Testing the Machine

In the World

Michael Jackson
jacksonma@acm.org

Verification Conference
Haifa
25 October 2006

Introduction

- Functional correctness and dependability
  - (also performance, security, ...)
  - For software-intensive systems
- Where is the function located?
  - The unavoidable problem world
- An approach to functional structure
  - Problems and subproblems
  - Normal engineering design
  - Components and composition
  - Subproblem and composition concerns
- Conjecture: relevance to testing?
  - Testing structure ≈ functional structure?
  - Testing for normal concerns?

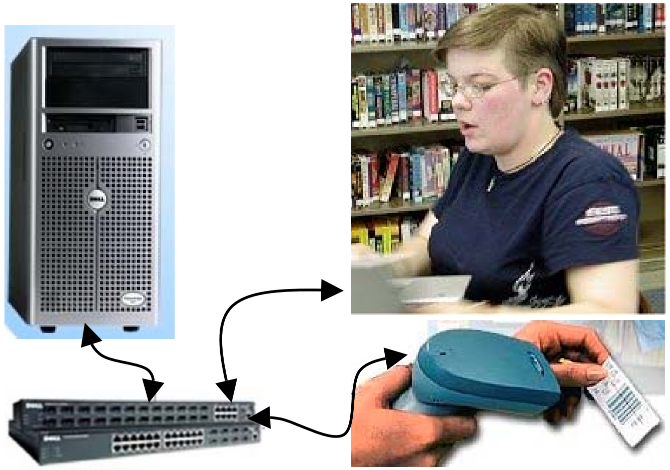
Software as an engineering product

> “… engineering [is] design and construction of any artifice which transforms the physical world around us to meet some recognised need.”

G F C Rogers;
The Nature of Engineering: A philosophy of technology;
The Macmillan Press, 1983, p51

- Much software development is engineering
    - Administrative systems
    - Embedded systems
    - Enterprise systems
    - Communication systems
    - …

