Capturing the requirements

- Requirement: a feature of the system or a description of something the system is capable of doing in order to fulfill the system's purpose
- Three kinds of requirements:
  - those that absolutely must be met
  - those that are highly desirable but not necessary
  - those that are possible but could be eliminated

Why are requirements important?

- Projects fail because (source: Standish Group survey 1994)
  - 13.1% incomplete requirements
  - 12.4% lack of user involvement
  - 10.6% lack of resources
  - 9.9% unrealistic expectations
  - 9.3% lack of executive support
  - 8.7% changing requirements and specifications
  - 8.1% lack of planning
  - 7.5% system no longer needed
Requirements Elicitation

Requirements documents

- **Requirements definition**: complete listing of what the customer expects the system to do
- **Requirements specification**: restates the definition in technical terms so that the designer can start on the design
- **Configuration management**: supports direct correspondence between the two documents

Configuration management

- Set of procedures that track
  - requirements that define what the system should do
  - design modules that are generated from requirements
  - program code that implements the design
  - tests that verify the functionality of the system
  - documents that describe the system
Functional vs. non-functional requirements

- Functional: describes an interaction between the system and its environment.
- Examples:
  - System shall communicate with external system X.
  - What conditions must be met for a message to be sent.
- Non-functional: describes a restriction or constraint that limits our choices for constructing a solution.
- Examples:
  - Paychecks distributed no more than 4 hours after initial data are read.
  - System limits access to senior managers.

Use Cases

- = a way to determine functional requirements
- partition system into set of logical, minimally related pieces
  - each piece describes some way how system will function (the particular use case)
- Advantages
  - can examine each case separately
  - can use cases for effort estimation
  - can track development in terms of use cases

Types of requirements

- Physical environment
- Interfaces
- Users and human factors
- Functionality
- Documentation
- Data
- Resources
- Security
- Quality assurance
Characteristics of requirements

- Are they correct?
- Are they consistent?
- Are they complete?
- Are they realistic?
- Does each describe something the customer needs?
- Are they verifiable?
- Are they traceable?

Static descriptions of requirements

- Indirect reference
  - Example: \( k \) equations in \( n \) unknowns
- Recurrence relations
  - Example: \( F(0) = 1; F(1) = 1; F(n+1) = F(n) + F(n-1) \)
- Axiomatic definition
- Expression as a language
  - Example: Backus-Naur form
Data abstraction

<table>
<thead>
<tr>
<th>Semester record</th>
</tr>
</thead>
<tbody>
<tr>
<td>Semester type</td>
</tr>
<tr>
<td>Semester date</td>
</tr>
<tr>
<td>Grade point average</td>
</tr>
<tr>
<td>Completed hours</td>
</tr>
<tr>
<td>Semester type</td>
</tr>
<tr>
<td>(Fall, Spring, Summer)</td>
</tr>
<tr>
<td>Address information</td>
</tr>
<tr>
<td>Telephone number</td>
</tr>
<tr>
<td>Street address</td>
</tr>
<tr>
<td>City</td>
</tr>
<tr>
<td>State</td>
</tr>
<tr>
<td>Postal code</td>
</tr>
</tbody>
</table>

Student record

<table>
<thead>
<tr>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student number</td>
</tr>
<tr>
<td>Address information</td>
</tr>
<tr>
<td>Number of semesters</td>
</tr>
<tr>
<td>(Semester record)</td>
</tr>
</tbody>
</table>

Dynamic descriptions

Decision tables

Table 4.1. Decision table.

<table>
<thead>
<tr>
<th>Rule 1</th>
<th>Rule 2</th>
<th>Rule 3</th>
<th>Rule 4</th>
<th>Rule 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>High standardized exam scores</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>High grades</td>
<td>-</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>Outside activities</td>
<td>-</td>
<td>-</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>Good recommendations</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>T</td>
</tr>
<tr>
<td>Send rejection letter</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Dynamic descriptions (2)

Functional descriptions and transition diagrams

\[ f(S_k, C_j) = S_k \]

Table 4.2. Transition table.

