Requirements Elicitation

Why is it difficult?
What to elicit?
How to elicit?

What are RE Processes?

A Framework
for initial model construction & subsequent reengineering
3 fundamental concerns: understand (formally) describe attain agreement on

 validations

🚶 Elicitation
determine what’s really needed, why needed, whom to talk to acquire as much knowledge as possible

🚶 Specification
produce a (formal) RS model: translate "vague" into "concrete", etc. make various decisions on what & how

🚶 Validation
assure that the RS model satisfies the users’ needs

Lawrence Chung
Why is it difficult?

A wicked problem

- many sources
  - customers, tellers, other employees
databases, files, chapters, literature, standards, legal issues
  - surveys of similar applications
  - (what are they, who are they, how successful are they)
- different views on wants and needs
- different notations, mental models

Uncertainties

Do we know whom to talk to?
Do they know what they want/need?
Do they change their minds?

User

User reqs.

Elicitation

Problem Domain

acct'g, banking, loan policies, etc.

- many sources
  - domain experts (accountants, bankers, loan managers, etc.)
databases, files, chapters, literature, standards, legal issues

Lawrence Chung

Sources of requirements

Unconstrained

(many opinions of a group of mgrs)
- DSS
- Corporate Acctg System
- M manufacturers of OS
- Enhancements to Corporate Acctg System
- Airline Flight Control System
- Missile Guidance System
  - (analysis of docs, specialized domain K. from engineers)

Highly Constrained

Plenty of a priori knowledge
Well-established discipline
Well-defined product & process

% of Reqs Gathered from People

Fraction of reqs. elicited from people increases as constraints on the sw reqs. process decrease

Lawrence Chung
Why is it difficult?

**A wicked problem**

- many sources: customers, tellers, other employees, databases, files, chapters, literature, standards, legal issues, surveys of similar applications (what are they, who are they, how successful are they)
- different views on wants and needs
- different notations, mental models

**Uncertainties**

- Do we know whom to talk to?
- Do they know what they want/need?
- Do they change their minds?

**Problem Domain**

- many sources: accountants, bankers, loan managers, etc., databases, files, chapters, literature, standards, legal issues

**Say-do problem:** people know how to do things they normally don’t describe (tacit knowledge); descriptions of such things may be highly inaccurate

**Identification process complex (repetitive interactions)**

**Communication, coordination process complex**

**Requirements volatility:**

- Reqs. change because the problem being solved changes, because people’s perception changes, because some involved persons were not contacted or were contacted but not in an appropriate manner.

What to elicit?

**Four Worlds of RE**

- Subject World: FRs, NFRs
- System World: FRs, NFRs
- Developer World: analysts, specifiers, designers, mgrs
- Enterprise Model: FRs, NFRs

Lawrence Chung
**How to elicit?**

**How to do RE?**

**Major themes of the course**

**Types of Errors**

- Jet Propulsion Laboratory [Kelly92]
  - Nearly 2/3 requirements defects are due to omission of key information
  - Techniques of completeness
  - Identifying system operations
  - Clarifying business objectives

- The Naval Research Laboratory
  - Navy AV-8 fighter aircraft operational flight program
  - Ongoing research since mid-70's
  - 17% of requirements errors were nonclerical
  - Techniques of completeness
  - Identifying system operations
  - Clarifying business objectives

- Error detection and removal in defining FRs, NFRs

- Why, What & How

---

**Four Worlds of RE**

**Enterprise Model**

- Subject World
  - System World
    - User World
      - Tellers, clients, mgrs...
  - Developer World
    - Analysts, specifiers, designers, etc.

---

**How to elicit?**

**How to do RE?**

**Major themes of the course**

**Error detection and removal**

<table>
<thead>
<tr>
<th>Conceptuality</th>
<th>Formality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Natural language</td>
<td>Semi-formal notations</td>
</tr>
<tr>
<td>customer/users</td>
<td>defects</td>
</tr>
</tbody>
</table>

**D Clarity**

- shall, will, must, should, etc. => formality (criticality)
- A or B => formality (inclusive-or-exclusive-or)
- mostly A => formality (A in cases C1, C2, ..., Cn)
- perhaps/could/may/might A => formality (A in cases C1, C2, ..., Cn)
- by and large, often, frequently A => formality (A in cases C1, C2, ..., Cn)
- A or ¬A => formally ignore as tautology, but may mean something
How to elicit?

How to do RE?

