COMPUTER SCIENCE

11 - UML & Requirements

Rick Adrion

COMPUTER SCIENCE

UML 2 Activity Diagrams

- object-oriented equivalent of flow charts and data flow diagrams (DFDs) from structured development
- typically used for
  - business process modeling
  - modeling the logic captured by a single use case or usage scenario
  - modeling the detailed logic of a business rule
- could potentially model
  - the internal logic of a complex operation
  - far better to simply rewrite the operation so that it is simple enough that you don't require an activity diagram

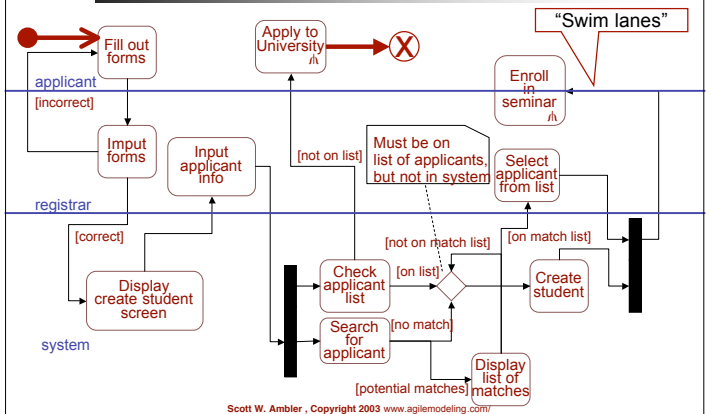
COMPUTER SCIENCE

Activity Diagram Notation

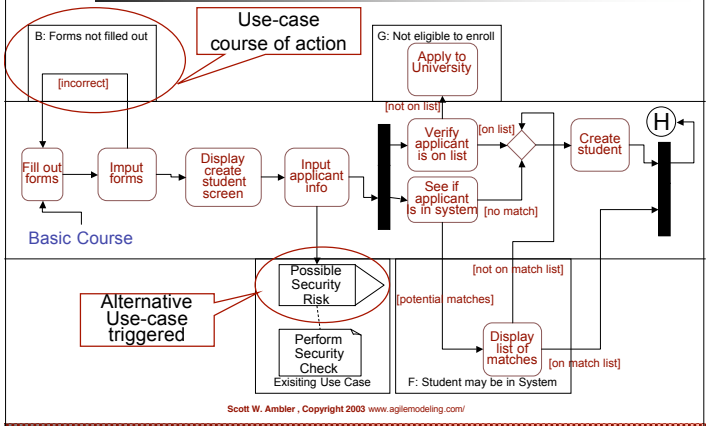
- Initial node
  - filled circle = starting point of the diagram, not required
- Activity final node
  - filled circle with a border is the ending point, can have zero or more activity final nodes.
- Activity
  - rounded rectangles represent activities, may be physical or electronic
- Flow/edge
  - arrows on the diagram
- Fork
  - A black bar with one flow going into it and several leaving it, parallel activity.
- Join
  - A black bar with several flows entering it and one leaving it, denotes the end of parallel processing.
- Condition
  - a guard which must evaluate to true in order to traverse the node
- Decision
  - A diamond with one flow entering and several leaving
- Merge
  - A diamond with several flows entering and one leaving, all incoming flows must reach this point until processing continues, unless otherwise noted
- Partition
  - also called swimlanes, indicating who/what is performing the activities
- Sub-activity indicator
  - rake in the bottom corner of an activity indicates that the activity is described by a more finely detailed activity diagram.
- Flow final
  - circle with the X through it, process stops at this point.
- Use case
  - non-official? indication that an included use case is being invoked.

