Topics

• Software requirement
  – What is it and how to specify it
  – Data and operations: basis of software specification

• Model for software and its environment
  – Data (static part): ER model
  – Operations (dynamic part): Finite State Model

• Decomposition of domain-model for subsystem identification
  – Software and its environment as two cooperative components in the decomposition
  – Interface: Sharing of state-information.
SHAPE OF A SOFTWARE

Software: A man-made object

Shape or "look" of Software:

- It is not the look when printed on paper!
- What is it then?
  - Is it the external-look (view or features) of the software?
  - Is it the internal-view (features) of the software?

- What is an external software feature?
- What is an internal software feature?

If we think of shape as an external-view, then shape of software = its requirements.
WHAT IS YOUR ...

• *favorite* software?

State some of its most important features:

–

–

–

• *dream* software that you would want to write??

What would some of its most important features be:

–

–

–
REQUIREMENTS FOR DISPLAY OF DATES AND DAYS IN A CALENDER-MONTH

- Organization of weeks (rows vs. columns) and the alignments of dates (left/right adjusted) within a column.

- Organization of months in a multi-column format.

<table>
<thead>
<tr>
<th>August</th>
</tr>
</thead>
<tbody>
<tr>
<td>S M T W T F S</td>
</tr>
<tr>
<td>1 2 3 4</td>
</tr>
<tr>
<td>5 6 7 8 9 10 11</td>
</tr>
<tr>
<td>12 13 14 15 16 17 18</td>
</tr>
<tr>
<td>19 20 21 22 23 24 25</td>
</tr>
<tr>
<td>26 27 28 29 30 31</td>
</tr>
</tbody>
</table>

For a two-column calendar;

- Which of the two is preferred?
- Does it depend on the organization of the weeks in a month?

<table>
<thead>
<tr>
<th>January</th>
<th>February</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>March</th>
<th>April</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>August</th>
</tr>
</thead>
<tbody>
<tr>
<td>S M T W T F S</td>
</tr>
<tr>
<td>5 12 19 26</td>
</tr>
<tr>
<td>6 13 20 27</td>
</tr>
<tr>
<td>7 14 21 28</td>
</tr>
<tr>
<td>8 15 22 29</td>
</tr>
<tr>
<td>9 16 23 30</td>
</tr>
<tr>
<td>10 17 24 31</td>
</tr>
<tr>
<td>11 18 25</td>
</tr>
</tbody>
</table>

- Why don’t we have row-labels for the top two tables?
A ROOM-LAYOUT DESIGN

**Key Design Features** (suitable for stores in a shopping-mall):

- Separate entrances allow independent store-owners and to have independent store-hours and open/close operations.
- Linear arrangement makes it easy for customers to visit multiple shops in the same shopping trip.

<table>
<thead>
<tr>
<th>Betty’s Cake</th>
<th>Specialty Candies</th>
<th>Kitchen &amp; Kitchen</th>
<th>Bed &amp; Bath</th>
<th>John’s Hardware</th>
</tr>
</thead>
</table>

**Question:** Which way the doors should open and why?
A ROOM-LAYOUT DESIGN: Contd.

The Same Design For Rooms In A House: Not good.

- What makes it a bad design here?

<table>
<thead>
<tr>
<th>Bath</th>
<th>Bedroom</th>
<th>Kitchen</th>
<th>Bedroom</th>
<th>Living room</th>
</tr>
</thead>
</table>

Different goals/requirements require different designs.
REQUIREMENTS: THE BASIS OF GOAL-ORIENTED SOFTWARE DESIGN

Input (normal + erroneous) → SOFTWARE (unknown; black-box) → Output (normal + ???)

**Question:** What should be the output for erroneous inputs?

**Requirements** (basis of acceptance test):

1. A set of criteria for the "external behavior" of the system ("what" vs. "how", internal behavior) of the system, that distinguishes the acceptable solutions from the others.

2. The requirements must be testable/verifiable.

![Requirements diagram]

Acceptable solutions $S_i$ Unacceptable alternatives

**Question:** Why do we emphasize here the "external behavior"?

**Diagram:**

Users/Customer → each requirement is a bridge → Software Product
REQUIREMENTS ARE DIFFICULT TO VISUALIZE AND TO STATE

Example.

• Some common and some uncommon designs/shapes of a simple cup for drinking-water (without the handle).

![Cup Designs](image)

Question:

•? State some requirements to filter out the uncommon shapes.
•? Are there other requirements that are not perhaps "seen" from these examples?

Formulating requirements:

• Always use example cases as a guide.

Requirements: You get what you ask for.
DESCRIBING A REQUIREMENTS CAN BE DIFFICULT EVEN WHEN "see" IT

Example. Consider the four situations that may arise for a set of products with the features $A$ and $B$.

