COMPUTER SCIENCE

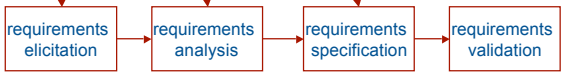
12 - Requirements

Rick Adrion

COMPUTER SCIENCE

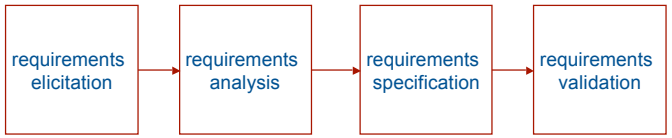
SW Requirements *Process*

- Requirements identification
- Identification of software development constraints
- Requirements analysis
- Requirements representation
- Requirements communication
- Preparation for validation of software requirements
- Managing the requirements definition process definition process.



COMPUTER SCIENCE

SW Requirements *Process*



COMPUTER SCIENCE

Software Requirement *Products*

- Requirements definition
  - Functional
  - Non-functional
  - Inverse
  - Design & implementation constraints
- Requirement documents
  - Standards
- "C-requirements"
- "D-requirements"

Customer/Developer

- Objectives
- Ranking of attributes
- Key contents
  - "C-requirements"
    - Functionality
    - Information definitions
    - Critical non-functional requirements
    - Critical design constraints
    - Acceptance criteria
  - "D-requirements"
    - Functionality
    - Information definitions
    - **Interfaces to external systems**
    - Critical non-functional requirements
    - Critical design constraints
    - Acceptance criteria **and tests**

Outcomes of a Good Process

- The buyers or users
  - often begin with only a vague idea of what they really need and with little idea of what software technology might offer.
  - a good process helps them explore and fully understand their requirements
    - separation of what they want and what they need
    - constraints that might be imposed on the system by technology, organizational practices or government regulations.
    - understand alternatives, both technological and procedural, that might be considered in the proposed system
    - understand the tradeoffs
  - a good understanding of the implications of their decisions ⇒
    - fewer surprises
    - users committed to the success of the project.

Outcomes of a Good Process

- software engineers and developers
  - solving the right problem for the users.
  - have clear, high-level specification of the system to be built.
  - solving a problem that is feasible from all perspectives, not only technical but human
  - customers will be able to use the system, like it, make effective use of it, and that the system will not have undesirable side effects
  - have the trust and confidence of the customers
  - gained knowledge of the domain of the system
  - they have a variety of peripheral or ancillary information about the system useful for making low-level tradeoffs and design decisions.
  - prevented the system from being overly specified
  - have freedom to make implementation decisions.

Outcomes of a Poor Process

- buyers and users can be dissatisfied
  - developers did not really listen to them, or if the developers dominated the process and tended to force their own views and interpretations on the buyers and users.
- a chaotic development process -- developers are missing important information
  - requiring additional meetings with the buyers and users
  - may make the wrong decisions or tradeoffs
  - requirements may change more often,
  - greater need for configuration management, or in delays or wasted effort in design and implementation
  - cost and schedule overruns, and sometimes failed or canceled projects.
- developers are solving the wrong problem
  - guarantees the failure of the whole project
- outcome
  - loss of money for the company developing or buying the software,
  - loss of reputation or credibility for the developers
  - a decline in the developers' morale.

Underlying Difficulties

- Articulation Problems
- Communication Barriers
- Knowledge and Cognitive Limitations
- Human Behavior Issues
- Technical Issues

Articulation Problems

- aware of needs, but unable to articulate them appropriately
- aware of a need but be afraid to articulate it
- not be aware of their needs
- users and developers different meanings for common terms
- users cannot don't understand the consequences or alternatives.
- no single person has the complete picture, no matter how articulate a user may be
- developers may not really be listening to the users
- developers may fail to understand, appreciate, or relate to the users
- developers overrule or dominate the users

Communication Barriers

- users and developers come from different worlds and have different professional vocabularies and views
  - users - high level attributes like usability and reliability
  - developers- lower-level attributes like resource utilization, algorithms, and hardware/ software tradeoffs.
- natural languages are inherently ambiguous
- social interactions
  - different personality types and different value systems among people.
  - can lead to unexpected difficulties in communication
  - SIS example
    - project leader was a high-level person in the company, and he would only talk to comparably high-level people in the university - deans and vice presidents
    - developers on the project would only talk to the IT & administrative staff in the university who (they thought) would actually use system
    - no one talked to faculty, students, and department staff

