Goal-Oriented Requirements Engineering (GORE)

- Why and What?
- Classical Logic Approach
- Traceability

Why Goal-Oriented Requirements Engineering?

Completeness: A Reference Model Perspec $S, K \vdash R$

[p. Zave and M. Jackson, Four Dark Corners of Requirements Engineering. *ACM Transactions on Software Engineering and Methodology* 6(1) 1-30. ACM Press. 1997.

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.

P, C ⊢ S

If the machine is as described in C, then execution of the program P will ensure the behaviour S at the machine's interface with the world.

(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.

The requirements are complete

if they are sufficient to establish the goal they are refining

[K. Yue, "What Does It Mean to Say that a Specification is Complete?', Proc. IWSSD-4, Fourth International Workshop on Software Specification and Design, Monterey, 1987.]

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]

Refresher

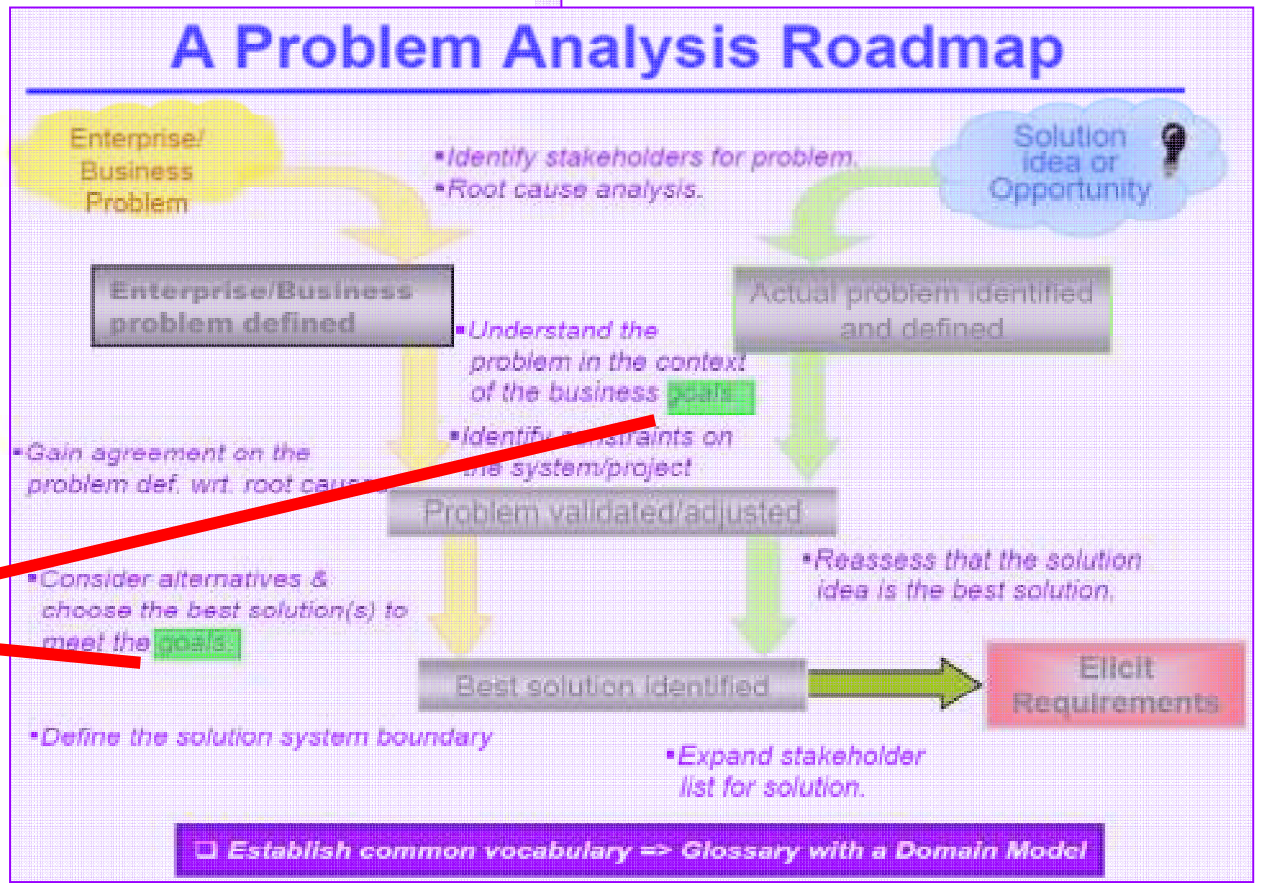
How to Elicit?
Goal-oriented Requirements Elicitation

