Requirements Engineering (the Activity)

- System engineering v. software engineering
  - What role does software play within the full solution?
- Contract model v. participatory design
  - Contract: carefully specify requirements, then contract out the development
  - Participatory: ultimate users, users' agents, and software engineers work together throughout development
Requirements Specification (the Document)

- **Purpose**
  - Serve as the fundamental reference point between builder and buyer/'consumer'
    (contract)
  - Define capabilities to be provided, without saying how they should be provided
  - Define constraints on the software
    » e.g. performance, platforms, language

- **Characteristics**
  - Unambiguous
    » Requires precise, well-defined notations
  - Complete: any system that satisfies it is acceptable
  - Consistent
    » There should be no conflicts or contradictions in the descriptions of the system facilities
  - Verifiable (testable)
  - No implementation bias (external properties only)
    » "One model, many realizations"
Users of a requirements document

- **System customers**
  - Specify the requirements and read them to check that they meet their needs. They specify changes to the requirements.

- **Managers**
  - Use the requirements document to plan a bid for the system and to plan the system development process.

- **System engineers**
  - Use the requirements to understand what system is to be developed.

- **System test engineers**
  - Use the requirements to develop validation tests for the system.

- **System maintenance engineers**
  - Use the requirements to help understand the system and the relationships between its parts.
Lifecycle Considerations

- Serve as basis for future contracts
- Reduce future modification costs
  - Identify items likely to change
  - Identify fundamental assumptions
- Structure document to make future changes easy
  - e.g. have a single location where all concepts are defined
# Requirements Volatility

![Figure 4-1: Matrix of Requirements Terminology](image)

<table>
<thead>
<tr>
<th>Observable to Users</th>
<th>Customer Doesn’t Care</th>
<th>Customer Cares</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement likely to change</td>
<td>Measurable</td>
<td>Requirement</td>
</tr>
<tr>
<td>Implementation detail</td>
<td>Constraint</td>
<td>Goal</td>
</tr>
</tbody>
</table>

**Source:** David Alex Lamb, Software Engineering, Planning for Change
Prentice Hall, 1988
Content of a Requirements Specification

◆ Application context
  – Describe the situations in which the software will be used. How will the situation change as a result of introducing the software system?
  – Identify all things (objects, processes, other software, hardware, people) that the system may, or will, affect.
  – Develop an abstraction for each of those things, characterizing their properties/behavior which are relevant to the software system. ("World model.")
  – How might this context change?
◆ Functional requirements ("features")
  – Identify all concepts (objects) that the system provides to the users.
  – Develop an abstraction for each of those concepts, characterizing their properties and functions which are relevant to the user.
    » What is the system supposed to do?
    » What is supposed to happen when something goes wrong?
Contents of a
Requirements Specification, cont..

- Performance requirements: speed, space
- Environmental requirements: platform, language, ...
- Subsets/supersets
- Expected changes and fundamental assumptions
- Definitions; reference documents
Non-functional requirement types

- Performance requirements
- Usability requirements
- Efficiency requirements
- Reliability requirements
- Portability requirements
- Interoperability requirements
- Ethical requirements
- Delivery requirements
- Implementation requirements
- Standards requirements
- Legislative requirements
- Space requirements
- Privacy requirements
- Safety requirements
World Model (OOA) versus Simple Input/Output Characterizations as Reqt.s Specs

- The application context may change because of extrinsic factors
- The software system modifies the usage context

- I/O is only meaningful in a specific context
- "Input" and "output" may not be simple concepts
  - Cruise control systems: many sensors, complex conditions, and timing constraints only understandable in the application context
Techniques for Requirements Analysis

- Conduct interviews
- Build and evaluate prototypes
- Construct glossaries
- Separate concerns
- Focus on structure
  - Abstraction and hierarchical decomposition
- Use precise notation (be careful with diagrams!)
- Ask yourself:
  - Is it testable? Complete? Consistent?
Canonical Diagram for Requirements Objects

Object Name

Description/Attributes

Operations it can be asked to perform
  • op1
  • op2
  • ...