Major themes of the course

C Error detection and removal

<table>
<thead>
<tr>
<th>Conceptuality</th>
<th>Formality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Natural language</td>
<td>Semi-formal notations</td>
</tr>
<tr>
<td>customer/users</td>
<td>Formalisms</td>
</tr>
<tr>
<td>defects ↑</td>
<td>defects ↓</td>
</tr>
</tbody>
</table>

D Consistency

\[ A \text{ and } \neg A \Rightarrow \text{formally false, ignorable, but could be typo} \]

\[ A \implies B \]

\[ A \]

\[ \neg B \]

\[ \text{modus ponens} \]

\[ A \text{ or } B \]

\[ \neg A \]

\[ \neg B \]

Slow children at play \(=\) semantic issue

D Correctness (external Consistency)

When two trains approach each other at a crossing, both shall come to a full stop and neither shall start up again until the other is gone. [Kansas legislature, early 1890’s]

\[ \Rightarrow \text{scenario analysis and/or} \]

\[ \Rightarrow \text{formalism} \]

Call forwarding: (B \rightarrow C); (A \rightarrow C); (D \rightarrow A)

(B \rightarrow C); (C \rightarrow B)
How to elicit?

How to do RE?

Major themes of the course

C Error detection and removal

<table>
<thead>
<tr>
<th>Conceptuality</th>
<th>Formality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Natural language</td>
<td>Semi-formal notations</td>
</tr>
<tr>
<td>customer/users</td>
<td>Formalisms</td>
</tr>
<tr>
<td>defects</td>
<td>modellers/specifiers</td>
</tr>
<tr>
<td></td>
<td>defects</td>
</tr>
</tbody>
</table>

D Completeness

hard => elicitation techniques
(logical formalism is modern philosophy)
=> use of "ontological" primitives
goals, agents, decisions, rationale
entities, activities, constraints

=> use of "epistemological" primitives

Lawrence Chung

How to do RE?

Major themes of the course

C Error detection and removal

<table>
<thead>
<tr>
<th>Conceptuality</th>
<th>Formality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Natural language</td>
<td>Semi-formal notations</td>
</tr>
<tr>
<td>customer/users</td>
<td>Formalisms</td>
</tr>
<tr>
<td>defects</td>
<td>modellers/specifiers</td>
</tr>
<tr>
<td></td>
<td>defects</td>
</tr>
</tbody>
</table>

D Overspecification

algorithms (sorting, searching, routing, serialization, normalization, etc.)
data structures (stack, queue, tree, graph, heap, etc.)

D Technically unsolvable problems

scheduling, pattern recognition, etc.

Also, do not worry at this time about acquiring the resources to build the house. You our first priority is to develop detailed plans and specifications. Once I approve these plans, however, I would expect the house to be under roof within 48 hours.

While you are designing this house specifically for me, keep in mind that sooner or later I will have to sell it to someone else. It therefore should have appeal to a wide variety of potential buyers. Please make sure before you finalize the plans that there is a consensus of the population in my area that they like the features this house has.

I advise you to run up and look at my neighbor's house he constructed last year. We like it a great deal. It has many features that we would also like in our new home, particularly the 75-foot swimming pool. With careful engineering, I believe that you can design this into our new house without impacting the final cost.

Lawrence Chung
How to elicit?

E Teleological view of systems

tel-e-o-l-o-gy n., pl. -gies. 1. The philosophical study of design or purpose in natural phenomena. 2. The use of ultimate purpose or design as a means of explaining natural phenomena.

A system has a set of goals it seeks to attain; a system’s behavior is explained in terms of its goals.

-> traceability, justifiability

E.g., Library System

- send an overdue notice via mail
- send an early notice

why? it takes time and money

to ensure books are regularly available

what’s regular availability?

books in shelf

why?

to satisfy book request

why?

to make the system effective

high-level goal

Alternatives

Goal-Directed Strategy

post a goal
refine the goal
do everything to meet the goal
consider alternatives
analyze tradeoffs

User

Elicitation

Problem Domain

Goal-Directed Strategy -> Goal structure

Goals initially stated by the client are incrementally refined into subgoals

traditionally AND/OR decompositions

goal1

AND

goal2

goal3

to satisfy the parent, satisfy all its descendants

goal1

OR

goal2

goal3

to satisfy the parent, satisfy any of its descendants
How to elicit?

**Goal-Directed Strategy 1: Classical Logic Approach**

**Expressive Power**

"Propositional and predicate logic provide all the basic concepts needed for a systematic engineering design methodology"


**Example: "good old vending machine"**

**step 1:** identify the top goal ---> "serve_customer"
identify more domain-specific goals ---> "dispensing cash",
"serving coffee", "vending candy bars", "shining shoes", etc.

