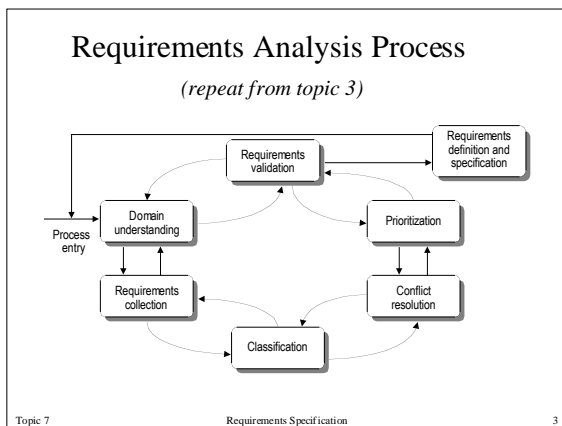
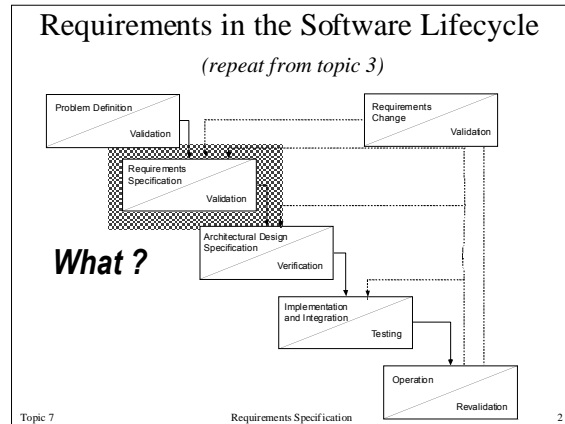
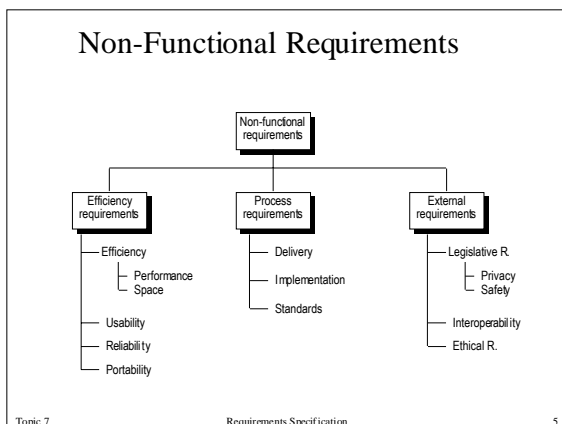


ICS 121 Topic 7:  
**Requirements Specification**

System Contexts  
 System Models  
 Use Cases  
 System and Acceptance Test Plans



- ### Products
- (repeat from topic 3)*
- Refinement of customer needs
  - Documentation of all requirements and constraints
    - functional
    - nonfunctional
  - Lifecycle considerations
  - Acceptance Test Plan
- Should not begin a project without a  
 GOOD CONTRACT  
 that completely describes customer expectations**
- Topic 7 Requirements Specification 4



- ### Method-based requirements analysis
- (repeat from topic 3)*
- Most widely used approach to requirements analysis
    - Depends on the application of some structured methods to understand the system
    - Results are expressed in a set of system models
    - Methods have different emphases: some are focused exclusively on requirements elicitation/analysis, others are very close to design
  - Structured methods usually include:
    - Process model (dataflow analysis, control scenario identification)
    - System modeling notations (diagrammatic, form-based, linguistic)
    - Rules applied to the system model
    - Design guidelines
    - Report templates
- Topic 7 Requirements Specification 6

### System Contexts

- Early in the analysis process, the boundaries of the system have to be defined
- Example:

Topic 7 Requirements Specification 7

### System Models

- A system model is an abstract description of the system to be developed
- Particular requirements analysis methods choose a set of system models as part of the method
  - different system models contribute in different ways to the understanding of the system (there is no ideal system model, nor is there an ideal method to develop such models !)
- Different system models are based on different approaches to abstraction (functional, data)
- Typical kinds of system models:
  - data-processing model
  - composition model
  - classification model
  - stimulus-response model

Topic 7 Requirements Specification 8

### System Models, *continued*

- Widely used system models:
  - Data-flow models:
    - Show how data is processed by a system
    - Data-flow models are basic system models of Structured Systems Analysis [DeMarco,1978]
  - Semantic data models:
    - Identify the data entities, their attributes, and inter-relationships
    - Examples: Entity-Relationship Modeling [Chen,1976] SDM [Hammer/McLeod,1981] RM/T (extension of the relational model) [Codd,1979]
  - Object models:
    - Represent data and its processing (together with structure of the data)
    - Various notations: [Booch,1994], [Coad/Yourdon,1990] [Rumbaugh et al.,1991], [Coleman et al.,1994]