Knowledge and Cognitive Limits

- requirements elicitor must have adequate domain knowledge
- no person has perfect memory
- informal or intuitive statistics are frequently interpreted differently
- scale and complexity
- preconceived approach to the solution of a problem
- "tunnel vision"
- impatience

Human Behavior Issues

- conflicts and ambiguities in the roles
- fear that installation of the software will necessitate change

Technical Issues

- complexity and social impact
- changing requirements
- changing software and hardware technologies
- many sources of requirements
- nature or novelty of the system

Requirements Engineering

- requirements elicitation
  - the process through which the customers, buyers, or users of a software system discover, reveal, articulate, and understand their requirements.
- requirements analysis
  - the process of reasoning about the requirements that have been elicited; it involves activities such as examining requirements for conflicts or inconsistencies, combining related requirements, and identifying missing requirements.
- requirements specification
  - the process of recording the requirements in one or more forms, including natural language and formal, symbolic, or graphical representations; also, the product that is the document produced by that process.
- requirements validation
  - the process of confirming with the customer or user of the software that the specified requirements are valid, correct, and complete.

Requirements Elicitation

- often called
  - identifying, gathering, determining, formulating, extracting, or exposing
- these terms have different connotations
  - gathering suggests that the requirements are already present somewhere and we need only bring them together
  - formulating suggests that we get to make them up
  - extracting and exposing suggest that the requirements are being hidden by the users
- some truth to all of these connotations

A General Elicitation Procedure

- identify relevant sources of requirements (the users).
- ask them appropriate questions to gain an understanding of their needs.
- analyze the gathered information, looking for implications, inconsistencies, or unresolved issues.
- confirm your understanding of the requirements with the users.
- synthesize appropriate statements of the requirements.
- how?
  - detailed processes
  - specific questions or categories of questions to as
  - structured meeting formats
  - specific individual or group behaviors, or
  - templates for organizing and recording information.

Participants

- lead
- support
- users
- no one person knows everything about what a software system should do
- there are always **many** participants in a successful requirements elicitation effort

Participants

- lead = software engineer (software requirements engineer)
  - responsible for producing the requirements specification
- support = other software engineers, documentation specialists, or clerical staff.
- users = depends on application
  - IS: sales representatives, order processing personnel, shipping department personnel, and accounting personnel. Department managers and company executives
  - Embedded System: design engineers (HW & SW), regulators, system users, managers
  - Productivity tools: users of existing packages, market researchers
  - SIS: students, faculty, advisors, department staff, college staff, registrars, bursars, financial aid, accountants, financial officers, admissions officers, administrators, laboratory technical staff, IT staff, human resources staff, …

General approach

- Asking
  - Identify the appropriate person, such as the buyer or user of the software, and ask what the requirements are.
- Observing and inferring.
  - Observe the behavior of users of an existing system whether manual or automated), and then infer their needs from that behavior.
- Discussing and formulating
  - Discuss with users their needs and jointly formulate a common understanding of the requirements.
- Negotiating with respect to a standard set
  - Beginning with an existing or standard set of requirements or features, negotiate with users which of those features will be included, excluded, or modified.

General approach

- Studying and identifying problems.
  - Perform investigations of problems to identify requirements for improving a system.
- Discovering through creative processes
  - For very complex problems with no obvious solutions, employ creative processes involving developers and users.
- Postulating
  - When there is no access to the user or customer, or for the creation of an unprecedented product, use creative processes or intuition to identify features or capabilities that the user might want.

Traditional methods

- Interviewing customers and domain experts
- Questionnaires
- Observation
- Study of documents and software systems

Interviews

- Interviewing customers and domain experts
- Questions to be avoided
  - Opinionated questions
  - Biased questions
  - Imposing questions

Interviews

- Tutorial interview
  - Expert(s) offers potential solutions and alternatives
- Focused interview
  - Analyst prepares topics but not questions
- Structured interview
  - Analyst prepares & follows a flexible topic structure
    - Open-ended questions
    - Close-ended questions
  - Card sorting, repertory grids
- Teachback interview
  - Users describe problem solving activity to analyst

questioning techniques

- scenario
  - system-specific questions
  - reflects less mature evaluation
- questionnaire
  - more general items
  - reflects more mature evaluation practices
- checklist
  - domain-specific
  - reflects more mature evaluation practices

Scenario

- a specified sequence of steps involving the use or modification of the system
- provides a means to characterize how well a particular architecture responds to the
- demands placed on it by those scenarios test what we normally call modifiability

Scenario usage -- current practice

- Form
  - Narrative text
  - Structured text
  - Diagrammatic notation
  - Images
  - Animations and simulations
- Content
  - System context
  - System interaction
  - System internals