Software for a lending library

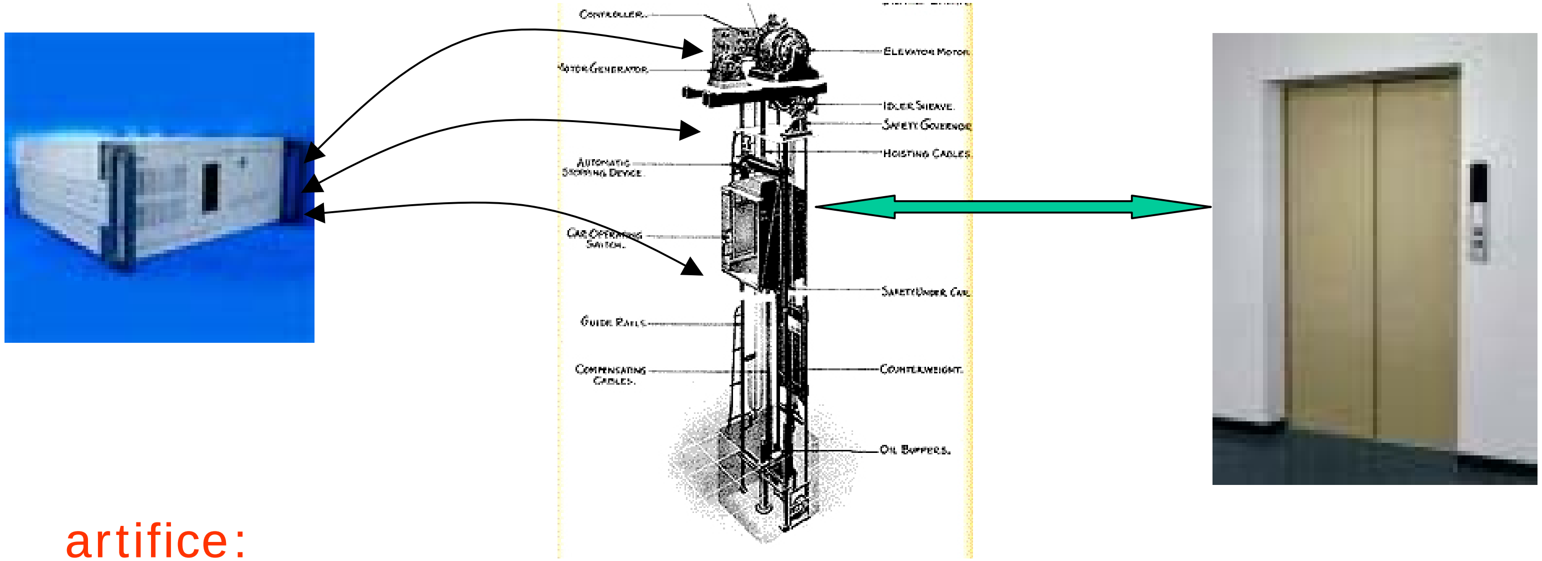


artifice:
the software
& the
computer

physical world:
books, swipe cards,
shelves, members,
catalogue, staff &c

recognised need:
only members can
borrow, reserved books
are not lent, catalogue
is correct &c

Software to control a lift

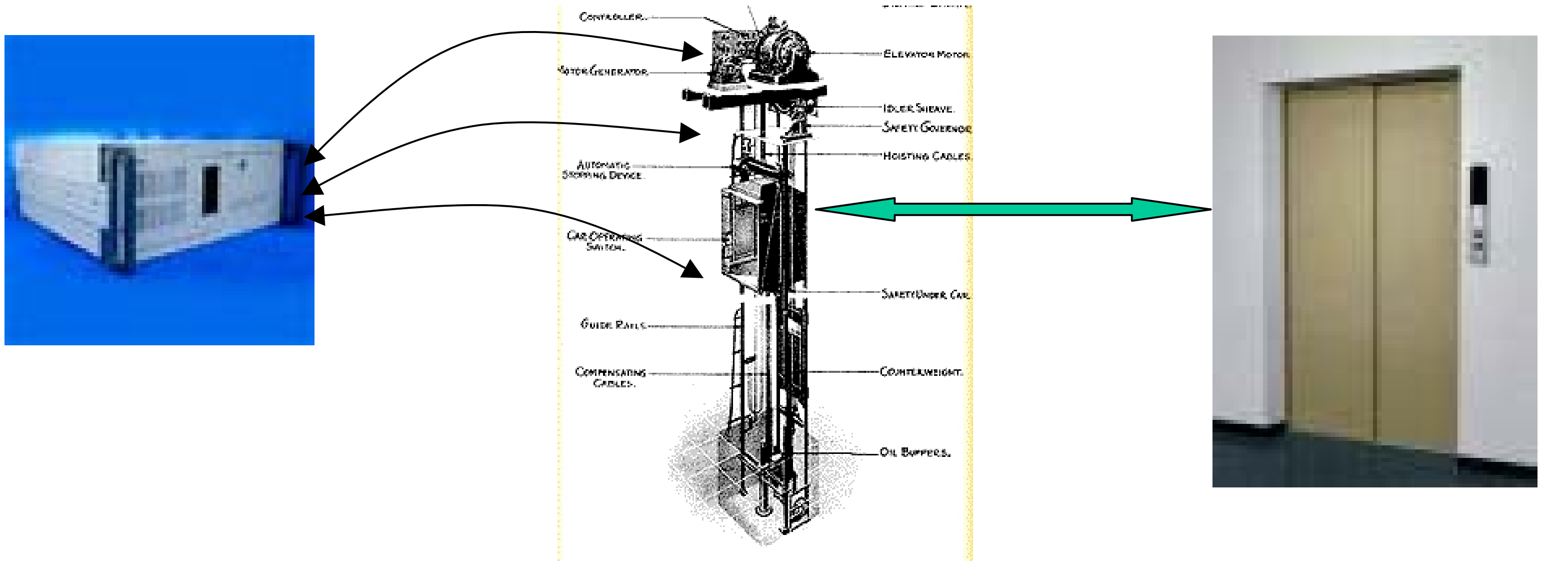


artifice:
the software
& the
computer

physical world:
lift car, shaft, doors,
motor, winding
gear, floors, users,
display lights &c

recognised need:
lift comes when called,
goes to requested
floor, operates
safely &c