Topic 7 Requirements Specification 9

### Requirements Specification

- Structured natural language
  - Extended, more detailed form of textual requirements definition
  - Advantage:
    - Uses expressiveness and understandability of natural language
  - Problems:
    - Inherent ambiguity of natural language
    - Requirements are not partitioned effectively by the language itself (it's difficult to find related requirements)
  - Example: Usage of standard forms

|                    |   |
|--------------------|---|
| <b>Function</b>    | Add node  |
| <b>Description</b> | Adds a node to an existing design. The user selects the type of node, and its position. [...] |
| <b>Inputs</b>      | Node type, Node position, Design identifier [...]   |

Topic 7 Requirements Specification 10

### Requirements Specification, *continued*

- Requirements/Program description languages
  - A PDL is derived from a programming language (e.g., Ada), but usually adds more abstract constructs to increase its expressive power
  - Various special-purpose requirements specification languages have been designed, e.g. PSL/PSA [Teichrow/Hershey,1977], RSL [Alford,1977]

**Example:** A PDL description of ATM operation

```

procedure ATM is
begin
loop
Get_card (Acc_no, PIN, Valid_card);
if Valid_card then
Validate_PIN (PIN, Valid_PIN);
if Valid_PIN then
Get_account (Acc_no, Balance); Get_service (Service);
while a service is selected loop
Deliver_selected_service (Get_service (Service));
end loop;
end loop;
Return_card;
endif;
endif;
end loop;
end ATM;
    
```

Topic 7 Requirements Specification 11

### Requirements Specification, *continued*

- Semi-formal/Graphical notations
  - Graphical notations have a loose semantics associated with the structure
  - Widely used, e.g. SADT [Ross,1977], SSA [DeMarco,1978] [Gane/Sarsen,1979] [Yourdon/Constantine,1979]
- Formal/Mathematical notations
  - Formal specifications base on a formal semantics (mathematical concept)
  - Specifications are unambiguous (reduce the arguments between customer and contractor about system functionality)
  - Difficult to understand for customer
  - Examples: Finite State Machines, Petri Nets [Peterson,1977], Algebraic Specifications, Z [Hayes,1987], VDM [Jones,1980]

Topic 7 Requirements Specification 12

### Structured Systems Analysis

- Semi-formal technique using graphics to specify software
  - Developed in the 1970s: [DeMarco,1978], [Gane/Sarsen,1979], [Yourdon/Constantine,1979]
- Data-flow models (dataflow diagrams) used to show how data flows through a sequence of processing steps
  - data is transformed at each step before moving on to the next
  - simple and intuitive (users participate in validating the analysis)
  - track and document how data associated with each process moves through the system (determine the logical data flow)
- Data-flow models show a functional perspective (each transformation represents a single function)

Topic 7 Requirements Specification 13

### Structured Systems Analysis, *continued*

- Notation:
  - Circles represent processing steps
  - Arrows annotated with the data name represent flows
  - Two lines represent data stores (or data sources)

Topic 7 Requirements Specification 14

### Structured Systems Analysis, *continued*

- DFDs can be expressed at a number of different levels of abstraction

- In practice, information about the system is acquired about different levels at the same time
- Lower-level models may be developed first then abstracted to create a more general model

Topic 7 Requirements Specification 15

### Structured Systems Analysis, *continued*

- SSA approach of Gane and Sarsen [Gane/Sarsen,1979]
  - Step 1: Draw the DFD
  - Step 2: Decide what sections to computerize and how (batch/online)
  - Step 3: Put in the details of data flows
  - Step 4: Define the logic of the processes (e.g. decision trees/tables)
  - Step 5: Define the data stores
  - Step 6: Define the physical resources
  - Step 7: Determine the input/output specifications
  - Step 8: Perform sizing
  - Step 9: Determine the hardware requirements

Topic 7 Requirements Specification 16

### Entity-Relationship Modeling

- Emphasize of SSA is on actions --> ER Modeling is a semi-formal data-oriented technique for specifying a system
  - Definition of the logical form of data processed by the system
  - Originally developed in 1976: [Chen,1976]
    - since then various extensions (e.g., including sub-typing, etc.)
- Data models are often used to supplement the information provided on DFDs
- ER models have been widely used in database design
  - Relational schemas can be easily derived from ER models (usually in third normal form)

Topic 7 Requirements Specification 17

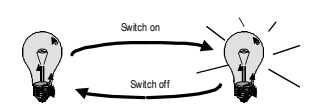
### Entity-Relationship Modeling, *continued*