Purpose of Scenarios

- Concretize abstract models
- Scenarios instead of abstract models
- Scenario use with prototypes
- Complexity reduction
- Agreement and consistency
- Scenario usage with glossaries
- Reflection on static models

When to use scenarios

- When abstract modeling fails
  - Cost
  - Inherent complexity
  - Team issues
- In conjunction with prototypes
  - Can yield symbiotic results
  - Steps
    - Develop scenarios
    - Develop prototypes
    - Validate prototypes
    - Refine scenarios

When to use scenarios

- For complexity reduction
  - Use-case approach
  - Scenarios become a structuring device
  - For exception handling & identification
- For achieving partial agreement
  - Stakeholders have different goals & interests
  - Use scenarios to drive the agreement process
- In conjunction with glossaries
  - Establish a common understanding of terms

Questionnaire

- a list of general and relatively open questions that apply to all systems
- how the requirements were generated and documented
- details of the requirements description
  - user interface aspects separated from functional aspects?

Checklist

- a more detailed set of questions that is developed after much experience evaluating a common (usually domain-specific) set of systems.
- help keep a balanced focus on all areas of the system
- more focused on particular qualities of the system than questionnaires
  - e.g., performance questions in a real-time information system
    - is the system writing the same data multiple times to disk?
    - has consideration been given to handling peak as well as average loads?

Questionnaires & Observation

- Questionnaires
  - In addition to interviews
  - Close-ended questions
    - Multiple-choice questions
    - Rating questions
    - Ranking questions
- Observation
  - Passive
  - Active
  - Carried for a prolonged period of time
  - People tend to behave differently

Other

- Study of documents and software systems
  - Use case requirements
    - Organizational documents
    - System forms and reports
  - Domain knowledge requirements
    - Domain journals and reference books
    - ERPS-s (e.g., Peoplesoft)

Modern methods

- Prototyping
- Joint Application Development (JAD)
- Rapid Application Development (RAD)

simulations, prototypes, etc

- may help to create and to clarify the requirements
- performance models are an example of a simulation
- simulation or prototype may answer an issue raised by a questioning technique
  - e.g., what evidence do you have to support this assertion?

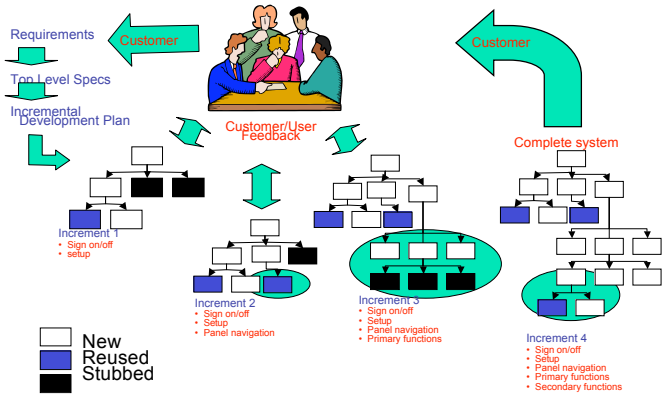
Prototyping

- strategies
  - throw-away prototype
  - evolutionary prototype
- advantages
  - users may be better able to understand and express their needs by comparing to an existing or reference system
- process
  - iterative process of building a prototype and evaluating it with the users.
  - each iteration allows the users to understand their requirements better, including understanding the implications of the requirements articulated in previous iterations.
  - eventually, a final set of requirements can be formulated and the prototypes discarded.

Prototyping

- distinguish the terms prototype and mock-up,
  - A prototype demonstrates behavior of a part of the desired system,
  - A mock-up demonstrates the appearance of the desired system
    - mock-ups of user interfaces are especially common.
- beneficial only if the prototype can be built substantially faster than the actual system
- prototyping should not be viewed as a euphemism for trial-and-error programming or "hacking."
- prototyping is properly used to elicit and understand requirements, followed by a structured and managed process to build the actual system
- useful in overcoming articulation problems and communication barriers.

Cleanroom method



Joint Application Design

- a technique for promoting cooperation, understanding, and teamwork among buyers, users, and developers
- facilitates creating a shared vision of what the system should be
- four main tenets of JAD
  - group dynamics (using facilitated group sessions to enhance the capabilities of individuals)
  - the use of visual aids to enhance communication and understanding
  - maintaining an organized, rational process
  - "what you see is what you get" documentation philosophy (using standard document forms that are filled in and endorsed by all participants in a session).

