Formalism and Intuition
in
Software Development

"... intuition and deduction, on which alone we rely in the acquisition of knowledge."

René Descartes

Michael Jackson
The Open University
jacksonma@acm.org

FME Meeting
CWI Amsterdam
March 8 2013

What do I mean
by 'formalism'
and
'intuition'?

Formalism is using calculi of symbols

- Reasoning over symbol structures according to the rules of a strict calculus, to deduce certain conclusions from axioms

"... decisive step of mathematical abstraction... forget what the symbols stand for ... many operations to carry out without having to look at the things they stand for."

Hermann Weyl, 1940

Intuition is comprehension of the world before us

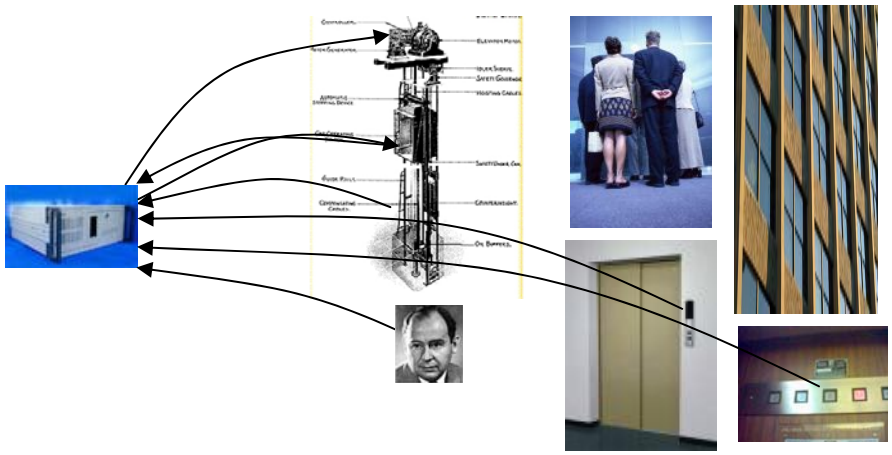
- Comprehending worldly truths by direct observation and past experience, without appeal to conscious reasoning

"Intuition is the way we translate our experiences into judgments and decisions ... using patterns to recognize what's going on in a situation."

Gary Klein; Intuition at Work, Doubleday, 2003

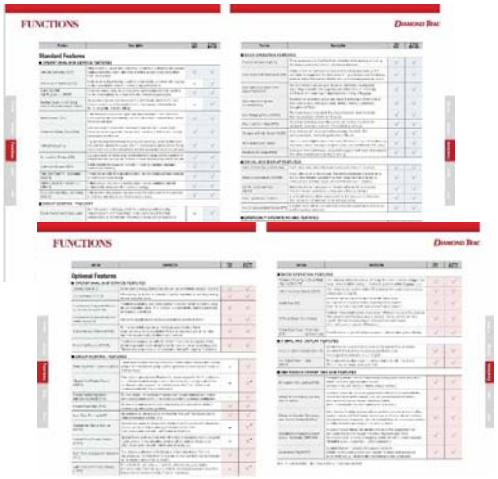
What do I mean by 'software development'? Here's an example

Lift-control system: a development problem



System behaviour must satisfy all of the stakeholders' requirements!

- A normal design task
  - Mitsubishi, Thyssen, Otis, Schindler, Fuji, Kone, …

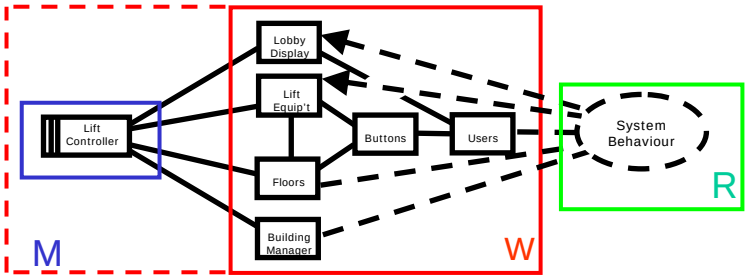


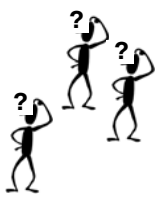
Diamond Trac Elevator (Mitsubishi): >50 Computer-Controlled Features