COMPUTER SCIENCE

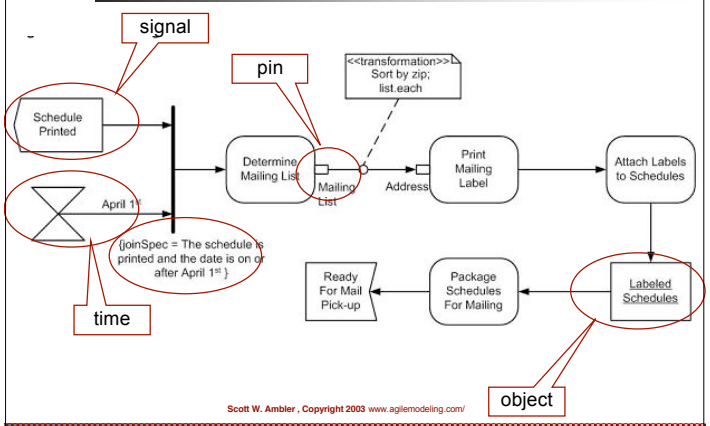
University enrollment example



University enrollment example



University enrollment example



When to Use Activity Diagrams

- Use activity diagrams when the behavior you are modeling ...
  - does not depend much on external events.
  - mostly has steps that run to completion, rather than being interrupted by events.
  - requires object/data flow between steps.
  - is being constructed at a stage when you are more concerned with which activities happen, rather than which objects are responsible for them (except partitions possibly).

Activity Diagram Modeling Tips

- Control flow and object flow are not separate. Both are modeled with state transitions.
- Dashed object flow lines are also control flow.
- You can mix state machine and control/object flow constructs on the same diagram (though you probably do not want to).

Wrap Up: Activity Diagrams

- Use Activity Diagrams for applications that are primarily control and data-driven, like business modeling …

… rather than event-driven applications like embedded systems.

- Activity diagrams are a kind of state machine until UML 2.0 …

… so control and object/data flow do not have separate semantics.

Statechart modeling

- Captures dynamic changes of class states – the life history of the class
- These dynamic changes describe typically the behavior of an object across several use cases
- State of an object – designated by the current values of the object's attributes
- Statechart Diagram – a bipartite graph of
  - states (rounded rectangles) and
  - transitions (arrows) caused by events
- The concepts of states and events are the same concepts that we know from Activity Diagrams – the difference is that "the states of the activity graph represent the states of executing the computation, not the states of an ordinary object"

Types of Events

- UML defines 4 kinds of events
  - Signal Event
    - Asynchronous signal received
    - e.g. evFlameOn
  - Call Event
    - operation call received
    - e.g. op(a,b,c)
  - Change Event
    - change in value occurred
  - Time Event
    - Relative time elapse
    - Absolute time arrived
    - e.g. tm(PulseWidthTime)

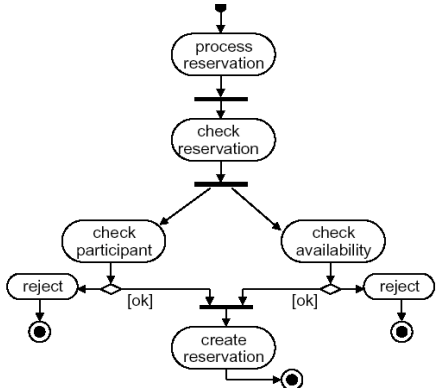
Types of Events

- Events are occurrences of interest that have both
  - Location
  - Absolute time of occurrence
- Signal events associate with Signals
  - A Signal is a specification of an asynchronous communication between structural elements (e.g. objects)
  - One type of Signal is Exception

States and transitions

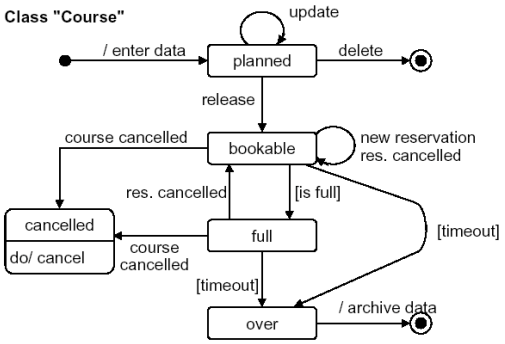
- Objects change values of their attributes but not all such changes cause state transitions
- We construct state models for classes that have interesting state changes, not any state changes
- Statechart Diagram is a model of business rules
  - Business rules are invariable over some periods of time
  - They are relatively independent of particular use cases