Question:

• Find a description that distinguishes the third situation from the others.
REQUIREMENTS, SOFTWARE, AND SOFTWARE-PROCESS

Requirements: A high-level, black-box, user-view of software.
− Black-box: requirements come before software is built.
− High-level: requirements focus on "what" and not "how".

Functional requirements:
− What inputs, including applicable constraints, does it need?
− What outputs, including their properties, does it produce?
− What functionality, i.e., relationship holds between the inputs and the outputs?

Question:
• What are some (comparatively) minor issues of a software?
• What situations make us look at the requirements once again after building a software?
EXAMPLE OF REQUIREMENTS

- Suppose we have (or plan to develop) a software to count #(words in a text-file) and #(characters in those words).

Input: a text-file

<table>
<thead>
<tr>
<th>Software</th>
<th>Output: values for wordCount and charCount</th>
</tr>
</thead>
</table>

Requirements:

- Functionality: Count #(words in a text-file) and #(characters in those words).
- An input oriented requirement (part of pre-condition):
  There is no restriction on the word-size. (Alternatively, one could say that each word is, say, \( \leq 20 \) characters.)
- An output-oriented requirement (part of post-condition):
  wordCount \( \leq \) charCount, both being 0 for an empty file.

Always Explain Requirements (with examples):

- For the file shown below as a string and \[\text{e}\] for end-of-file, we have word/char counts = 5/22 or 6/21 (including ".").

  This is a short text-file. \[\text{e}\]

Question:

- Give a requirement related to word-separators Which category of functional requirement does this fall into?
TWO WordCharCount FUNCTIONS

- What would be some requirements that would make one or both of these invalid?

```c
#define WORDLEN 20
void WordCharCounts(FILE *inFile)
{
    int i;
    char word[WORDLEN+1];
    wordCount = charCount = 0;
    while (fscanf(inFile, "%s", word) > 0) {
        wordCount++;
        for (i=0; i<=WORDLEN; i++)
            if (‘\0’ == word[i]) break;
        else charCount++;
    }
}

void WordCharCounts(FILE *inFile)
{
    char ch;
    wordCount = charCount = 0;
    while (fscanf(inFile, "%c", ch) > 0)
        if ((ch != ’ ‘) && (ch != ’\n’) ) {
            charCount++; wordCount++;
            while (fscanf(inFile, "%c", ch))
                if ((ch != ’ ’) && (ch != ’\n’) )
                    charCount++;
                else break;
        }
}
```

Question:

- How do you defend that none of these functions find the number of lines in the text file? How could you fail to defend?
STATING REQUIREMENTS IS DIFFICULT

Requirements for WordCharCount-function:
Determine #(words in a textfile) and #(characters in those words).

How good is the above requirement-statement:

• Does it specify the input/output and their relationship?
  – What domain concepts are used to express the requirements?

• Is there any ambiguity in the requirement?
  – Is a punctuation mark (’,’) next to a word part of it?
  – Is "John’s" one word? How about "don’t"? Is "open/close" one word or two words?
  – What happens when a word is split between two lines as in
    Mr. Johnson is an extraordinary person.

  – What characters other than blanks and new-lines are considered word-separators?

• Is this requirements testable?

Determining requirements is not an easy task, but good requirements are must for a successful software.
FIND SOME GOOD REQUIREMENTS FOR A FLOWCHART-DISPLAY SOFTWARE

- Use the following two example displays below as a guide.
- $15 and $10 rewards for the two top requirements.
EXAMPLE OF REQUIREMENTS FOR AN AUTOMATED LIBRARY CHECK-OUT SYSTEM

Requirements related to a book:
- Searchable by author-name, book-title, and subject-keyword.
- Check availability, return-date, and hold-status.
- Check-out, renew, put/cancel hold, and return.
- Report lost, stolen, damaged/out-of-service.

Requirements related to renew operation:
- Can be renewed only by the current borrower.
- Can be renewed (on-line) only if the number of times renewed is below the renewal-limit.
- Can be renewed only if there is no hold.

Requirements related to Overdue/Early-return notice:
- Issued when the due-date is passed.
- Issued when there is an existing "hold" on the book.

Requirements for borrowing an on-hold book:
- Within a specified time-limit after the book is returned.

Relate each requirement with an entity or an operation.
**FINITE STATE MODEL**

**FSM for a book:**
- Assume for now that at most one hold on a borrowed book, and all books are returned (no lost-book).
- POR = Periodic overdue/early-return-required notice.
- BHA = book-on-hold available notice.