Requests made of other objects

Note: this will not be the appropriate notation for all application contexts!
**Mailing List Manager**

**Mailing Address**

A place where mail can be delivered. Name, Title, Street, City, State, ZipCode.

Operations:
1. Change any of the specified attributes to have a particular value.
2. Read any or all of the attributes.
3. Create/delete address.

Note: are the values to the “puts” or received from the “gets” strings? Only strings?

**Storage**

An indexed set of places where chunks of ASCII data can be stored. Number of indices, size of data currently stored in each index.

Operations:
1. Fetch data at index.
2. Store data at index.

**Mailing List**

A list of Mailing_Address objects. Name (of list).

Operations:
1. Add Mailing_Address to list.
2. Delete Mailing_Address from list.
3. Sort list.
4. “Print” list.

Note: What about querying the list to see if a particular address --- or part of one --- is already a member?

**Mailing List Set Ops**

Supports manipulation of multiple mailing lists.

Operations:
1. Union of two lists.
2. Intersection of two lists.
3. Subtraction of one list from another.

**User Interface**

What the human user interacts with in order to manipulate or obtain any info.

Attributes: media and modes.

Operations:
1. Login (authenticate user).
2. Parse and execute command.

Note: requests between objects not shown. Neither the application context nor the customer imposes any constraints on how these objects may interact.
# Mailing List Manager, Take 2

Is this better, or worse?

## Mailing List

A list of `Mailing_Address` objects.

Name (of list)

Operations:
1. Add `Mailing_Address` to list
2. Delete `Mailing_Address` from list
3. Sort list
4. “Print” list
5. Combine (union) two lists
6. Intersection of two lists --> list
7. `List2 = List1 - List0`
8. Store list
9. Retrieve list

## Mailing Address

A place where mail can be delivered.

Name, Title, Street, City, State, ZipCode.

Operations:
1. change any of the specified attributes to have a particular value.
2. read any or all of the attributes
3. create/delete address

## Storage

An indexed set of places where chunks of ASCII data can be stored. Number of indices, size of data currently stored in each index.

Operations:
1. Fetch data at index
2. Store data at index

## User Interface

What the human user interacts with in order to manipulate or obtain any info.

Attributes: media and modes

Operations:
1. Login (authenticate user)
2. Parse and execute command
Different Circumstances, Different Techniques

- Finite state machines
  - telephony examples
  - http://www.uclan.ac.uk/facs/destech/compute/staff/casey/integ/mscfs.html
- Numerical systems
  - e.g. matrix inversion package
Acceptance Test Plan

- An operational way of determining consistency between the requirements specification and the delivered system
- If the system passes the tests demanded by this plan, then the buyer has no (legal) basis for complaint
- Develop a plan for conducting test to examine
  - Functional properties
  - Performance properties
  - Adherence to constraints
  - Subsets
- Representative technique: Property/test matrix: for each test case, what properties/behaviors will be demonstrated?
V-Model of Development and Testing Activities

- Specify Requirements
  - Requirements Review
  - Develop System/Acceptance Tests
  - Design
    - Design Review
    - Develop Integration Tests
    - Code
      - Code Review
      - Develop Unit Tests
  - Execute System Tests
  - System/Acceptance Tests Review
  - Execute Integration Tests
    - Integration Tests Review/Audit
    - Execute Unit Tests
      - Unit Tests Review/Audit

University of California, Irvine
Incremental Development of Tests

- Acceptance test plan (and tests): develop during requirements analysis
- Integration test plan (and test): develop during system architecture and detailed design specification
- Unit test plan (and tests): develop during implementation
ICS 52 Requirements Analysis Exercise

- Develop a requirements specification and acceptance test plan for the class project
- TAs are the customer