"... Requirements Engineering is the branch of Systems engineering concerned with

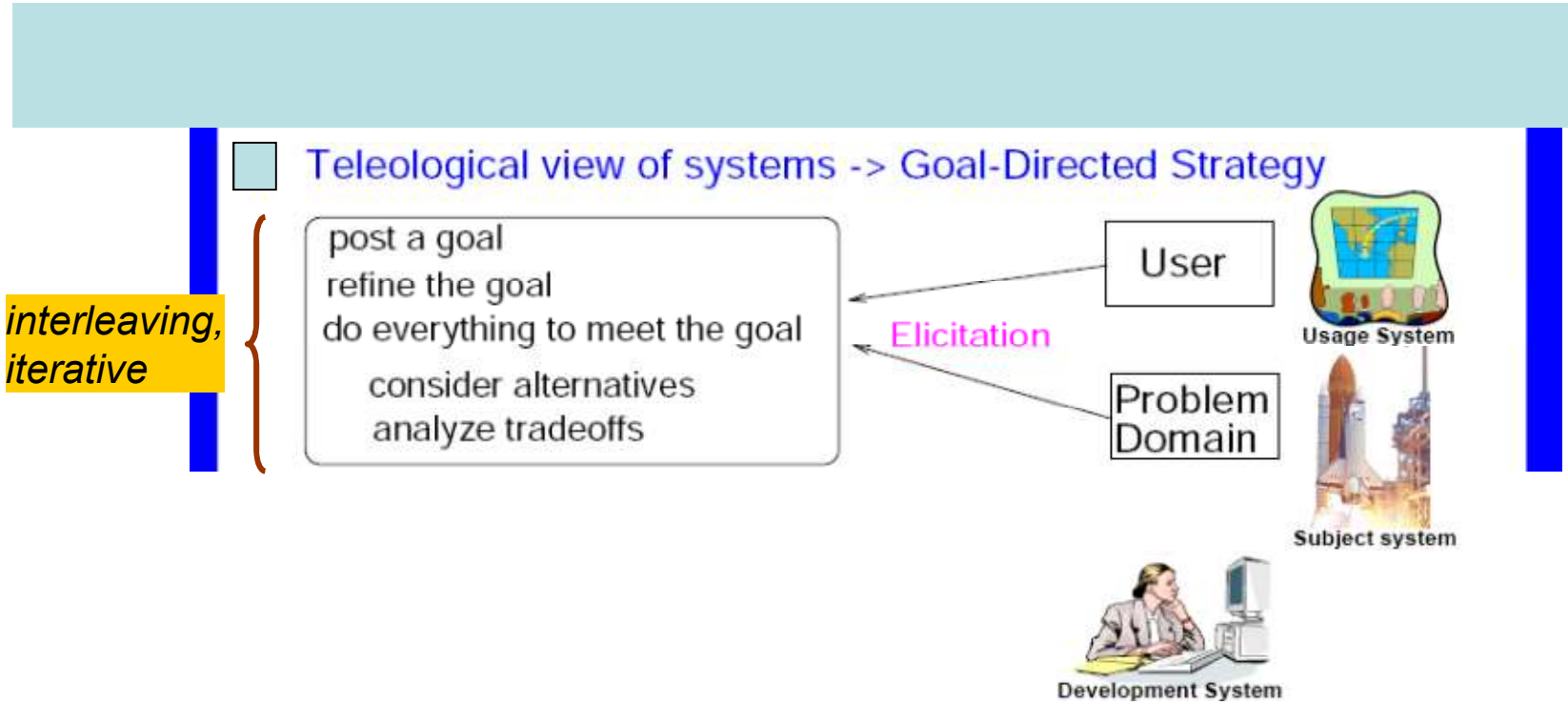
real-world goals for,
services provided by, and
constraints on
software systems

Requirements engineering is also
the relationships of these fa
to precise specification
and
to their evolution over ti

Where is Use Case Diagram?