![Finite State Machine Diagram]

**Example of Constraints:**
- renew: (renewer.id = borrower.id) and (there is no hold)
- POR: POR-notice is sent only a fixed max. number of times.
- BHA: BHA-notice is sent only a fixed max number of times.
EXTENSION OF BOOK-FSM TO THE CASE OF MULTIPLE HOLDS

Additional Condition-Guards:

- $C_1 = \text{Has one hold and } C_2 = \text{Has } \geq 2 \text{ holds.}$
• A customer may at any time have a minimum of 0 books and a maximum of $n$ books borrowed.

• A book may at any time be borrowed by a minimum of 0 and a maximum of 1 customer.

• A customer may at any time have put hold on a minimum of 0 books and a maximum of $m$ books

• A book may at any time have hold by a minimum of 0 and a maximum of 1 customer.

• The books in BOOKS-HL cannot have the two minimum cardinalities 0 at the same time.
ALL-BOOKS:

\((BookId, \text{BookType}, \text{Title}, \text{Author}, \text{Publisher}, \text{PurchaseDate}, \text{PurchasePrice})\)

SEARCH & USE-STATISTICS:

\((BookId, \text{SearchCount}, \text{BorrowCount}, \text{TotalUseDuration})\)

FINES & LOAN-DURATIONS:

\((\text{BookType}, \text{CustomerType}, \text{LoanDuration}, \text{LoanRenewalDuration}, \text{HoldingPeriod}, \text{Fine}, \text{ReplacementCostPolicy})\)

BOOKS-HL:

\((BookId, \text{BookType}, \text{HasHold})\)

CUSTOMERS:

\((\text{CustomerId}, \text{CustomerType}, \text{Address}, \text{TotalBorrowCount}, \text{TotalHoldCount}, \text{TotalLateRetCount}, \text{TotalLostBookCount})\)

BORROWS:

\((BookId, \text{CustomerId}, \text{LoneDate}, \text{ReturnDate}, \text{DueDate}, \text{ReminderCount})\)

HOLDS:

\((BookId, \text{CustomerId}, \text{HoldReqDate}, \text{HeldStartDate}, \text{HeldEndDate})\)
Not Shown In The Model:

- For multiple holds, we need additional attribute `holdSequenceNum` in HOLDS-relation.
- The person currently borrowing a book cannot put a hold on it.

Note:

- The same person cannot put multiple hold on a book is captured by the key `(BookId, CustomerId)` of HOLDS.
IN SEARCH OF THE REQUIREMENTS
FOR THE FLOWCHART-DISPLAY SOFTWARE

Entities and attributes:

- Nodes: position, shape, and content, entry-position (for incoming lines to it), and exit-position (for outgoing lines from it).
- ConnectingLines: shape, start/end nodes.

Non-spatial Relationships (not related to user-operation):

- Block-structure among nodes; each block is a "super-node" with its own shape.
- Nested structure among blocks.

Spatial Relationships (display operations, but no user-operation):

- Within a block among its nodes and their connecting lines.
- Among nodes, blocks, and their connecting lines (including spatial separation among them).

Question: Does this help to formulate the requirements?
ORTHOGONAL DECOMPOSITION OF FINITE-STATE MODEL

Example: A pay-and-view slide-show system with 6 operations.

$c_1$: Pay the amount $c_1$ to view the slide-show  
$U$: Unlock the viewer  
$p$: Push a knob to select a show and start the pre-view  
$s$: See the slide-show  
$L$: Lock the viewer  
$c_2$: Pay a reduced amount $c_2 < c_1$ to skip the slide-show and select a new one.

There are no final-states.

Question:

- If we want to delegate some of the operations to a subsystem, what would be such a subset of operations?
ONE OF MANY POSSIBLE SOLUTIONS

Subsystem Operations: \{U, L\}.

- Decompose the global model based on the subset of operations.

Resulting Models in The Decomposition:

- Either one can be taken as the environment and the other as a subsystem. Here, the state \(\sigma_{1,5} = \{\sigma_1, \sigma_5\}\), etc.

(i) The environment model \(M_E\).

(ii) The subsystem model \(M_S\).

Question: What makes this decomposition valid or useful?
COMPOSITION OF THE SUBMODELS GIVES THE ORIGINAL MODEL

- Valid = No loss of information in the decomposition.

(i) The environment model $M_E$.

(ii) The subsystem model $M_S$.

(iii) The product composition $M_E \times M_S$. 
ANOTHER VALID DECOMPOSITION

(i) The environment model $M_E$.

(ii) The subsystem model $M_S$. 

The original model.
AN INVALID DECOMPOSITION

(i) The finite-state model $M_E$.

(ii) The finite-state model $M_S$. 

The original model.
CONCLUSION

- Requirements and models are the two firsts for building a good and successful software.
- Building requirements is hard
  - Build requirements around the data and operations.
  - Build ER-model for data
  - Build finite-state model for operations
- Identify subsystems by decomposition of finite-state models
  - This helps in simplifying and defining the overall system architecture