Statecharts & Activity Diagrams



Heinrich Hussmann

States and transitions



Heinrich Hussmann

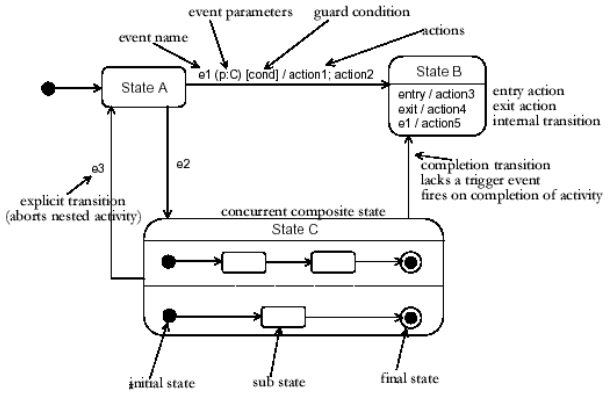
Statechart Diagram

- Normally attached to a class, but can be attached to other modeling concepts, e.g. a use case
- When attached to a class, the diagram determines how objects of that class react to events
  - Determines – for each object state – what **action** the object will perform when it receives an event
  - The same object may perform a different action for the same event depending on the object's state
  - The action's execution will typically cause a state change

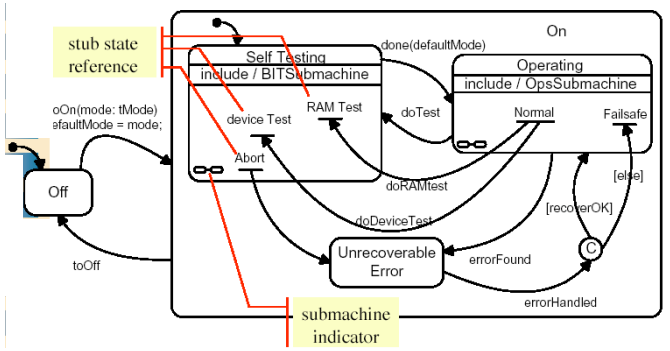
Statecharts

- Capture state-dependent requirements
- Statechart created for each state-dependent class
- UML provides hierarchical state transition diagrams
  - Based on Harel statecharts
- Information Captured
  - States
    - Capture all possible states of the class
    - Events and conditions
  - Describe transitions between states
    - Actions
  - Indicate processing that occurs on entry or exit to/from a state