What is Goal-Oriented Requirements Engineering?

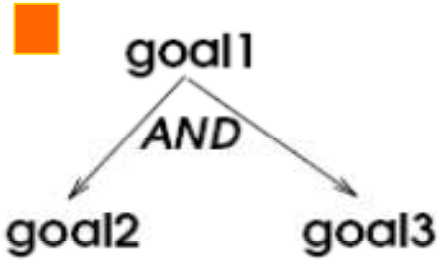


GORE: Classical Logic Approach

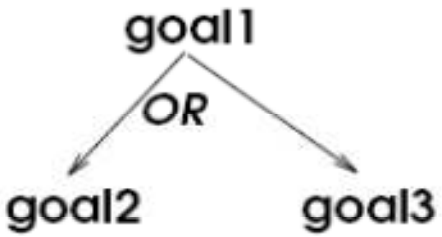
Goal-Directed Strategy -> Goal structure

Goals initially stated by the client are incrementally refined into subgoals

traditionally AND/OR decompositions



to satisfy the parent,
satisfy all its descendants



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"

[C. A. Hoare, Mathematical Models for Computing Science, Oxford Univ. Rpt., Aug. 1994]

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

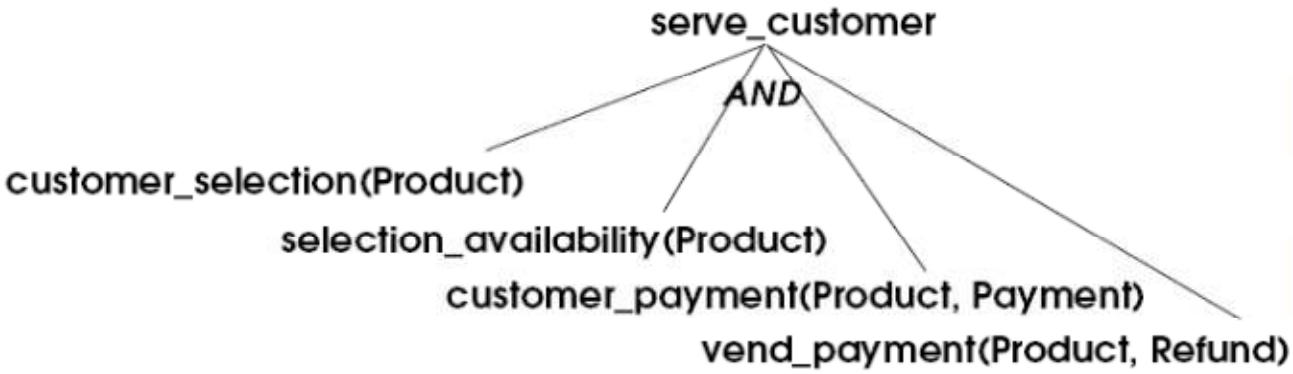
step 2: examine what is (really) needed to satisfy the goal.

- 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"