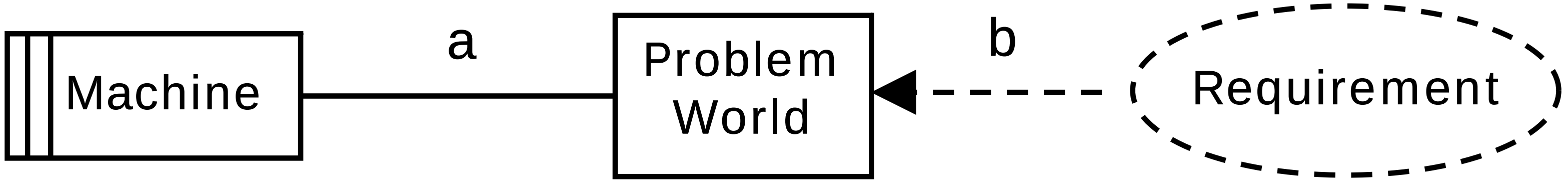
Software to control a lift



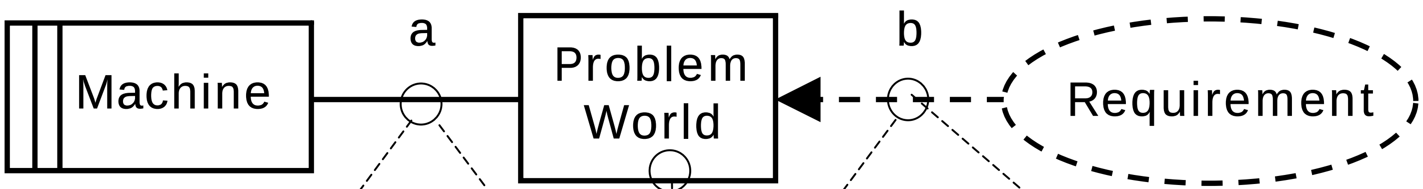
artifice

physical world

recognised need



Functional correctness of a system — version 1



When button[2f] is pressed, and sensor[0] is the most recent sensor on, and motorOn is false:

set dirnUp and motorOn; wait until sensor[2] is on; set motorOff; …

([Machine] Specification $\mathcal{S}$)

button[2f] is pressed iff a user calls the lift to come to floor 2;

dirnUp and motorOn causes the lift car to rise;

sensor[2] comes on when the lift is at floor 2;

when motorOn is false lift stops; …

(Problem World Properties $\mathcal{W}$)

When the lift is at one floor, and a user calls the lift to another floor, it comes;

…

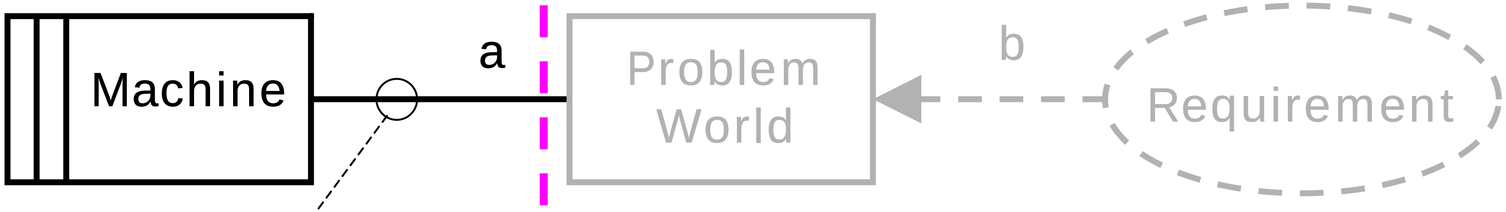
(Requirement $\mathcal{R}$)

$$\mathcal{S}, \mathcal{W} \models \mathcal{R}$$

Software-intensive problem worlds

- The problem world of a software-intensive system
  - Is non-formal and has few regular patterns
  - Has problem-significant state
- A non-formal world admits reasoning
  - Using imperfect abstractions
  - Using formal reasoning (but the conclusions may be false even if the premisses are true)
  - The notion of ‘correctness’ is dubious
- Problem world state
  - May be (imperfectly) represented in the software
  - May change without machine interaction

Can we ignore the problem world?