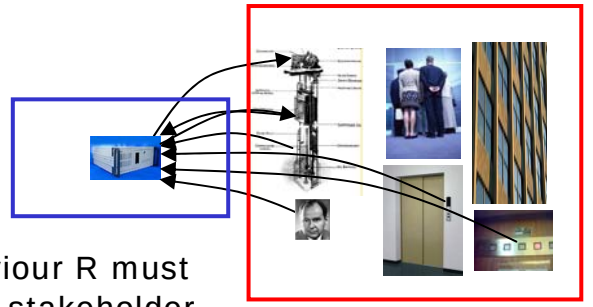
- But I'll consider it as a radical design task

What are the development activities inviting possible use of formalism and/or intuition?

Lift-control system: development activities







(Behaviour R must satisfy stakeholder requirements)

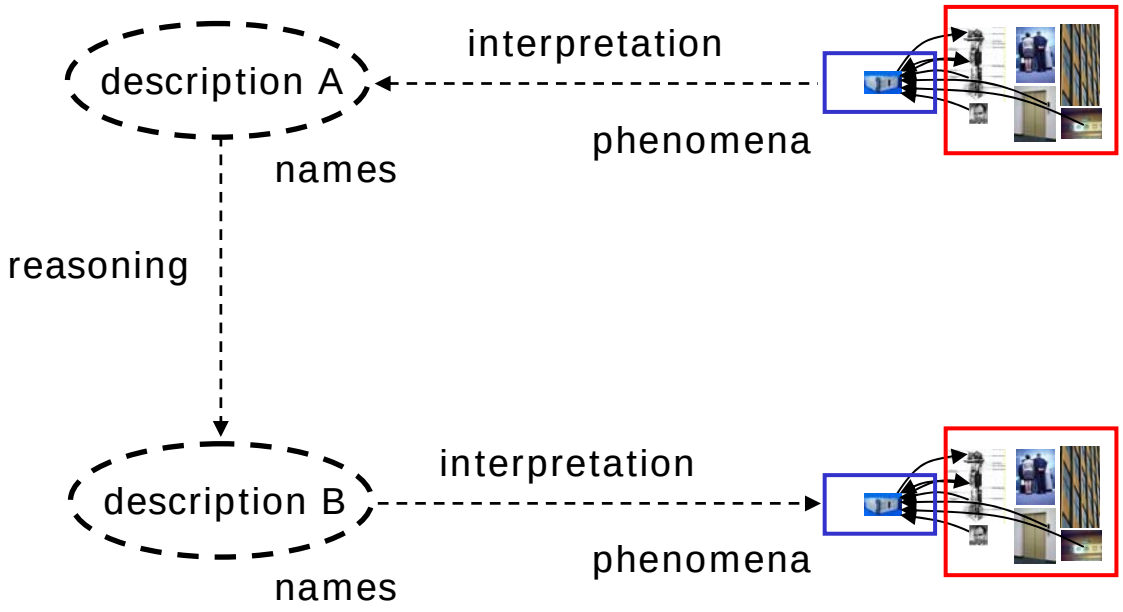
In some order / interleaving ...

- Identify stakeholders, discover and describe requirements
- Discover, describe and analyse problem world properties W
- Design a satisfying system behaviour R
- Ensure R satisfies requirements (and stakeholders agree)
- Invent and specify a feasible machine behaviour M
- Formally demonstrate that M,W |= R
- Convince the regulator that system is safe
- ...

How do formalism and intuition apply to these activities?