**step 2:** examine what is (really) needed to satisfy the goal.
- what product the customer wants
- if the vending machine can dispense the customer’s choice (has sufficient inventory?)
- if the customer has the resources (cash or credit card) and is ready to pay for the selection
- if the customer has deposited more money than necessary for the purchase
- if the selection and any change were properly dispensed.

---

**Executable specification (e.g., in Prolog)**

```
serve_customer :-
customer_selection(Product),
selection_availability(Product),
customer_payment(Product, Payment),
vend_payment(Product, Refund).
```
How to elicit?

E    Goal-Directed Strategy 1: Classical Logic Approach
F  Example: "good old vending machine"

**step 3:** incremental expansion
E.g., the customer needs to know about the choices, and these choices should be displayed in some form; the machine should be able to accept the customer selection.

```
serve_customer
  AND
customer_selection(Product)
  AND
  selection_availability(Product)
  AND
  customer_payment(Product, Payment)
  AND
  vend_payment(Product, Refund)
```

F  --> Executable specification (e.g., in Prolog)
```
serve_customer :~
customer_selection(Product)
  selection_availability(Product)
  customer_payment(Product, Payment)
  vend_payment(Product, Refund).
```

**step 4:** how-to elaboration

```
customer_selection(Product) :~
display_choices
  get_customer_choice(Product).
```

---

**How to elicit?**

E    Goal-Directed Strategy 1: Classical Logic Approach
F  Example: "good old vending machine"

**step 4:** how-to elaboration

```
customer_selection(Product)
  selection_availability(Product)
  customer_payment(Product, Payment)
  vend_payment(Product, Refund)

serve_customer
  AND

display_choices
  get_customer_choice(Product)
```

F  --> Executable specification (e.g., in Prolog)
```
serve_customer :~
customer_selection(Product)
  selection_availability(Product)
  customer_payment(Product, Payment)
  vend_payment(Product, Refund).
```

```
customer_selection(Product) :~
display_choices
  get_customer_choice(Product).
```

---

**How to elicit?**

E    Goal-Directed Strategy 1: Classical Logic Approach
F  Example: "good old vending machine"

**step 4:** how-to elaboration

```
customer_selection(Product)
  selection_availability(Product)
  customer_payment(Product, Payment)
  vend_payment(Product, Refund)

serve_customer
  AND

display_choices
  get_customer_choice(Product)
```

F  --> Executable specification (e.g., in Prolog)
```
serve_customer :~
customer_selection(Product)
  selection_availability(Product)
  customer_payment(Product, Payment)
  vend_payment(Product, Refund).
```

```
customer_selection(Product) :~
display_choices
  get_customer_choice(Product).
```

---

**How to elicit?**

E    Goal-Directed Strategy 1: Classical Logic Approach
F  Example: "good old vending machine"

**step 4:** how-to elaboration

```
customer_selection(Product)
  selection_availability(Product)
  customer_payment(Product, Payment)
  vend_payment(Product, Refund)

serve_customer
  AND

display_choices
  get_customer_choice(Product)
```

F  --> Executable specification (e.g., in Prolog)
```
serve_customer :~
customer_selection(Product)
  selection_availability(Product)
  customer_payment(Product, Payment)
  vend_payment(Product, Refund).
```

```
customer_selection(Product) :~
display_choices
  get_customer_choice(Product).
```

---

**How to elicit?**

E    Goal-Directed Strategy 1: Classical Logic Approach
F  Example: "good old vending machine"

**step 4:** how-to elaboration

```
customer_selection(Product)
  selection_availability(Product)
  customer_payment(Product, Payment)
  vend_payment(Product, Refund)

serve_customer
  AND

display_choices
  get_customer_choice(Product)
```

F  --> Executable specification (e.g., in Prolog)
```
serve_customer :~
customer_selection(Product)
  selection_availability(Product)
  customer_payment(Product, Payment)
  vend_payment(Product, Refund).
```

```
customer_selection(Product) :~
display_choices
  get_customer_choice(Product).
```

---

**How to elicit?**

E    Goal-Directed Strategy 1: Classical Logic Approach
F  Example: "good old vending machine"

**step 4:** how-to elaboration

```
customer_selection(Product)
  selection_availability(Product)
  customer_payment(Product, Payment)
  vend_payment(Product, Refund)

serve_customer
  AND

display_choices
  get_customer_choice(Product)
```