Joint Application Design

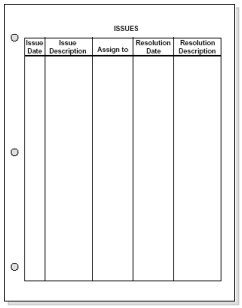
- two major steps:
  - JAD/Plan
    - addresses requirements elicitation and specification
  - JAD/Design
    - addresses software design.
- each step has three phases:
  - customization
    - consists of preparation tasks for the session
    - organizing the team, tailoring the process for the particular system to be built, and preparing materials
  - session
    - one or more structured and facilitated meetings involving the developers and users
    - requirements (or the design) are developed and documented
  - wrap-up
    - converting the information from the session phase into its final form, such as the requirements specification document.

Participants in JAD

- all participants need training in JAD techniques,
- session leader
  - responsible for the overall success of a JAD effort
  - leader and facilitator at meetings
  - good meeting management skills and experience in the application area
- analyst
  - responsible for the production of the output documents of the JAD sessions
  - an experienced developer who can understand the technical issues
- executive sponsor
  - manager or executive who has ultimate responsibility for the product being built
- user representatives
  - requirements elicitation- managers or key people within the organization
  - design - variety of other users
- Information systems representatives
  - help the users understand what is and is not reasonable or feasible
- specialists
  - from the user community
  - from the developer community

JAD details

- JAD/plan customization phase
  - conduct orientation
  - organize the team
  - tailor the process
  - prepare materials
- the JAD/plan session phase
  - conduct orientations
  - define high-level requirements
  - bound the scope of the system
  - identify and estimate JAD/designs
  - identify participants for JAD/design step
  - schedule JAD/design meetings
  - conclude the session phase
- JAD/plan wrap-up phase
  - complete the JAD/plan document
  - review the JAD/plan document.
  - obtain executive sponsor approval

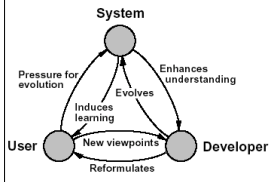


RAD

- Evolutionary prototyping
- CASE tools
- Specialists with Advanced Tools (SWAT)
- Interactive JAD
- Timeboxing

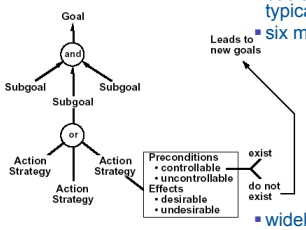
Adaptive Loops Framework

- similar in spirit to JAD - uses an adaptive process of learning cycles or loops.
  - developers are assisted by the users to gain new viewpoints about their requirements, and through reformulating the requirements, the user learns more about them
  - system receives pressure for evolution as the users learn more about how it can be used, and the system induces that learning on the users
  - system evolves by actions of the developers, who in turn gain enhanced understanding of the system through that evolution.
- especially useful when there are requirements articulation problems and to overcome the technical issues of complex systems.

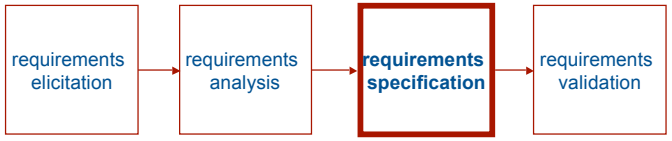


Critical Success Factors Analysis

- basic premise = the effectiveness of a system typically depends on a small set of critical factors
- six major steps:
  - understand the operation of the system.
  - identify the factors that are critical for the effectiveness of the system.
  - identify the strengths and weaknesses of the system with respect to each of
  - these factors.
  - identify areas of problems and opportunities.
  - gather relevant details for enhancing system performance relative to these critical success factors.
  - formulate requirements using these details.
- widely used in building information and decision support systems
- useful in addressing some of the difficult technical and cognitive issues of requirements elicitation.



SW Requirements *Process*



Requirements document

Requirements Document
Table of Contents

1. **Project Preliminaries**
   - 1.1 Purpose and Scope of the Product
   - 1.2 Business Context
   - 1.3 Stakeholders
   - 1.4 Ideas for Solutions
   - 1.5 Document Overview
2. **System Services**
   - 2.1 The Scope of the System
   - 2.2 Function Requirements
   - 2.3 Data Requirements
3. **System Constraints**
   - 3.1 Interface Requirements
   - 3.2 Performance Requirements
   - 3.3 Security Requirements
   - 3.4 Operational Requirements
   - 3.5 Political and Legal Requirements
   - 3.6 Other Constraints
4. **Project Matters**
   - 4.1 Open Issues
   - 4.2 Preliminary Schedule
   - 4.3 Preliminary Budget