Formalism and intuition have a common pattern, but differ in their basic characteristics

The common pattern of describing and reasoning



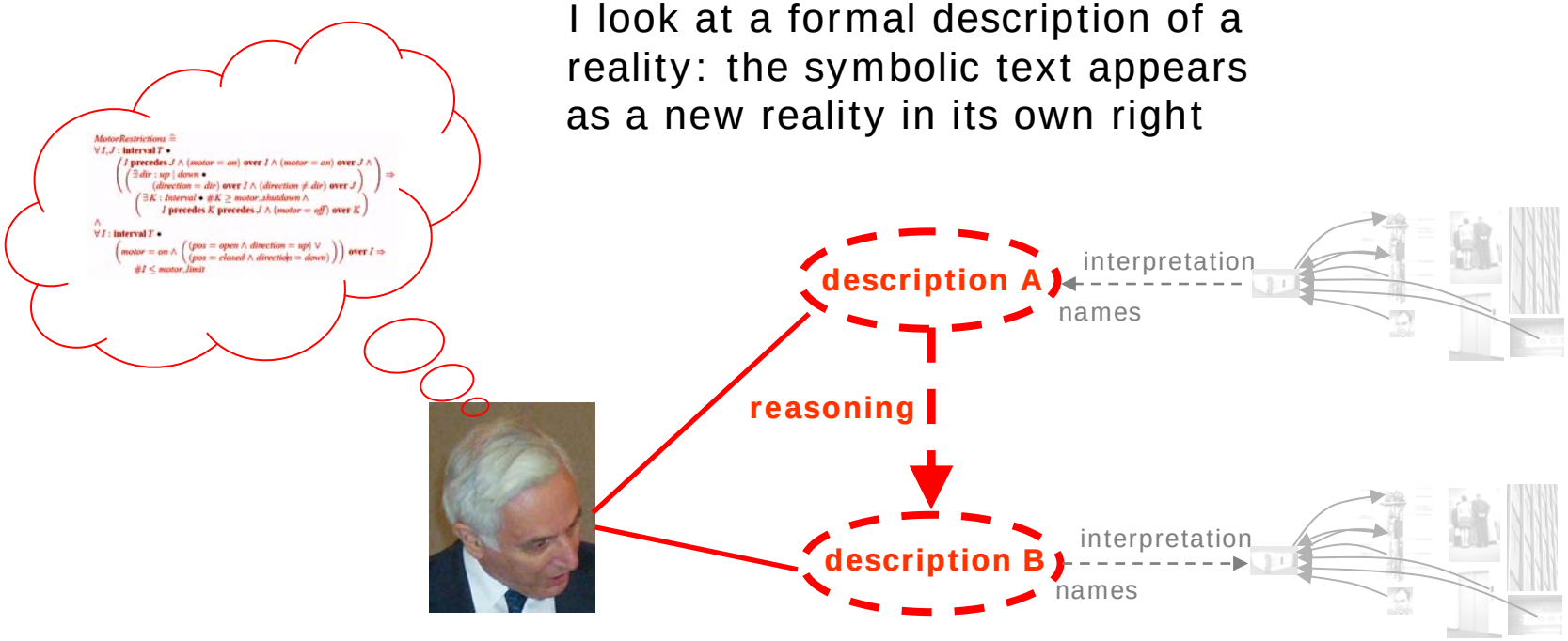
- Describe, reason, conclude, interpret
  - Phenomena
    - Reality
    - Names
  - Language
    - Types
    - Connectives
    - Logic
  - Inference
    - Precise
    - Vague

Formalism emphasises correct reasoning

"It must be possible to replace in all geometric statements the words point, line, plane, by table, chair, mug."

