Goal-Oriented Requirements Engineering (GORE)

- Why and What?
- Classical Logic Approach
- Traceability

©Lawrence Chung
Why Goal-Oriented Requirements Engineering?
If the five following criteria are satisfied, then requirements engineering, in the strongest sense, is complete.

(1) There is a set \( R \) of requirements. Each member of \( R \) has been validated (checked informally) as acceptable to the customer, and \( R \) as a whole has been validated as expressing all the customer’s desires with respect to the software-development project.

(2) There is a set \( K \) of statements of domain knowledge. Each member of \( K \) has been validated (checked informally) as true of the environment.

(3) There is a set \( S \) of specifications. The members of \( S \) do not constrain the environment, they are not stated in terms of any unshared actions or state components, and they do not refer to the future.

(4) A proof shows that:

\[
S, K \vdash R
\]

This proof ensures that an implementation of \( S \) will satisfy the requirements.

(5) There is a proof that \( S \) and \( K \) are consistent. This ensures that the specification is internally consistent and consistent with the environment. Note that the two proofs together imply that \( S, K, \) and \( R \) are consistent with each other.

©Lawrence Chung
The requirements are complete if they are sufficient to establish the goal they are refining.


Teleology (Greek: telos: end, purpose) is the philosophical study of design and purpose. A teleological school of thought is one that holds all things to be designed for, or directed toward, a final result, that there is an inherent purpose or final cause for all that exists.

[Wikipedia]
How to Elicit?

Goal-oriented Requirements Elicitation

"... Requirements Engineering is the branch of Systems engineering concerned with real-world goals for, services provided by, and constraints on software systems. Requirements engineering is also about the relationships of these facts to precise specifications and to their evolution over time."

Where is Use Case Diagram?
What is Goal-Oriented Requirements Engineering?

Teleological view of systems -> Goal-Directed Strategy

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

<table>
<thead>
<tr>
<th>User</th>
<th>Elicitation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem Domain</td>
<td></td>
</tr>
</tbody>
</table>

©Lawrence Chung
Goal-Directed Strategy -> Goal structure

Goals initially stated by the client are incrementally refined into subgoals

traditionally AND/OR decompositions

<table>
<thead>
<tr>
<th>goal1</th>
<th>goal1</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>OR</td>
</tr>
<tr>
<td>goal2</td>
<td>goal2</td>
</tr>
<tr>
<td></td>
<td>goal3</td>
</tr>
</tbody>
</table>

to satisfy the parent, satisfy all its descendants  
to satisfy the parent, satisfy any of its descendants

©Lawrence Chung
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.
  
  => domain analysis

  **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.
How to elicit?

Goal-Directed Strategy 1: Classical Logic Approach

Example: "good old vending machine"

serve_customer

\( \text{AND} \)

customer_selection(Product)

selection_availability(Product)

customer_payment(Product, Payment)

vend_payment(Product, Refund)

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

serve_customer :-

customer_selection(Product),

selection_availability(Product),
customer_payment(Product, Payment),
vend_payment(Product, Refund).

H :- B1, ..., Bn. to show/solve H, show/solve B1 and ... and Bn.

e.g.,
sibling(X,Y) :- parent(Z,X), parent(Z,Y).

parent(X,Y) :- father(X,Y).

parent(X,Y) :- mother(X,Y).

©Lawrence Chung
How to elicit?

- Goal-Directed Strategy 1: Classical Logic Approach

  - 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.

  ![Diagram of logic representation]

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

  ```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).
  ```

Lawrence Chung
©Lawrence Chung
How to elicit?

Goal-Directed Strategy 1: Classical Logic Approach

Example: "good old vending machine"

step 4: how-to elaboration

---

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).

display_choices :- vertical_display,
                  horizontal_display.
```

© Lawrence Chung
How to elicit?

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

serve_customer

AND

customer_selection(Product)

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

vertical_display

display_choices

get_customer_choice(Product)

OR

horizontal_display

mouse_selection(Product)

button_selection(Product)

voice_selection(Product)

=> Dependency => Traceability.
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

©Lawrence Chung
More on Traceability

=> Dependency => Traceability:
   From design to requirements (avoid any erroneous, accidental design)
   From requirements to design (ensure every requirement is met)

What would be “forward traceability”?

What would be “backward traceability”?

Traceability matrix vs. graph-oriented traceability?
A traceability matrix is created by associating requirements with the work products that satisfy them. Tests are associated with the requirements on which they are based and the product tested to meet the requirement.

Sample Traceability Matrix

<table>
<thead>
<tr>
<th>ID</th>
<th>USER REQUIREMENTS</th>
<th>FORWARD TRACEABILITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>U2</td>
<td>Users shall process retirement claims</td>
<td>S10, S11, S12</td>
</tr>
<tr>
<td>U3</td>
<td>Users shall process survivor claims</td>
<td>S13</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>FUNCTIONAL REQUIREMENTS</th>
<th>BACKWARD TRACEABILITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>S10</td>
<td>The system shall accept requirement data.</td>
<td>U2</td>
</tr>
<tr>
<td>S11</td>
<td>The system shall calculate the amount of retirement.</td>
<td>U2</td>
</tr>
<tr>
<td>S12</td>
<td>The system shall calculate point-to-point travel time.</td>
<td>U2</td>
</tr>
<tr>
<td>S13</td>
<td>The system shall calculate the amount of survivor annuity</td>
<td>U3</td>
</tr>
</tbody>
</table>

Eliminate, rewrite, or correct traceability?
## Requirements Traceability Matrix: Variations


### Requirements Traceability Matrix

<table>
<thead>
<tr>
<th>User Requirements (List the title or number of the user functional requirement)</th>
<th>System Requirements (When applicable, list the title or number title of the corresponding system requirement)</th>
<th>Design Specification (When applicable, list any specifications which must be satisfied by the design)</th>
<th>Coding Component Reference (When applicable, code units, subroutines, or modules which implement the requirement)</th>
<th>Integration or System Test Case Reference (Include number of the test script for the requirement)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


©Lawrence Chung