# Six (Im)possible Things Before Breakfast[1]
# Building Blocks for Wise Computing


Assaf Marron, Brit Arnon, Achiya Elyasaf, Michal Gordon, Guy Katz, Hadas Lapid, Rami Marelly,
Dana Sherman, Smadar Szekely, Gera Weiss, and David Harel

Weizmann Institute of Science, Stanford University, Holon Institute of Technology, Ben-Gurion University


## Wise Computing

**Computerize the software/system-engineering tasks that require knowledge and skills of humans**

*A software/system-engineering (SE) project review meeting in the era of wise computing: Examining prototypes of a medical patient-assistance robot (on desks) and associated software. A computer (for illustration is shown in a green suit, on the right) will be able to contribute questions and observations in ways commonly expected only from humans.*

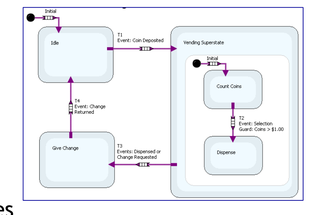
## Special Human Competencies

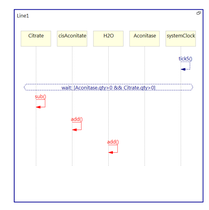
1. Notice irregularities, unexpected properties:
   - ❖ *"The arm movement is not smooth!"*
   - ❖ *"Hear this strange noise!"*
2. Detect missing requirements and undocumented assumptions:
   - ❖ *"Will it understand voice commands from a hoarse patient?"*
   - ❖ *"Can it process the patient's voice commands when the TV is on?"*
3. Ask and answer hard "what if" and "why" questions
   - ❖ *"Will a loud command from a TV show, like in a gym class, confuse the robot?"*
   - ❖ *"Why is it walking aimlessly around ? Is it looking for something?"*
4. Communicate concurrently with multiple levels of abstraction:
   - ❖ *"In <this> line of code we turn on the busy bit; this will make the system status light blink; the user will thus know that he/she should wait before stating the next voice command"*
     (from programming detail → system capability → user need)
5. Free Association:
   *"I have heard that someone remotely hacked a pacemaker. Can this happen here?"*
6. Creativity: Thinking outside the box
   - ❖ *"I bet it will be confused if I ask it to fetch a bottle that is glued to the table"*

*<Etc. >*

## CF: A Common Formalism for all Knowledge: Project, SE, Domain

- A single formalism for
  - All SE artifacts
  - SE Knowledge
  - Domain knowledge
  - Externalizing internal information from SE tools
- With extensive annotations and relationships
  - Enables applying general knowledge to specific cases
- Extensive ability for meta-referencing
- Based on, and substantially enhancing, Statecharts and Scenario-Based Programming (LSC)




## AE: An Analysis Engine that Mimics Unique Human Skills

- **Ongoing analysis of all project artifacts (in CF)**
  - Simulations and testing
  - Verification
  - Detection of unexpected properties
  - Monitoring
  - Automated Repair
  - Application of external knowledge
  - Source code and documentation analysis
  - . . .

- **Computational tools used (by AE):**
  - Model checking
  - Program synthesis
  - Specification mining
  - Machine learning
  - Constraint solving (SAT/SMT
  - AI algorithms
  - Scenario-based heuristics
  - Natural language processing
  - . . .

## IL & IE: An Interaction Facility for Communicating on all SE Matters

### The Interaction Language (IL)

- Captures everything engineers may state about a system
  - Steps in an algorithm
  - Requirements/User needs
  - Development tasks
  - System components, configuration
  - Bug descriptions
- A single interface to all wise computing tools
- Based on text statements
- Controlled English
- Fully formalized

Program Scenario S1:
"When an autonomous vehicle reaches a red traffic light, it stops"

Customer-Request Ticket T2:
"When an ambulance reaches a red traffic light, it turns to green"

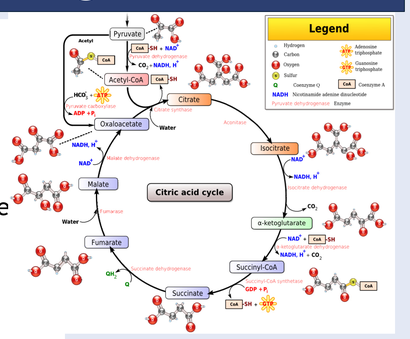
Disambiguation/Confirmation:
"Please confirm that 'it' refers to the vehicle (and not the light, the ambulance, the running scenario, or anything else)"