David Hilbert, quoted by Hermann Weyl, 1944

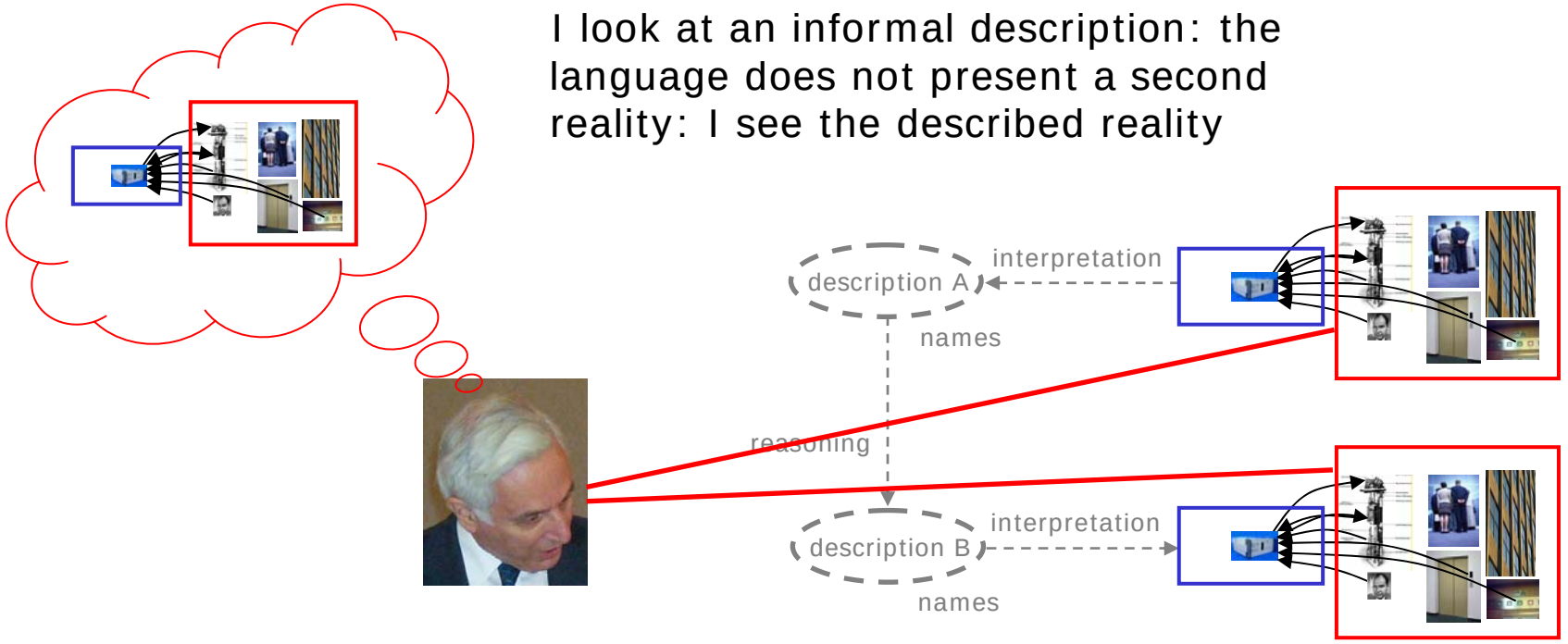
I look at a formal description of a reality: the symbolic text appears as a new reality in its own right



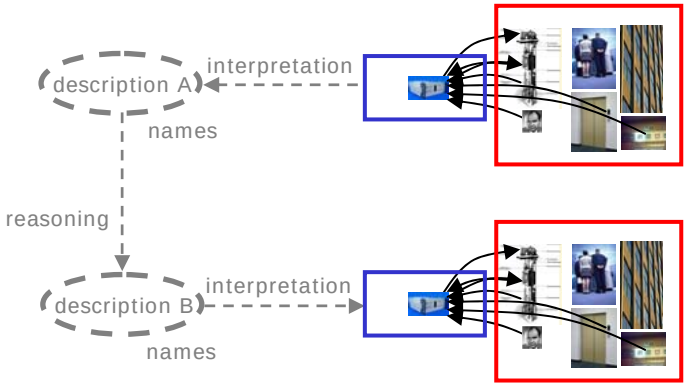
Intuition emphasises the subject reality

"Uttering a word is like striking a note on the keyboard of the imagination."