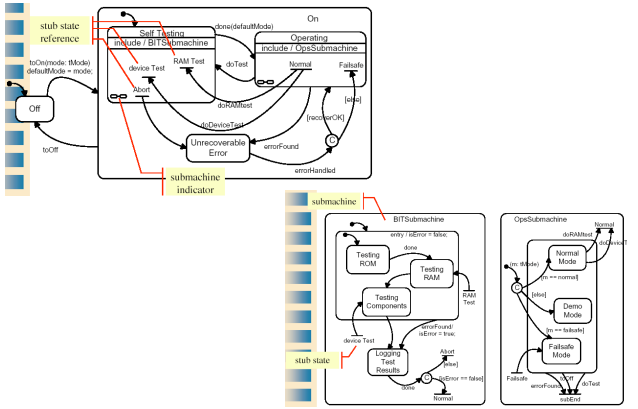
Statechart notation



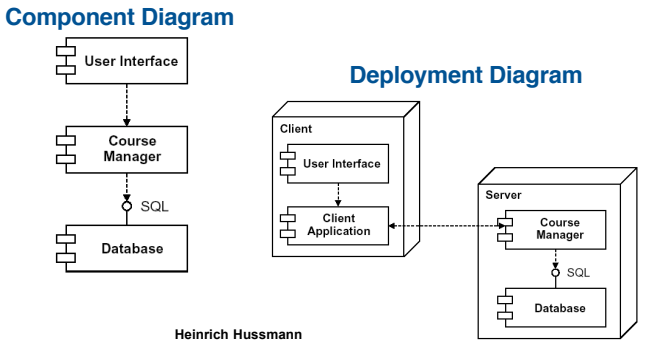
Example



Nested submachines

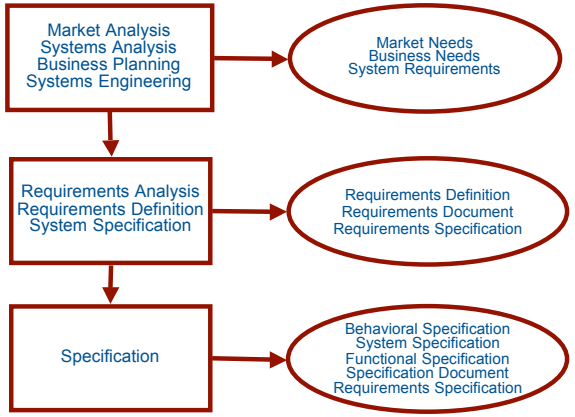


Implementation diagrams



Heinrich Hussmann

Requirements Process & Products



System Planning

- Business strategy
  - Small organizations
  - Large organizations
- Approaches
  - Strength, Weaknesses, Opportunities, Threats (SWOT)
  - Value Chain Model (VCM)
  - Business Process Re-Engineering (BPR)
  - Information Systems Architecture (ISA)
- Effectiveness vs. efficiency

Approaches

- SWOT
  - Top-down classification, ranking and selection of projects based on: mission statement, internal strengths and weaknesses, external opportunities and threats, objectives, goals, strategies, and policies
- VCM
  - Look at “value chain” – from raw materials to final products sold and shipped to customers and identify critical areas where IT can transform organization’s value chain
- BPR
  - Aimed at radical redesign of business processes, based on business process”ownership,” and horizontally cross-cutting processes with end at points of contact with customers. IT support enables BPR
- ISA
  - A neutral architectural framework with stakeholders (planner, owner, designer, builder, subcontractor) and activities(what, how, where, who, when, why)

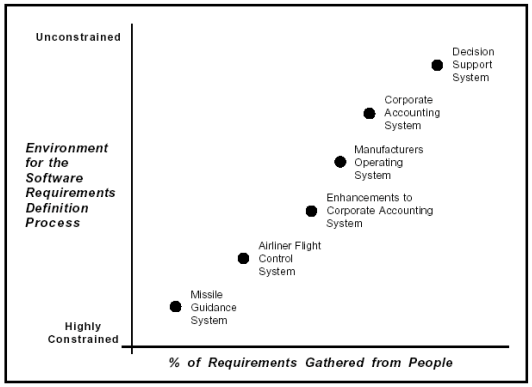
Systems and management levels

Level of decision making	Focus of decision making	Typical IS applications	Typical IT solutions
Strategic	Strategies in support of organizational long-term objectives	Market and sales analysis, Product planning, Performance evaluation	Data mining, Knowledge management
Tactical	Policies in support of short-term goals and resource allocation	Budget analysis, Salary forecasting, Inventory scheduling, Customer service	Data warehouse, Analytical processing, Spreadsheets
Operational	Day-to-day staff activities and production support	Payroll, Invoicing, Purchasing, Accounting	Database, Transactional processing, Application generators

SW Requirements Defn 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.

Source of requirements



Requirements identification

- elicited from people or derived from system requirements
- Software needs -- Context analysis
  - documents why software is to be created and why certain technical, operational, and economic feasibilities establish boundary conditions
- Elicitation from people
- Deriving from system planning requirements
- Task analysis to develop user interface

More steps

- Identification of software development constraints
  - Costs, hardware/software, reliability, portability
- Requirements analysis
  - Assessment of potential problems
  - Classification of requirements mandatory, desirable, and non-essential
  - Evaluation of feasibility and risks

Requirements representation

- Use of models
  - A good model:
    - Reduces the amount of complexity that must be comprehended at one time.
    - Is inexpensive to build and modify compared to the real thing.
    - Facilitates the description of complex aspects of the real thing.
- Roles for prototyping
  - prototype is not a substitute for a thorough written specification
  - a system can be captured in a prototype

More steps

- Requirements communication
  - Present to stakeholders for review
- Preparation for validation of software requirements
  - Establish criteria
  - Identify techniques to be used
- Managing the requirements definition process definition process.
  - a major project management challenge.
  - an application that must support five different classes of users with significantly different expectations could easily involve a requirements definition process that is five times more difficult than the corresponding process for a homogeneous group

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 = 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, …
- no one person knows everything about what a software system should do
- always many participants in a successful requirements elicitation effort

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

- Tutorial interview
  - Expert 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

Interviews

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

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

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