<table>
<thead>
<tr>
<th>Current state</th>
<th>Input</th>
<th>Next state</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_1</td>
<td>0</td>
<td>S_2</td>
</tr>
<tr>
<td>S_2</td>
<td>1</td>
<td>S_3</td>
</tr>
<tr>
<td>S_3</td>
<td>0</td>
<td>S_4</td>
</tr>
<tr>
<td>S_4</td>
<td>1</td>
<td>S_5</td>
</tr>
<tr>
<td>S_5</td>
<td>0</td>
<td>S_6</td>
</tr>
<tr>
<td>S_6</td>
<td>1</td>
<td>S_1</td>
</tr>
</tbody>
</table>
**Dynamic descriptions (3)**

- Event tables

<table>
<thead>
<tr>
<th>Mode</th>
<th>Event 1</th>
<th>Event 2</th>
<th>Event 3</th>
<th>Event 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graphics</td>
<td>X</td>
<td>Action 8</td>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td>Architecture</td>
<td>Action 3</td>
<td>X</td>
<td>Actions 5 and 6 in parallel</td>
<td>0</td>
</tr>
<tr>
<td>Native</td>
<td>0</td>
<td>Action 4</td>
<td>Actions 1, 2 and 5</td>
<td>Action 7</td>
</tr>
</tbody>
</table>

- Petri nets

**Object-oriented specifications**

- Each entity in the system is an object.
- A method or operation is an action that can be performed directly by the object or can happen to the object.
- Encapsulation: the methods form a protective boundary around an object.
- Class hierarchies of objects encourage inheritance.
- Polymorphism: same method for different objects, each with different behavior

**Additional notations**

- Hierarchical techniques
- Warnier diagrams
- Data flow diagrams
- Software Requirements Engineering Methodology (SREM)
- Structured Analysis and Design Technique (SADT)
- Z
R_NET: PROCESS_TRANSACTION

INPUT_INTERFACE_ACCOUNT_REQUEST_RECORD
EXTRACT_DATES
DO (REQUEST=TRANSACTION)
RECORD_TRANSACTION
TERMINATE
OTHERWISE
FIND_ACCOUNT-RECORDS
COMPUTE_SAVINGS_BALANCE
AND COMPUTE_CHECKING_BALANCE
AND COMPUTE_MONEY-MARKET_BALANCE
PRINT_BALANCES
TERMINATE
END

SREM steps

<table>
<thead>
<tr>
<th>Phase</th>
<th>Focus</th>
<th>Criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>Define kernel</td>
<td>Identify input and output, R-nets, transformations.</td>
<td>All input messages processed. All output messages generated.</td>
</tr>
<tr>
<td>Establish baseline</td>
<td>Clean up database: Plot R-nets.</td>
<td>All naming consistent.</td>
</tr>
<tr>
<td>Define data</td>
<td>Define input, output for each transformation.</td>
<td>No data used before given a value.</td>
</tr>
<tr>
<td>Establish traceability</td>
<td>Generate consistent traceability requirements.</td>
<td>All top requirements satisfied.</td>
</tr>
<tr>
<td>Simulate functionality</td>
<td>Simulate subsystem functions performed.</td>
<td>Validation that all are processed correctly.</td>
</tr>
<tr>
<td>Identify performance requirements</td>
<td>Define traceable, testable performance subsystem requirements.</td>
<td>Each path constrained by response time and accuracy.</td>
</tr>
<tr>
<td>Demonstrate feasibility</td>
<td>Rapid prototype of all critical algorithms.</td>
<td>Accuracy requirements satisfied by prototype algorithms.</td>
</tr>
</tbody>
</table>

Z example

ST = Key
a
INIT
st: ST
st' = {}
INSERT
st, st': ST
k: KEY
v: VAL
k ∈ dom(st) ∧ st' = st ∪ {k'a
v
LOOKUP
st, st': ST
k: KEY
v: VAL
k ∈ dom(st) ∧ v' = st(k) ∧ st' = st
DELETE
st, st': ST
k: KEY
st' = st ∩ {k

CS 1530 Software Engineering Fall 2004
Prototyping requirements

- Throw-away prototypes
- Evolutionary prototypes
- Rapid prototypes

Requirements documentation

- Requirements definition document: what the customer wants
  - general purpose
  - background and objectives of system
  - description of customer-suggested approach
  - detailed characteristics
  - operational environment

- Requirements specification document: what the designers need to know

Example

4.1.3.1 INITIATE TRACK ON IMAGE. Logical processing shall be done to INITIATE TRACK ON IMAGE. This shall have as input HANDOVER DATA. This shall have as output HOIQ, STATE DATA, and IMAGE_ID. This logical processing shall, when appropriate, identify a new instance of IMAGE. This logical processing, when appropriate, shall identify the type of entity instance as being IMAGE ON TRACK. NOTE: A request for pulses is made by entering a formal record into the HOIQ which feeds the pulse-sending procedures.