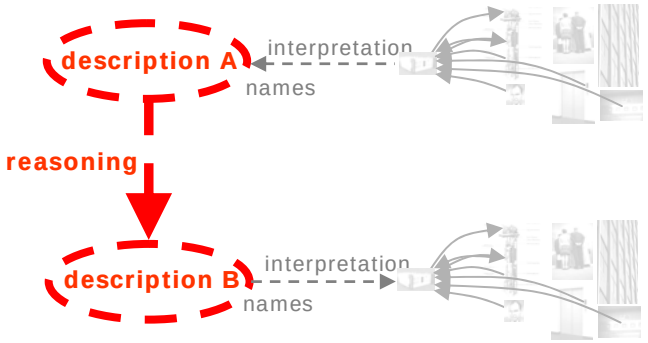
Ludwig Wittgenstein; Philosophical Investigations, I.6



How formalism and intuition differ



- Intuition allows
  - Flexible language definition
  - Phenomena of any types
  - Flexible description syntax
  - Extensible reasoning logic
  - Partial inconsistent models
- Intuition presents understanding
  - Reasoning is unreliable
- Conclusion is unreliable in reality!



- Formalism demands
  - Exact language definition
  - Phenomena ≈ language types
  - Predefined description syntax
  - Predefined reasoning logic
  - One fully consistent model
- Formalism presents theorems
  - Description A => description B
- Conclusion is unreliable in reality!

Formalism is for
proving theorems,
intuition is for
learning, exploring,
approximating
and inventing

Formalism and intuition: what they are for

"Thus logic and intuition have each their necessary rôle. Each is indispensable. Logic, which alone can give certainty, is the instrument of demonstration; intuition is the instrument of invention."

Henri Poincaré, 1908

- A practical description combines some of both
  - eg: non-formal main structure of a formal text
  - eg: local use of FSM, FOPL, ... in non-formal text
- So: how can we use formalism to best advantage?
  - Where do its disadvantages hurt most?
  - How can we benefit from its certainties?

A paradox?
Formalism can't handle the real difficulties of software complexity

Complexity of requirements

- Train control system
  - “Never 2 trains in one segment”
    - Assembling a train? Crash rescue?

- Lift control system
  - “Door_Open => Car_At_Floor”
    - Firefighter mode? Maintenance mode?
  - “Car_Stopped => Car_At_Floor”
    - Free-fall prevention on broken cable?

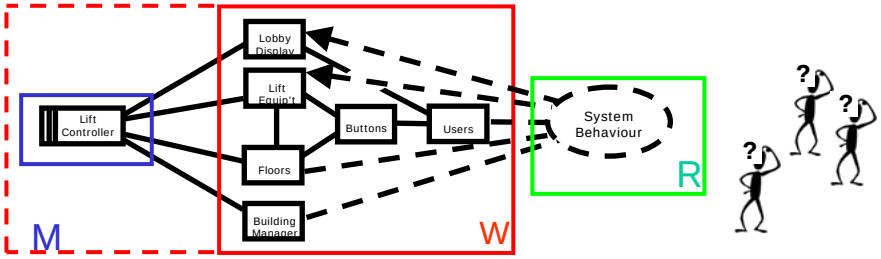
- Access control system
  - “In_Room (p,r)” => “Authorised(p,r)”
    - Forbidden by fire regulations!

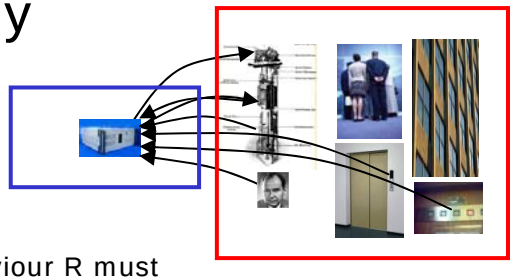
Complexity of the problem world

- Fuzzy phenomena
  - “Lamp_Lit”
    - Dim bulb? Bulb failed? Dirty glass?
  - “Lift_At_Floor[f]”
    - Within 1cm of floor f home position?
    - Lift car stationary at floor f?
    - What if sensor actuator is broken?
- Dubious physical properties
  - “Switch-On => Lamp_Lit”
    - Switch failure? No power?
    - Circuit interrupted?
  - “Motor_On => Lift_Rising”
    - Motor burnout? Relay failure?
    - Gearbox failure? Hoist cable broken?