- ---> 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).
```

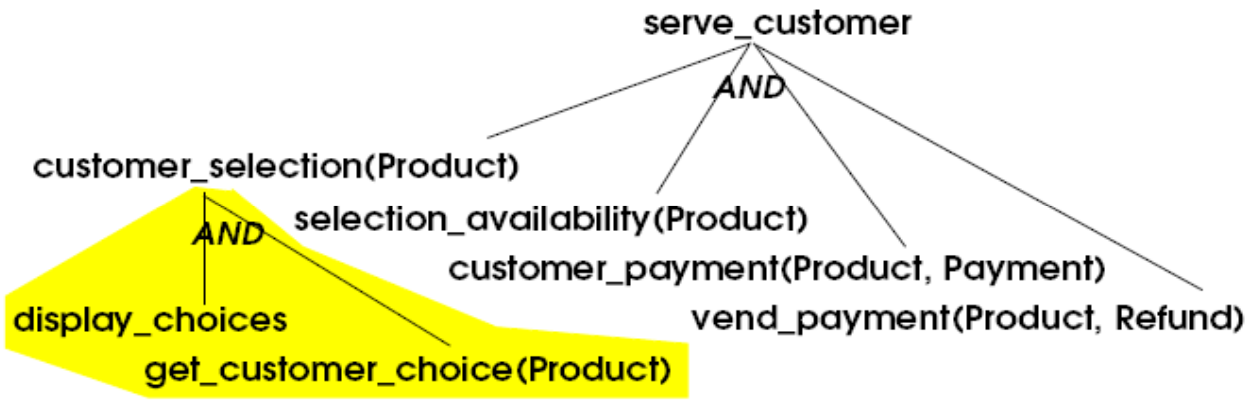
How to elicit?

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.



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

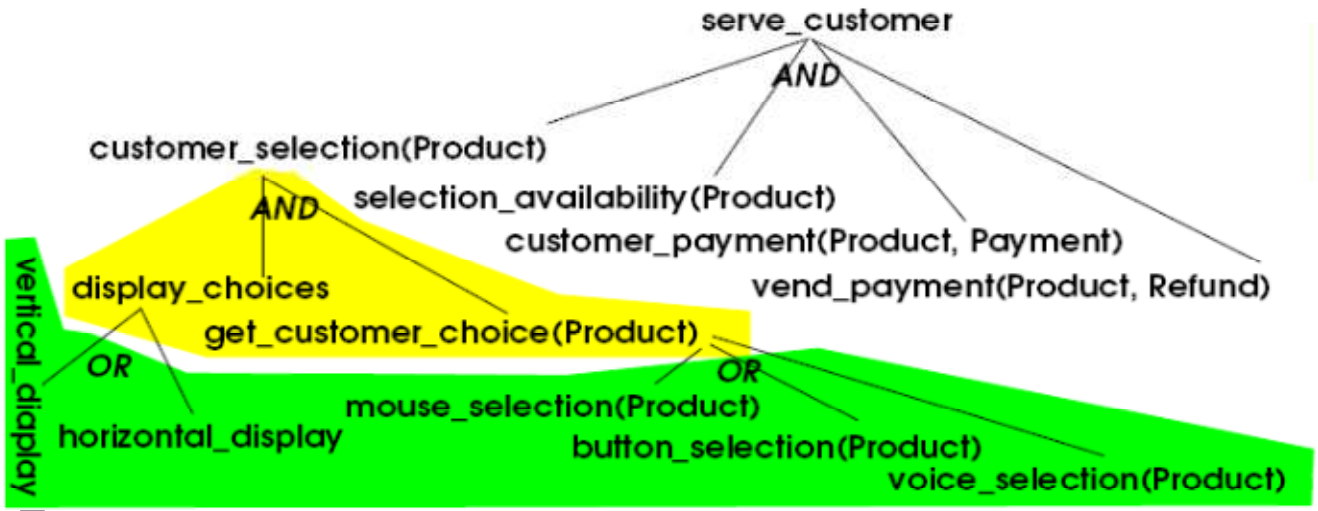
```
serve_customer :- customer_selection(Product),
                  selection_availability(Product),
                  customer_payment(Product, Payment),
                  vend_payment(Product, Refund).