F  --> Executable specification (e.g., in Prolog)
```
serve_customer :~
customer_selection(Product)
  selection_availability(Product)
  customer_payment(Product, Payment)
  vend_payment(Product, Refund).
```

```
customer_selection(Product) :~
display_choices
  get_customer_choice(Product).
```

---

**How to elicit?**

E    Goal-Directed Strategy 1: Classical Logic Approach
F  Example: "good old vending machine"

**step 4:** how-to elaboration

```
customer_selection(Product)
  selection_availability(Product)
  customer_payment(Product, Payment)
  vend_payment(Product, Refund)

serve_customer
  AND

display_choices
  get_customer_choice(Product)
```

F  --> Executable specification (e.g., in Prolog)
```
serve_customer :~
customer_selection(Product)
  selection_availability(Product)
  customer_payment(Product, Payment)
  vend_payment(Product, Refund).
```

```
customer_selection(Product) :~
display_choices
  get_customer_choice(Product).
```

---

**How to elicit?**

E    Goal-Directed Strategy 1: Classical Logic Approach
F  Example: "good old vending machine"

**step 4:** how-to elaboration

```
customer_selection(Product)
  selection_availability(Product)
  customer_payment(Product, Payment)
  vend_payment(Product, Refund)

serve_customer
  AND

display_choices
  get_customer_choice(Product)
```

F  --> Executable specification (e.g., in Prolog)
```
serve_customer :~
customer_selection(Product)
  selection_availability(Product)
  customer_payment(Product, Payment)
  vend_payment(Product, Refund).
```

```
customer_selection(Product) :~
display_choices
  get_customer_choice(Product).
```

---

**How to elicit?**

E    Goal-Directed Strategy 1: Classical Logic Approach
F  Example: "good old vending machine"

**step 4:** how-to elaboration

```
customer_selection(Product)
  selection_availability(Product)
  customer_payment(Product, Payment)
  vend_payment(Product, Refund)

serve_customer
  AND

display_choices
  get_customer_choice(Product)
```

F  --> Executable specification (e.g., in Prolog)
```
serve_customer :~
customer_selection(Product)
  selection_availability(Product)
  customer_payment(Product, Payment)
  vend_payment(Product, Refund).
```

```
customer_selection(Product) :~
display_choices
  get_customer_choice(Product).
```

---

**How to elicit?**

E    Goal-Directed Strategy 1: Classical Logic Approach
F  Example: "good old vending machine"

**step 4:** how-to elaboration

```
customer_selection(Product)
  selection_availability(Product)
  customer_payment(Product, Payment)
  vend_payment(Product, Refund)

serve_customer
  AND

display_choices
  get_customer_choice(Product)
```

F  --> Executable specification (e.g., in Prolog)
```
serve_customer :~
customer_selection(Product)
  selection_availability(Product)
  customer_payment(Product, Payment)
  vend_payment(Product, Refund).
```

```
customer_selection(Product) :~
display_choices
  get_customer_choice(Product).
```

---

**How to elicit?**

E    Goal-Directed Strategy 1: Classical Logic Approach
F  Example: "good old vending machine"

**step 4:** how-to elaboration

```
customer_selection(Product)
  selection_availability(Product)
  customer_payment(Product, Payment)
  vend_payment(Product, Refund)

serve_customer
  AND

display_choices
  get_customer_choice(Product)
```

F  --> Executable specification (e.g., in Prolog)
```
serve_customer :~
customer_selection(Product)
  selection_availability(Product)
  customer_payment(Product, Payment)
  vend_payment(Product, Refund).
```

```
customer_selection(Product) :~
display_choices
  get_customer_choice(Product).
```

---

**How to elicit?**

E    Goal-Directed Strategy 1: Classical Logic Approach
F  Example: "good old vending machine"

**step 4:** how-to elaboration

```
customer_selection(Product)
  selection_availability(Product)
  customer_payment(Product, Payment)
  vend_payment(Product, Refund)

serve_customer
  AND

display_choices
  get_customer_choice(Product)
```

F  --> Executable specification (e.g., in Prolog)
```
serve_customer :~
customer_selection(Product)
  selection_availability(Product)
  customer_payment(Product, Payment)
  vend_payment(Product, Refund).
```

```
customer_selection(Product) :~
display_choices
  get_customer_choice(Product).
```
How to elicit?

Goal-Directed Strategy 1: Classical Logic Approach
Example: "good old vending machine"

From design to requirements (avoid any erroneous, accidental design)
From requirements to design (ensure every requirement is met)

Rational Design
basis for alternatives, tradeoffs, rationale

Lawrence Chung