- Notation:
  - Entities (with entity attributes)
  - Relations (with relation attributes)
  - Cardinality of relationships (1:1, 1:m, n:m)

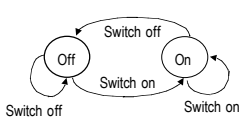
Topic 7 Requirements Specification 18

### Finite State Machines

- State machine modelling assumes that the system is always in one of a number of possible states
- Example:
 



| Action     | Current State |     |
|------------|---------------|-----|
|            | Off           | On  |
| Switch on  | On            | On  |
| Switch off | Off           | Off |



Topic 7
Requirements Specification
19

### Finite State Machines, *continued*

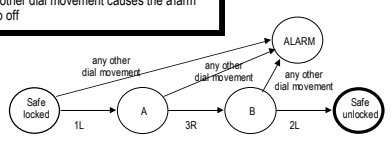
- Basic elements of a state machine (J,K,T,S,F):
  - a finite, nonempty set of states J, including
    - the initial state S
    - the set of final states F
  - a finite, nonempty set of input events K
  - the transition function T: (J\F) x K -> J
    - specifies next state given the current state and an input event, no transition for final states
- For specifying a system, a useful extension of FSMs is to add a set of predicates P:
  - the transition function is then T: (J\F) x K x P -> J
    - specifies next state given the current state and an input event when the predicate is true

Topic 7
Requirements Specification
20

### Finite State Machines, *continued*

- Example: Safe with combination lock [Brady,1977]
 

- Combination lock with 3 positions: 1,2,3
  - The dial can be turned left L or right R
  - Any time there are 6 possible dial movements, 1L, 1R, 2L, 2R, 3L, 3R
  - The combination of the lock is: 1L, 3R, 2L
  - any other dial movement causes the alarm to go off



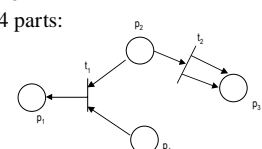
Topic 7
Requirements Specification
21

### Finite State Machines, *continued*

- Use of FSM approach is widespread in computing applications:
  - Every menu-driven user interface is implementation of a FSM
- FSM approach is more precise than other graphic, semi-formal approaches, but almost as easy to understand
- Problem:
  - For large systems the number of (state, event, predicate) triples grows rapidly
  - Timing considerations are not handled in the basic formalism
  - Extension of FSMs = statecharts [Harel et al., 1990] similar extension is used in UML

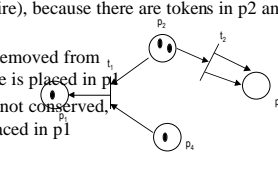
Topic 7
Requirements Specification
22

### Petri Nets

- A major problem with specifying concurrent systems is coping with timing
  - real-time systems, for instance, require careful specification of timing aspects
- Petri nets [Petri,1962] are widely applicable in software engineering
  - especially useful for describing concurrent interrelated activities
- A Petri net consists of 4 parts:
  - A set of places P
  - A set of transitions T
  - An input function I
  - An output function O

Topic 7
Requirements Specification
23

### Petri Nets, *continued*

- A marking of a Petri net
  - is an assignment of tokens to that Petri net represented by a vector, e.g. (1,2,0,1)
  - a transition is enabled if each of its input places has as many tokens in it as there are arcs from the place to that transition, e.g.
    - t1 is enabled (ready to fire), because there are tokens in p2 and p4
    - if t1 fires, one token is removed from p2 and one from p4, one is placed in p1
    - the number of tokens is not conserved, i.e. only one token is placed in p1
    - (1,2,0,1)
    - fire t1

Topic 7
Requirements Specification
24

### Petri Nets, *continued*

- Petri nets are nondeterministic
  - if more than one transition is able to fire, then any one of them may be fired
- Extension: Inhibitor arc
  - A transition is enabled if there is at least one token on each of its input arcs and no tokens on any of its inhibitor input arcs

Topic 7 Requirements Specification 25

### Petri Nets, *continued*

- Example: Elevator button

Topic 7 Requirements Specification 26

### Petri Nets, *continued*

- In classical Petri net theory transitions are instantaneous
  - motion from floor g to floor f takes place instantaneously
- For modeling practical situations timed Petri nets are needed (transitions with a nonzero time) [Coolahan/Roussopoulos,1983]

Topic 7 Requirements Specification 27

### Use Cases

- A “Use Case” is a description of one small task the user would do when using the system.
  - Something that the user wants to accomplish. e.g., “I would like to check the spelling of my paper.”
  - A conscious and specific user goal.
    - “Write a paper” is too high level.
    - “Spell check a document” is about right.
    - “Press a key” is too low level, and too system oriented.
  - Often corresponds to a feature of the system. e.g., spell checker a word processor document.

