementation
- Assigned by customers or their surrogates
- Gives project manager more control
 - New requirements added during development
 - Budget cuts
 - Schedule and/or cost overruns
 - Team member departure
- Requirement priority levels example
 - **High priority** – must be in next product release
 - **Medium priority** – necessary, but can be deferred to later release
 - **Low priority** – nice to have, but may be dropped if insufficient time or resources



Unambiguous Requirement

- Reader should be able to draw only one interpretation
 - Multiple readers should arrive at same interpretation
 - This may be hard to do and should be checked
- Should be void of subjective words (*easy, simple, rapid, efficient, etc.*)
- Should be written in simple, straightforward language of user domain
- How to reveal possible ambiguity . . .
 - Formal inspections of requirements specification
 - Write test cases from requirements
 - Create user scenarios showing expected behavior
- Example: “*The HTML Parser shall produce an HTML markup error report which allows quick resolution of errors when used by HTML novices.*”
 - Ambiguous – the word, “*quick*”; the word “*novices*”

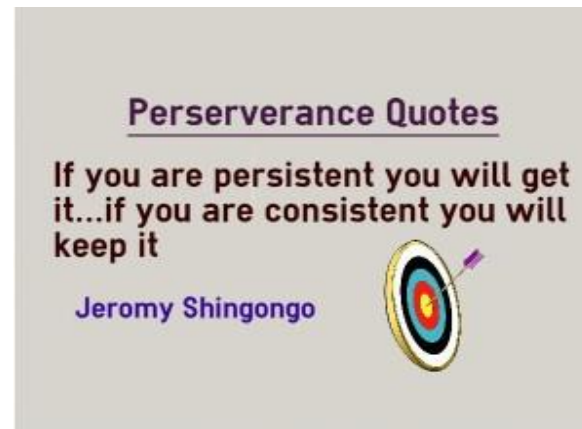


Complete SRS

- No requirements or necessary information should be missing
 - It is usually hard to spot missing requirements
- Ways to achieve completeness
 - *Use case method* – focus on user tasks rather than system functions during elicitation
 - *Graphical analysis models* – represent different views of requirements
- Flagging missing requirements information
 - TBD (“to be determined”) – these need to be resolved before product development
- Example: *“The product shall provide status messages at regular intervals not less than every 60 seconds.”*
 - Incomplete – what are the specific status messages? How are they supposed to be displayed to the user and really how often?

Consistent SRS

- Different requirements should not conflict
 - If you have conflicting requirements, that means that the system, when completed, cannot implement the full set of requirements!
- Disagreements among requirements must be resolved before development can proceed
- Ensure requirements modification does not introduce new inconsistencies



Traceable SRS

- Link each requirement back to its source
 - To a Higher-level system requirement
 - To a Use case
 - To a *Voice-of-the-customer* statement
- Link each requirement toward its development implementation
 - Design elements
 - Source code
 - Test cases
- Uniquely label each requirement
- Express each requirement in a structured, fine-grained way

Requirement	Functional design	Internal design	Code	Tests
Restaurant has two ordering stations	Mgmt screen #2	Page 45	Line 12485	34, 57, 63
A waiter may order from any station	Order screen	Page 19	Line 6215	12, 14, 34, 57, 92
Any customer at a table may request a separate check	Order screen	Page 39	Line 2391	113, 85
A customer may get checks from more than one station	Check printing	Page 138	Lines 49234, 61423	74, 104

Quality Requirements Guidelines

- Keep sentences and paragraphs short
- Use active voice
- Use proper grammar, spelling, and punctuation
- Use terms consistently and define them in a glossary or data dictionary
- Read requirements from developer's perspective
 - to check if well-defined
- Use the right 'level of granularity'
 - Avoid long narrative paragraphs containing multiple requirements
 - Write individually testable requirements
- Use consistent level of detail throughout document
- Avoid stating requirements redundantly
 - Makes maintenance of document easier
 - Less vulnerable to inconsistencies

Test-Driven Requirements

- This is cited as one of the best techniques for Agile, but many teams are not using it
- This is done before a single line of code is written – it helps to reduce requirements rework that impacts the design and code
- The technique works as follows:
 1. **Review the requirements for that iteration** – look for initial areas of requirements specification problems
 2. **Get feedback on any problems discovered** – update requirements as necessary
 3. **Develop test cases from these requirements** -- The test case design will uncover other problems
 4. **Get feedback on any problems discovered** – update requirements, as necessary
 5. Then write the code . . . and use the test. If error, fix the code.

Requirement Management Tools

- Requirement management is the process of managing the evolution of requirements, from inception, through changes, to system/software retirement
 - New requirements
 - Changes to existing requirements
 - Deleting requirements
- Changing requirements can affect many software process work products, incur change costs, and cause impacts to schedules
- Tools exist to help manage the details of requirements management
 - *Office tools (spreadsheets)* – used to document and track requirements. Usually a manual process. OK for small projects, usually less than 1000 requirements or so
 - *Industry sized tools* – integrates a number of features (RM, Tracing, Risk management, test management, interfaces to other specific tools, etc.)
 - DOORS
 - Jama Software
 - Numerous agile type tools
 - Industry tools frequently interface or provide interoperability with other compatible industry tools or software development tools (through IDE, typically)

Summary: Requirements -- The Basics

- Requirements define the problem to be solved and establish the terms by which mission success will be measured
- Requirements problems are the single biggest problem on development projects
 - Care in creating good requirements always pays off
- The later a requirements problem is discovered, the more costly it is to recover from
- Requirements are distributed within the system architecture via flow-down, allocation, and derivation
- Requirements traceability is a technique of tracking the source and connections between requirements
 - It is used to assess the consequences of potential requirements changes
- When a system is decomposed into smaller segments, interfaces are created that must be defined and managed

Requirements Specification in Class

- Start with the problem description
 1. Identify unique nouns, verbs, adjectives
 - a. Use these to identify classes, methods, and attributes, respectively
 2. Number each requirement R_1, \dots, R_n
 - a. Related requirements can be grouped $R_{1.1}, R_{1.2}$, etc.
 - b. Capture these in a spreadsheet - as in the attached (last page of this presentation)
 - c. When captured, highlight the related text - make sure all text is highlighted that should be
 3. Look for missing converse requirements
 - a. i.e., for a **login requirement** there should be a **logout requirement** somewhere
 - b. For an '*Add books*' requirement there should be a '*Delete books*' requirement
 4. Look for incomplete requirements
 - a. Ask how many? how long? How often? etc.

Requirements Specification in Class (cont'd)

5. Don't capture tabular requirements as text
 - a. Use "*POS shall be alerted as specified in table 2 of the project description.*"
6. Make sure that all requirements have the following:
 - a. All nouns (classes) have been captured
 - b. All verbs (methods/actions) have been captured
 - c. All attributes (make, model, numbers, etc.) have been captured
 - d. Make sure that any un-highlighted text (see next sheet) should not be a requirement
7. Capture functional, non-functional, and interface requirements
8. Make assumptions where not clear - *but do not change the functionality of the problem by stating an assumption*
 - a. Assumptions should support functionality, not change it
 - b. Assumptions that do change the functionality are not assumptions, but likely errors

Example Project Requirement Development

- This is a process used to identify requirements (functional and interface) from a previous CSE Class project

- Project description



ALSS Description

- Project markup - showing identified requirements to make it easy to trace



ALSS
Requirements Identification

- Requirements trace in Excel