### The Interaction Engine (IE)

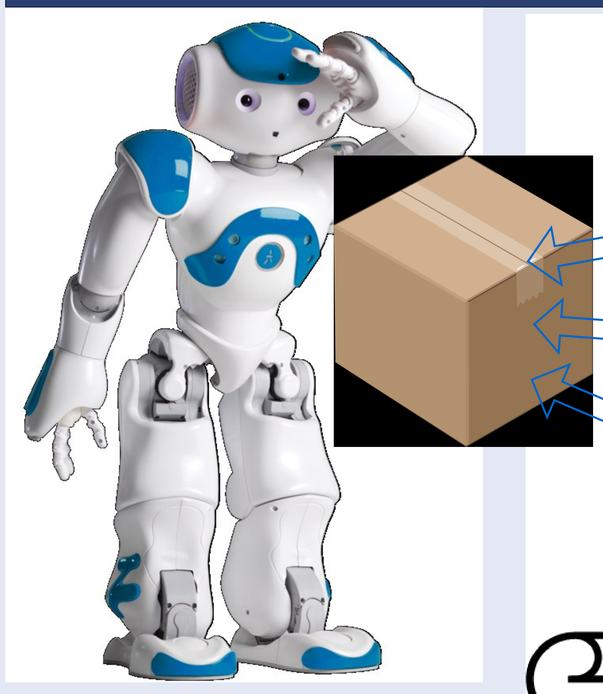
- Controls all interaction with engineers
  - In offline analyses and interactive sessions
- Clarifies exact meaning of user inputs
  - Disambiguation, context, confirmation
- Enables all tools to produce consistent information with regard to all development stages, tasks and artifacts

## DP: Design Patterns that Facilitate Wise Computing

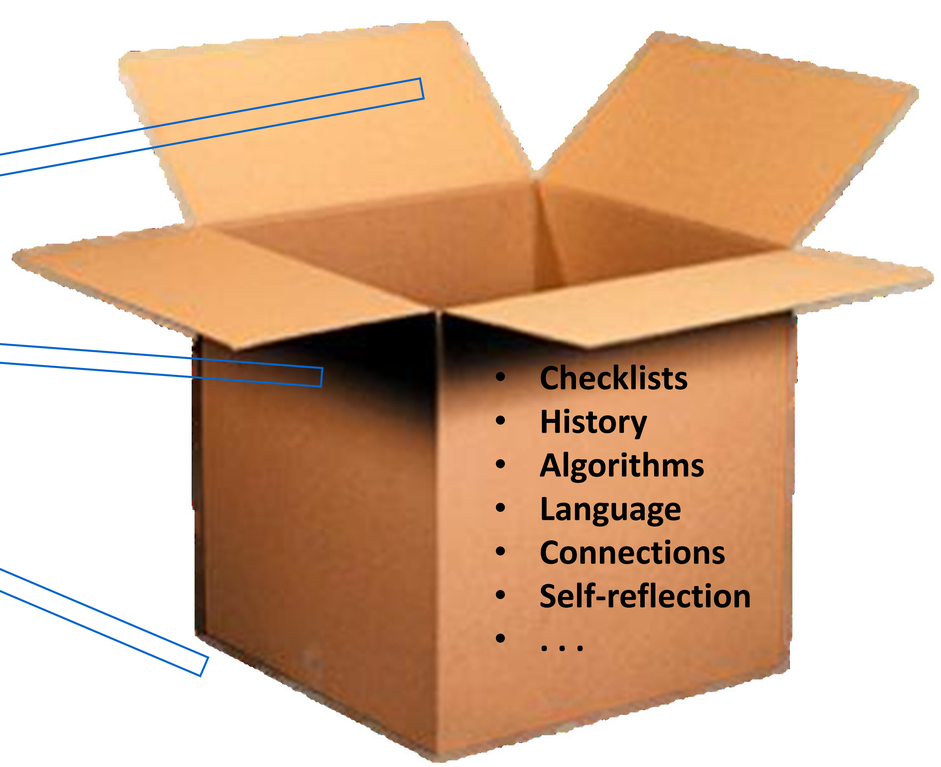
- Specify knowledge as scenarios that try to apply it
- Components self-integrate into a target system without requiring that the system be changed to accommodate them
  - Actively seek out/collect component's own input
  - Components make outputs and constraints available to all
  - New components comply with constraints imposed by existing one
  - Example: A model of Citric Acid (Krebs) Cycle in the cell, where chemical reactions are modeled separately




## If you want t



137. Remember to check c
     of type T1 for proper
138. Object O1 is also kno
     as Z1, Z2
139. When verification V1
     takes too long, abstra
     D1,D2 into D.
140. If there is no way to d
     within the allotted tir
     ask the human if this
     required
141. The solutions S1, S2 h
     in the past for similar
142. Before reporting a bu
     similar one was alrea
143. . . .



- **Checklists**
- **History**
- **Algorithms**
- **Language**
- **Connections**
- **Self-reflection**
- **. . .**

[1] With apologies to Lewis Carroll