Topic 7 Requirements Specification 28

### Use Case Actors

- Actors are types of users
  - Different kinds of people who use the system in different ways
  - Often relates to the roles people play in a company or project
  - For example, one actor might be a “visitor” to a web site, another actor is the “webmaster” of the site
    - And you might also define types of visitors such as first-time visitors, members, contributors, etc.

Topic 7 Requirements Specification 29

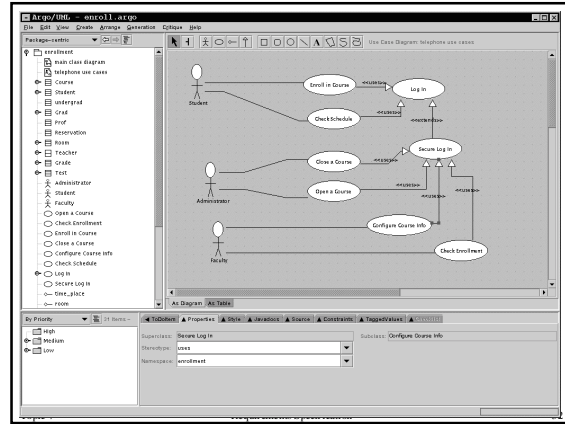
### Use Case Diagrams

- Shows expected actors and use cases
- Show which actors do which use cases
- Show dependency and inheritance among use cases

Topic 7 Requirements Specification 30

### Dependency and Inheritance

- A simple line between an actor and a use case means that that actor is expected to perform that use case
- A line with an arrow head from an actor to an actor defines a special kind of actor: e.g., Student, Grad, Undergrad
- A line with an arrow head from use case to use case is labeled
  - «extends» : The bottom use case is a special way to do the more general task
  - «uses» : The bottom use case is a larger task that includes the top use case as one step

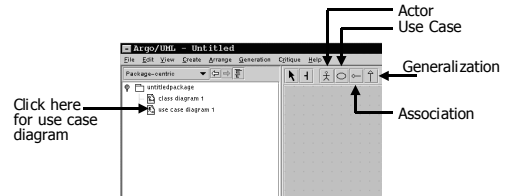


### Launching Argo/UML

Log in to an ICS Sun workstation and type **/home/jrobbins/argo**  
 Or you can download argo to run on your own PC <http://www.argouml.org>  
 Launching takes 30-60 seconds  
 You will also need to be able to run Argo/UML to complete some other homework assignments later

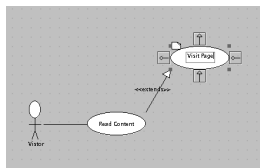
### Using Argo/UML

- When you launch Argo/UML it automatically makes a class diagram and a use case diagram
- Click on “use case diagram 1” in tree pane
- Use the toolbar to place actors and use cases



### Using Argo/UML, continued

- Click and type to name actors or use cases
- You can use selection buttons or toolbar buttons to make associations or generalizations
- All generalizations are «extends» by default, use the properties tab to change it to «uses»



### Assignment 4: Use cases for Virtual Mall Online

- Define the actors and use cases for Virtual Mall Online
  - Hint: expect 3-5 actors and 4-8 use cases
- Use Argo/UML to draw a use case diagram.
- Homework solution should include:
  - Printed use case diagram
  - Short textual descriptions of each actor and use case
  - Describe one use case in detail: a paragraph with detailed steps

### Testing: System Test Plan

- Developed as part of requirements analysis and specification
- Basic goal is to test behavior of each specified software feature
- Non-functional testing of other behavioral features, qualities stated in requirements specification
  - load/stress testing
  - performance testing
  - reliability testing
  - robustness/recovery testing
  - storage testing
  - configuration testing
  - security testing
  - safety testing
  - real-time response testing
  - documentation testing
  - usability testing
  - compatibility testing
  - installability testing

Topic 7

Requirements Specification

37

### Testing: Acceptance Test Plan

- An operational way of determining consistency between the requirements document and the delivered system

If the system passes the tests demanded by this plan, then the user has **less** basis for complaint

- Develop a plan for conducting tests to examine:
  - functional requirements
  - non-functional constraints
  - subsets

Topic 7

Requirements Specification

38

### System Test Plan Process

- For a given requirement:
  - Design test cases to test that requirement
    - design typical test cases for functionality
    - design test cases for non-functional aspects
      - robustness
      - safety
      - security
      - performance
    - design special and boundary value test cases
  - For each test case, provide the "values" of parameters and any environment (e.g., persistent data) required
  - Plan the order of the test cases for this requirements node
    - initialize, set-up, process
- Plan the order of system testing

Topic 7

Requirements Specification

39