customer_selection(Product) :- display_choices,
                               get_customer_choice(Product).
```

How to elicit?

Goal-Directed Strategy 1: Classical Logic Approach

F Example: "good old vending machine"

step 4: how-to elaboration



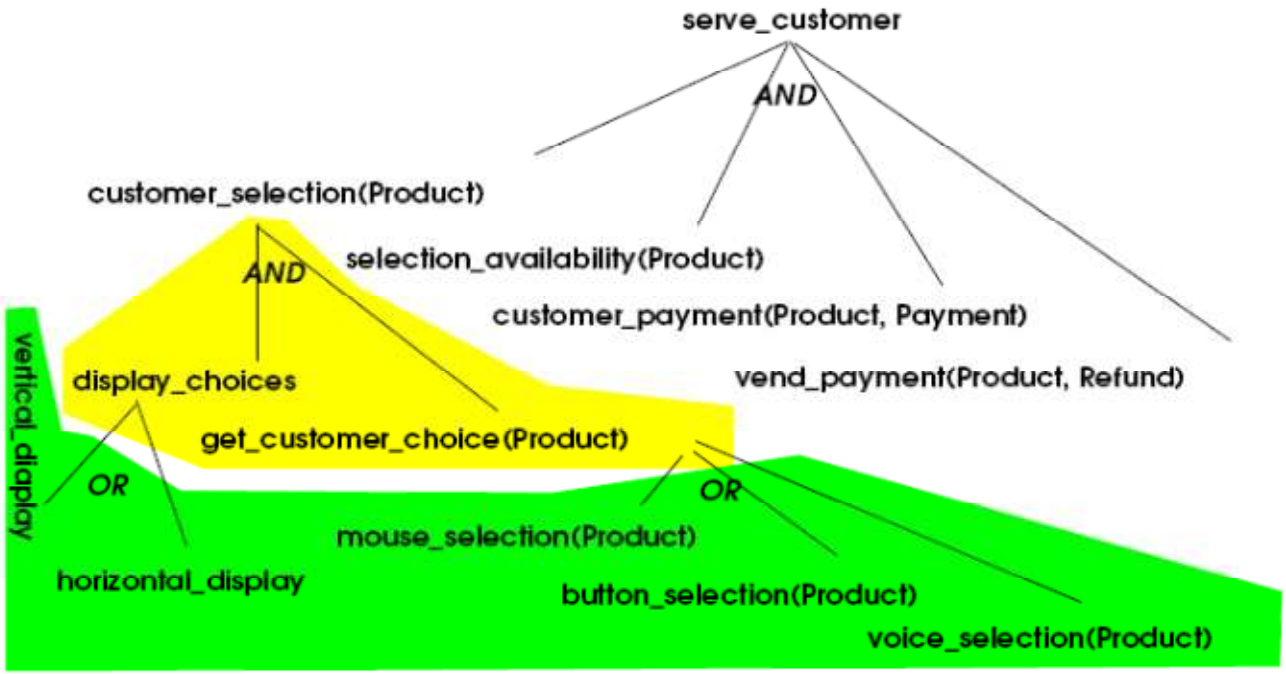
---> Executable specification (e.g., in Prolog)

```
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_diaplay .
display_choices :- horizontal_display .
```

How to elicit?

Goal-Directed Strategy 1: Classical Logic Approach

F Example: "good old vending machine"



- => 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

More on Traceability

=> Dependency => Traceability:

From design to requirements (avoid any erroneous, accidental design)

From requirements to design(ensure every requirement is met)

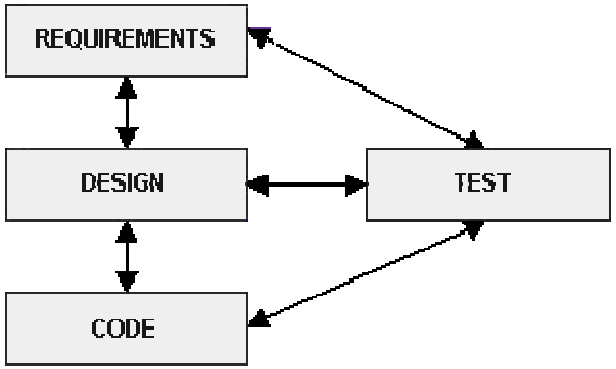
*What would be "**forward traceability**"?*

*What would be "**backward traceability**"?*

Traceability matrix vs. graph-oriented traceability?

Requirements Traceability Matrix

- 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.
  [http://www.jiludwig.com/Traceability_Matrix_Structure.html]



Sample Traceability Matrix

W, R, or S?

ID	USER REQUIREMENTS	FORWARD TRACEABILITY
U2	Users shall process retirement claims.	S10, S11, S12
U3	Users shall process survivor claims.	S13

ID	FUNCTIONAL REQUIREMENTS	BACKWARD TRACEABILITY
S10	The system shall accept requirement data.	U2
S11	The system shall calculate the amount of retirement.	U2
S12	The system shall calculate point-to-point travel time.	U2
S13	The system shall calculate the amount of survivor annuity	U3

Eliminate, rewrite, or correct traceability?

Requirements Traceability Matrix: Variations

http://departmentforms.dpsk12.org/dots/smedocs/requirements%20traceability%20matrix.pdf

Requirements Traceability Matrix

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

http://www.fns.usda.gov/wic/StateInformationSystems/FReD/AppendixAFRED-ERTMVersion1.0Summer2002.pdf