Formalism is used most effectively when the context of application is restricted

Context: source and solution of complexity





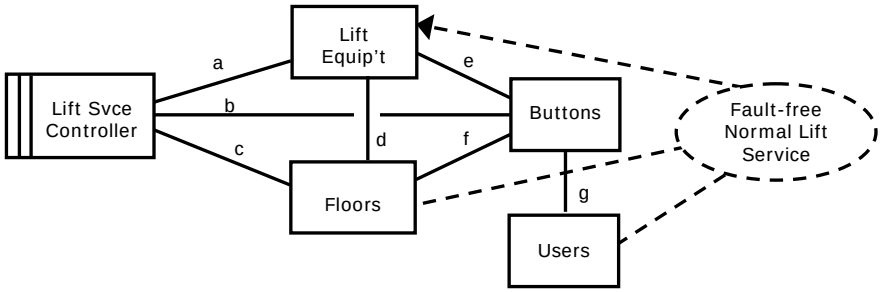
(Behaviour R must satisfy stakeholder requirements)

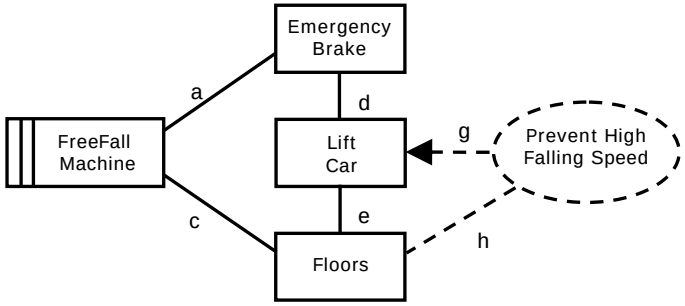
- Many operational contexts
  - Service demand patterns: daily, weekly, occasional, ...
  - Equipment maintenance, licensing inspection, ...
  - Fire in building, earthquake, flood, ...
  - Many possible equipment faults of varying severity
    - Failure of one floor sensor, doors won't open / close, failure of hoist motor, hoist cable broken, ...
- In each ensemble of concurrent contexts ...
  - Certain problem world assumptions hold well enough
  - Certain associated requirements are to be satisfied

We must identify,
structure, and
understand
the many contexts
informally

Some candidate contexts

- Fault-free normal lift service
- Firefighter lift service
- Test mode behaviour
- Maintenance mode
- Failure-reduced service
- Maintain lobby display
- Door open and close
- Overloaded travel prevention (?)
- Fault detection and diagnosis
- Free fall prevention
- Editing priority scheme
- Managing priority schemes
- Safe lift parking
- Tycoon floor security (?)
- … ??





Behaviours in contexts

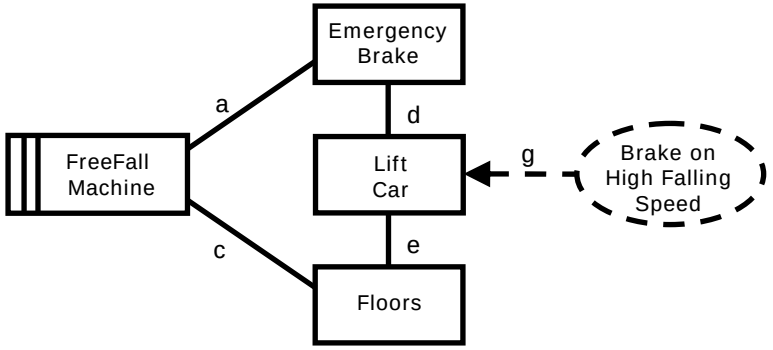
are inventions

(contrivances)

