How To Program If You Cannot

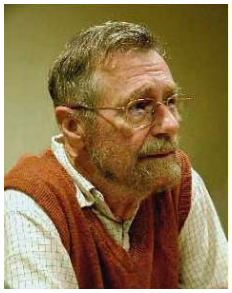
–

Software Engineering and Programming Methodology

Michael Jackson
The Open University
jacksonma@acm.org

WG2.3 Meeting
York
4th February 2019

Dijkstra (1989) on software engineering



"... If you carefully read its literature and analyze what its devotees actually do, you will discover that *software engineering* has accepted as its charter, *'how to program if you cannot'*."

E W Dijkstra;
On the Cruelty of Really Teaching Computing Science;
CACM 32, 12 pp1397-1414, December 1989

* Why software engineering is programming
  * So what does 'you cannot' mean?
  * So how can you program if you cannot?

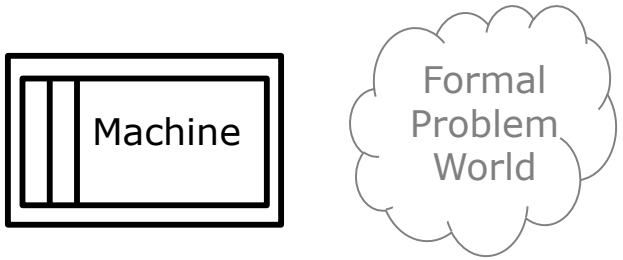
What is software engineering ?

Two Aspects

1. Engineering OF software

   GCD, factorise integer, JVM,
   solve PDEs, 3D convex hull, ...

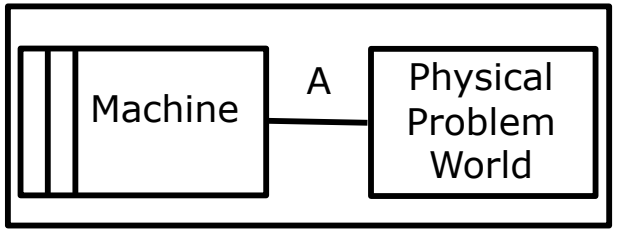
   Formal Problem World
   Unipartite System



2. Engineering BY software

   passenger lift, chemical plant,
   avionics, radiotherapy machine, ...

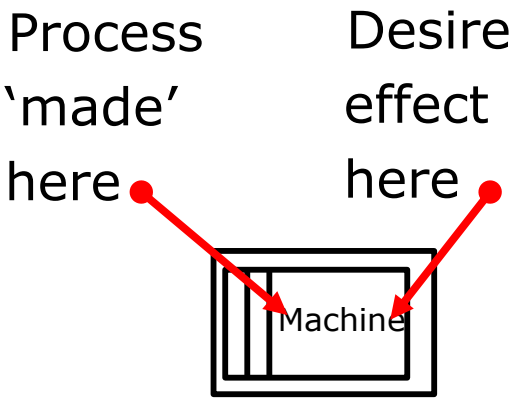
   Physical Problem World
   Bipartite System



Dijkstra (1968) on programming

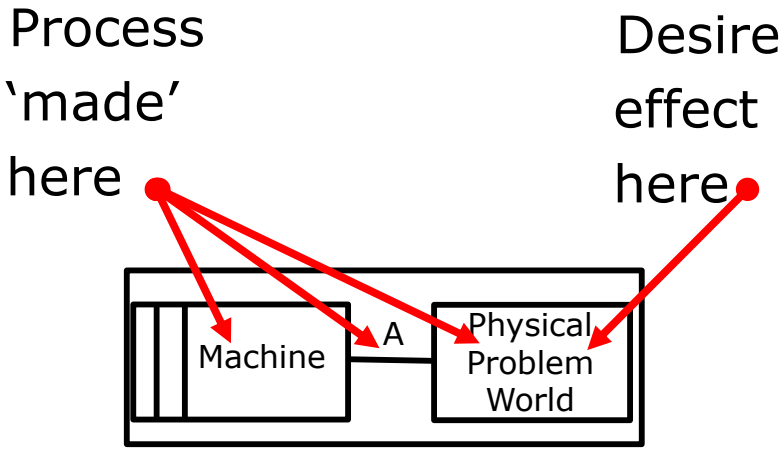
"... the process taking place under control of his program is the [programmer's] true subject matter ... it has to effectuate the desired effect ... the 'making' of the corresponding process is delegated to the machine."