Appendices
- Glossary
- Business Documents and Forms
- References

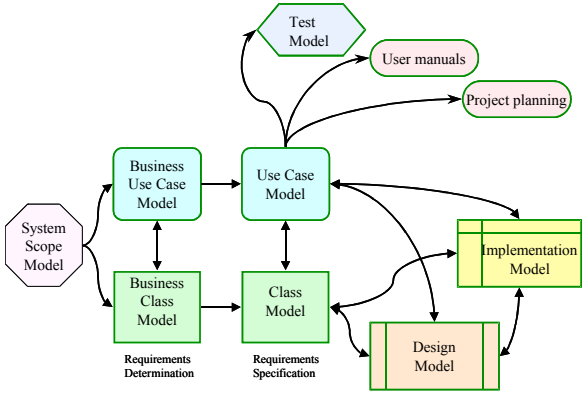
Project preliminaries chapter

- Targets managers and decision makers
- Begins with purpose and scope of the project
- Makes a business case for the system
- Identifies stakeholders
- Offers initial ideas for the solution
- Includes an overview of the rest of the document

System services chapter

- Dedicated to the definition of **system services** -what the system must accomplish
- Likely to account for more than half of the entire document
- Contains high-level requirements business models
  - **Context diagram** (the system scope)
  - **Business use case diagram** (function requirements)
  - **Business class diagram** (data requirements)

Requirements business model



System constraints chapter

- Dedicated to the definition of system constraints - how the system is constrained when accomplishing services with regard to
  - Interface requirements
  - Performance requirements
  - Security requirements
  - Operational requirements
  - Political and legal requirements
  - Other constraints
    - Usability
    - Maintainability

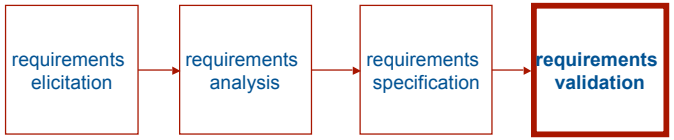
Project matters chapter

- Open issues
  - Future requirements
  - Current requirements to be implemented in the future – enhancements
  - Potential problems once when the system deployed
- Preliminary schedule
  - Human and other resources
  - Planning charts (PERT, Gantt)
- Preliminary budget
  - Project cost – range rather than figure

Appendices chapter

- Glossary
  - Terms
  - Acronyms
  - Abbreviations
- Documents and forms
  - Examples of completed (filled in) forms
- References
  - To books and other published sources
  - Meetings' minutes, memoranda, internal documents

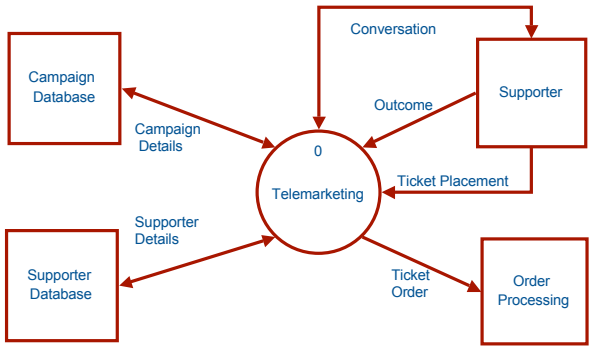
SW Requirements *Process*



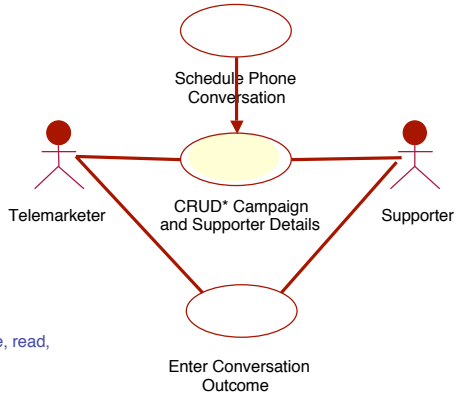
Requirements Negotiation & Validation

- Negotiation
  - Based on draft of document
- Validation
  - Based on (almost) complete document
- Issues
  - Scope
  - Dependencies
  - Risks
  - Priorities

System scope model



Business use case model



CRUD* - create, read, update, delete

Requirements dependency matrix

Requirement	R1	R2	R3	R4
R1	X	X	X	X
R2	Conflict	X	X	X
R3			X	X
R4		Overlap	Overlap	X

Requirements risks

- Technical
- Performance
- Database integrity
- Development process
- Political
- Legal
- Volatility