Here's an example

An example context: free fall prevention

- An identified context is invented
  - A requirement to be satisfied
  - Properties holding well enough
- Invention begins with an idea
  - Unforeseen details come later



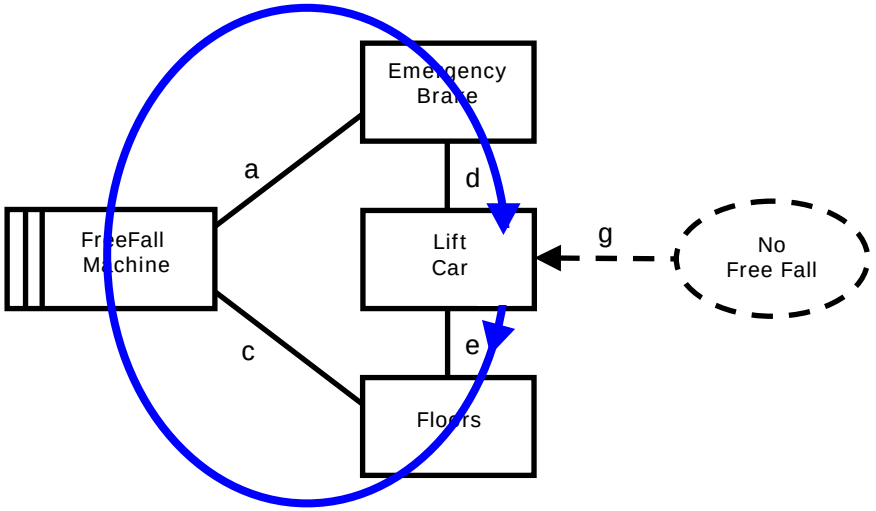
- Invention can't be done by formalism!

"The purpose of abstraction is not to be vague, but to create a new semantic level in which one can be absolutely precise."

Edsger Dijkstra, Turing Award Lecture, 1972

- Invention in the material world depends on imprecision
  - Unforeseen details compel adjustment to any abstraction

The role of formalism: to check the product of intuition



a: FFM! Apply_Brake
c: Flrs! Sensor_On[f]
d: EBr! Lock_Car_In_Shaft
e: LCar! Arrive_Floor[f]
g: Lift_Car_Falling_Speed

- Operational principle explains how it works
  - Traversal: g → e → c → a → g
- Not physics or mathematics, but 'logic of contrivance'
  - "... parts combining to an overall operation which achieves the purpose of the machine" (Polanyi)
- Formal mathematical and physical analysis
  - Follows and evaluates causal links in traversal

Two thoughts about software development method

Development method — 1

"Method consists entirely in properly ordering and arranging the things to which we should pay attention."

Rene Descartes; Works Volume X, p379:
Rules for the Direction of the Mind, Rule V.

- Development must rest on rich understanding ...
  - ... obtained by informal exploring, discovering, learning ...
  - ... unconstrained by premature abstractions
- Any one formalism inherently compels many abstractions
  - eg abstracting from agency and control
  - eg abstracting from 'machine vs problem world' distinction
  - eg abstracting from 'given vs required' distinction
  - eg abstracting from non-truth-functional phenomena
  - eg abstracting from the logic of contrivance (Polanyi)
- So: formalise late, formalise locally, formalise variously!

Development method — 2

Formalism's neglected roles

- Formalism too often aims at a unified consistent model
  - Eventually the system must be (in some sense) unified …
  - … but only once the local contexts are understood
- Tasks for formalism
  - Formalising and verifying in each local context
  - Analysing mutual consistency of local contexts
  - Exploiting 'logic of contrivance' for safety cases
  - Reconciling heterogeneous local contexts
  - Transforming reconciled contexts to software view
  - …

Envoi:
one last word

Envoi

"A formal method of system development has the same virtues and limitations as an arithmetic method of designing a house"

Anonymous

Thank you!