ALPHA: INITIATE_TRACK_ON_IMAGE.
INPUTS: HANDOVER_DATA.
OUTPUTS: HOIQ, STATE_DATA, IMAGE_ID.
CREATES: IMAGE
SETS: IMAGE_ON TRACK.
DESCRIPTION: "A 4.1.3.1A REQUEST FOR PULSE IS MADE BY ENTERING A FORMAL RECORD REQUEST INTO THE HOIQ WHICH FEEDS THE PULSE SENDING PROCEDURES."
Participants in requirements process

- Contract monitors
- Customers and users
- Business managers
- Designers
- Testers
- Requirements analysts

How developers see users

- Users don’t know what they want.
- Users can’t articulate what they want.
- Users have too many needs that are politically motivated.
- Users want everything right now.
- Users can’t prioritize needs.
- Users refuse to take responsibility for the system.
- Users are not committed to system development project.
- Users are unwilling to compromise.
- Users can’t remain on schedule.

How users see developers

- Developers don’t understand operational needs.
- Developers place too much emphasis on technicalities.
- Developers try to tell us how to do our jobs.
- Developers can’t translate clearly-stated needs into a successful system.
- Developers say no all the time.
- Developers are always over budget.
- Developers are always late.
- Developers set unrealistic standards for requirements definition.
- Developers are unable to respond quickly to legitimately changing needs.
- Developers are always late.

Table 4.6. Requirements validation techniques.

<table>
<thead>
<tr>
<th>Manual techniques</th>
<th>Automated techniques</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reading</td>
<td>Automated cross-referencing</td>
</tr>
<tr>
<td>Manual cross-referencing</td>
<td>Automated models to enact functions</td>
</tr>
<tr>
<td>Interviews</td>
<td>Mathematical proofs</td>
</tr>
<tr>
<td>Reviews</td>
<td></td>
</tr>
<tr>
<td>Checklists</td>
<td></td>
</tr>
<tr>
<td>Manual models to check functions and relationships</td>
<td></td>
</tr>
<tr>
<td>Scenarios</td>
<td></td>
</tr>
<tr>
<td>Mathematical proofs</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Automated techniques</td>
<td>Automated models to enact functions</td>
</tr>
<tr>
<td>Automated cross-referencing</td>
<td>Prototype</td>
</tr>
</tbody>
</table>

CS 1530 Software Engineering Fall 2004
Requirements review

- Review goals and objections of system.
- Compare requirements with goals and objectives.
- Describe operational environment.
- Examine
  - interfaces
  - information flow
  - functions
- Check for omissions, incompleteness, inconsistency.
- Document risk.
- Discuss how system will be tested.

Choosing a specification technique

- Applicability
- Implementability
- Testability/simulation
- Checkability
- Maintainability
- Modularity
- Level of abstraction/expressibility
- Soundness
  - Verifiability
  - Run-time safety
  - Tools maturity
  - Looseness
  - Learning curve
  - Technique maturity
  - Data modeling
  - Discipline

Information system example

Advertising Campaign = *Entity. Records the conditions and aims for a campaign to advertise a product.*

Campaign Number + Campaign Start Date + Campaign End Date
+ Target Audience + Target Rating Percentage
+ Campaign Predicted Rating + Cam paign Budget Total
+ Required Spot Duration
+ (Required Spot Duration)
*Work necessary to remove or justify the repeating group.*

Target Audience = *Data element. The audience at which a campaign is aimed. See Audience Type for values.*

Audience Type = *Data element. Used to classify ratings figures.*
[Romans | Romansulus | Adults | Men | Women | Children]
Real-time example

STATE SEL_WORKING;
INPUT Clear(line);
DECISION ((cond(line) == OOS) || (cond(WORK) == OOS));
(TRUE):
    TASK 'badinput = 1';
    NEXTSTATE;
ENDDECISION;
TASK 'cond[line] = NORMAL';
DECISION ((line == PROT) && (cond[WORK] > cond[PROT]));
(TRUE):
    NEXTSTATE SEL_PROTECTION;
ENDDECISION;
NEXTSTATE;
ENDSTATE SEL_WORKING;

CLAIM:
(WPENV FIRSTMSG Done)
&& (envmsg == CONDSWITCH)
&& (sel != psel))
IMPLIES (cond[sel] <= pcond[psel])
UNTIL (sel == psel);
ENDCLAIM;

CS 3530 Software Engineering Fall 2004