E W Dijkstra; "GO TO letter" March 1968



Program a unipartite system

☛ computational result

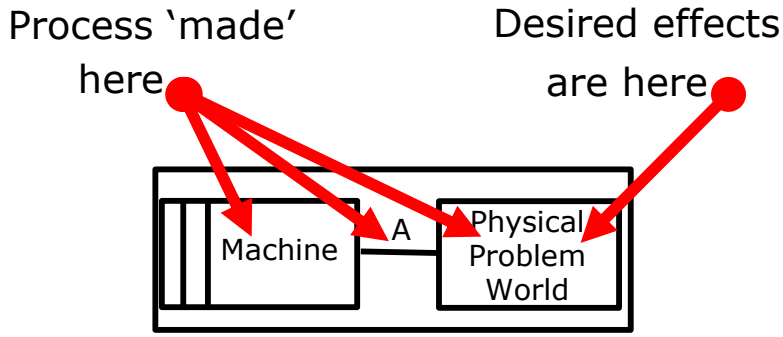


Program a cyber-physical system

☛ bipartite physical behaviour

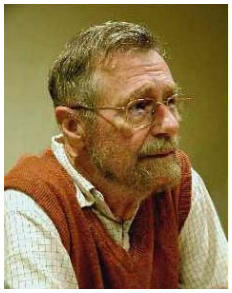
What is a bipartite behaviour?

* A behaviour is a purposeful bipartite program in which ...
  ... the machine governs the physical world behaviour
* The machine constrains the behaviour by its program ...
  ... the world constrains the behaviour by its given properties



* The basic mechanism of constraint is physical causality
  * Failure of a behaviour is failure of a causal link

Dijkstra (1989) on 'you cannot'



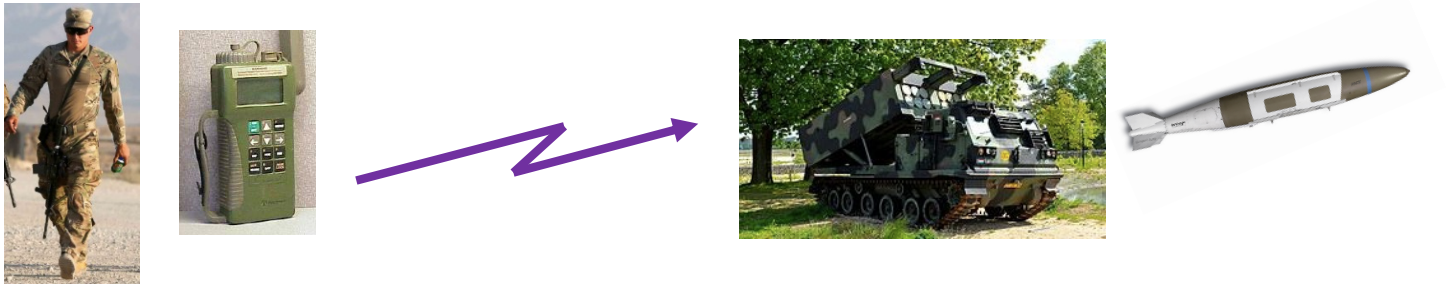
"... the programmer's task is not just to write down a program ... main task is ... formal proof ... program meets its equally formal functional specification."

E W Dijkstra;
On the Cruelty of Really Teaching Computing Science;
CACM 32, 12 pp1397-1414, December 1989

* 'You cannot' program a unipartite system if you cannot ...
  * Work with formal computer science (logics, discrete maths)
  * Master the abstract problem world (eg number theory)
* 'You cannot' program a bipartite system if you cannot also ...
  * Develop programs as structures of bipartite behaviours
  * Make adequate formal models of non-formal physical worlds

Bipartite behaviour in a small system

A small example: handheld GPS missile controller



Soldier turns on plugger; sets missile target; sees red light (battery warning); replaces battery; sends firing command.

Some Questions:

1. What is the behaviour structure?
2. How has it been designed?
3. What could go wrong?

Multiple constituent behaviours in a larger system

A larger example: contemporary car

Engine management; Anti-lock braking; Stop-start; Air conditioning; Automatic parking; Lane departure warning; Restrict speed; Cruise control; Rear-view camera; Active suspension; Automatic emergency brake; Driver settings; Door locking; Charcoal filter cleaning; Ignition cycle ...

* Concurrent multiple enactments of constituent behaviours
  * Behaviour enactments form a dynamic enactment tree

Some Questions:
1. Is each of these a constituent behaviour?
2. Time span of each behaviour?
3. Temporal relationships among behaviours?
4. Cruise Control and Restrict Speed?

Adequate formal models of non-formal physical worlds

In a formal model	In the physical world
Timeless reasoning	Everything is in flux
Basic axioms	Postulated causality
Atoms	None at CPS scales
Truth by construction	Truth by observation
Perfect precision	Finite representation
Discrete values	Everything is continuous
Disjoint assemblages	Everything is connected
Frame conditions	Ubiquitous side-effects
Compositionality	Invasive combination
Proofs	Contingent reasoning
. . .	. . .

Adequate formal models of non-formal physical worlds

In a formal model	In the physical world
Timeless reasoning	Everything is in flux
Basic axioms	Postulated causality
Atoms	None at CPS scales
Truth by construction	Truth by observation
Perfect precision	Finite representation
Discrete values	Everything is continuous
Disjoint assemblages	Everything is connected
Frame conditions	Ubiquitous side-effects
Compositionality	Invasive combination
Proofs	Contingent reasoning
. . .	. . .

Adequate?

for toy systems?

. . .

. . .

. . .

for critical systems?

Behavioural requirements in non-formal physical worlds

* Infer driver intent from throttle movement
* Release door locks on frontal collision impact
* Stop wipers if driver's door opens
* Do not allow two trains in same section
* Do not allow unauthorised entry to any room
* Never open lift car doors except at a floor
* Stop lift car travel only at a floor rest position
* Do not leave a floor if doors are open
* Do not leave a floor if car is overloaded
* Infer airspeed from multiple Pitot tube outputs
* Use and save prescribed or last validated setting

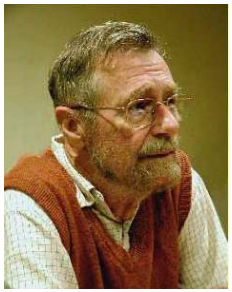
Behavioural requirements in non-formal physical worlds

* Infer driver intent from throttle movement	* A/c may demand torque
* Release door locks on frontal collision impact	* This is a thieves' charter
* Stop wipers if driver's door opens	* Lock malfunction at 70mph
* Do not allow two trains in same section	* Trains must be assembled
* Do not allow unauthorised entry to any room	* Against FD regulations
* Never open lift car doors except at a floor	* Motor fails very near a floor
* Stop lift car travel only at a floor rest position	* Stop ASAP if cable breaks
* Do not leave a floor if doors are open	* Firefighters may need it
* Do not leave a floor if car is overloaded	* Firefighters may need it
* Infer airspeed from multiple Pitot tube outputs	* Ice in AF447 Pitot tubes
* Use and save prescribed or last validated setting	* Saved position may creep

Behaviour benefits for complexity and modelling reality

* In developing constituent behaviours
  * Models are specific to each behaviour and its context
  * Monitoring a behaviour's context is itself a behaviour
  * Causal links exhibit the operational principle
* In combining constituent behaviours
  * Models identify shared variable interference
* By traversing operational principle
  * Causal links identify potential and actual failures
* By supporting 'correctness' structures
  * Fault-tolerance, recovery blocks, requirements monitors, ...

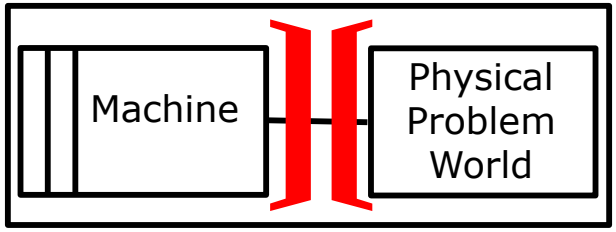
Envoi: Dijkstra's firewall



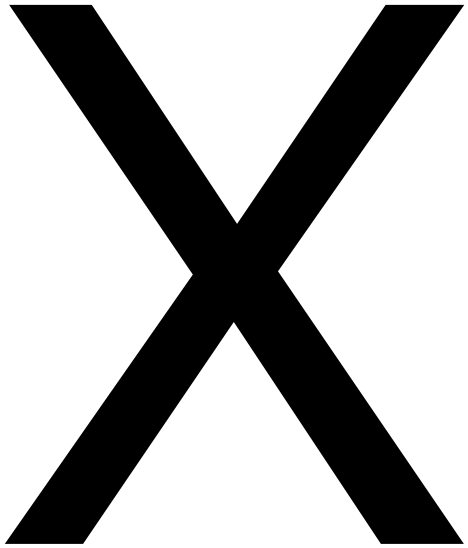
"... functional specifications ... act as a logical 'firewall' between ... the 'pleasantness problem,' ... whether an engine meeting the specification is the engine we would like to have... [and] the 'correctness problem' ... how to design an engine meeting the specification."

E W Dijkstra;
On the Cruelty of Really Teaching Computing Science;
CACM 32, 12 pp1397-1414, December 1989

* Room for discussion?
  * Can we–should we–separate developing the machine from developing physical behaviours?



Thank you



Envoi: Dijkstra's firewall?

* A firewall cannot work well in machine and world dialogue
  * Neither side of a dialogue makes sense in isolation
  * Machine variables are essential to behaviour enactment
  * Text pointers in machine are essential to dialogue progress
  * World lacks means to control constituent behaviour