When button[2f] is pressed, and sensor[0] is the most recent sensor on, and motorOn is false:

set dirnUp and motorOn; wait until sensor[2] is on; set motorOff; …

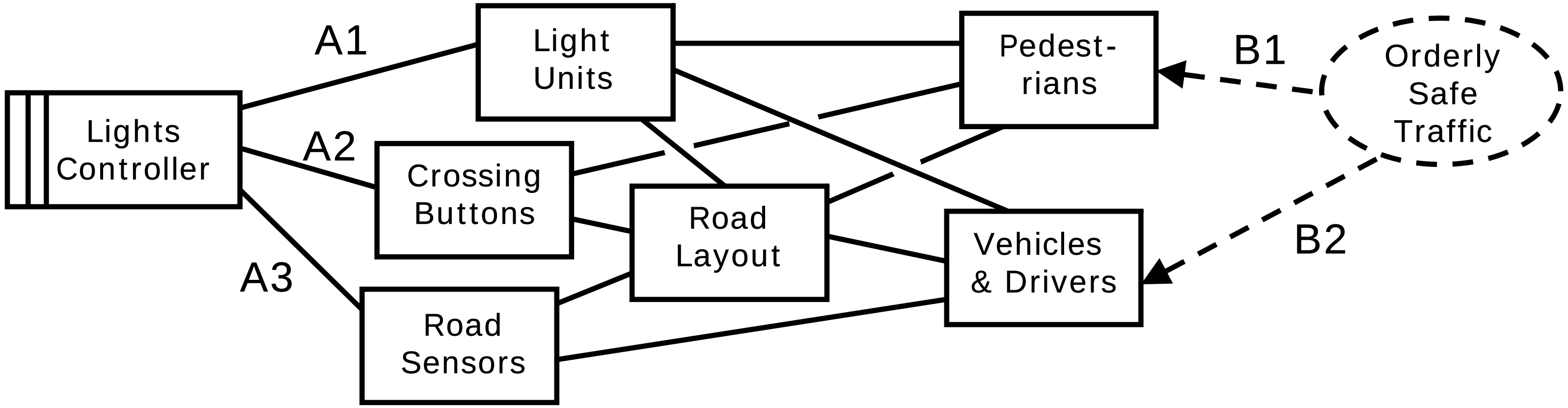
([Machine] Specification *S*)

The programmer's task is [only] to build the (formal) Machine to satisfy a given (formal) Specification at interface a: the specification acts as a 'logical firewall' …

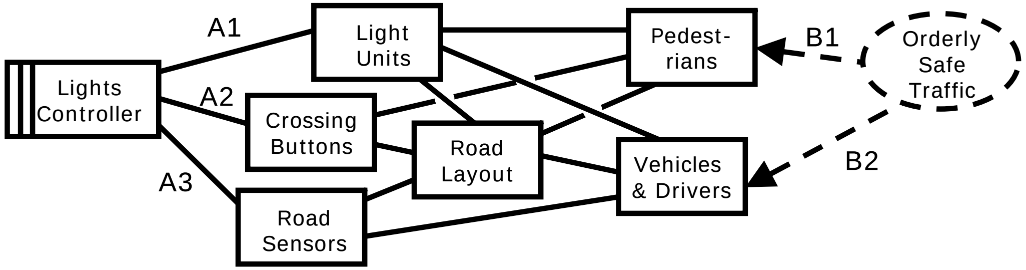
adapted from: E W Dijkstra;
On the Cruelty of Really Teaching Computer Science;
CACM 32,12, December 1989, pp1398-1404

The problem world is unavoidable

- Controlling a complex traffic intersection with traffic lights, pedestrian crossings and road sensors
- Control machine specification?
- The problem world:



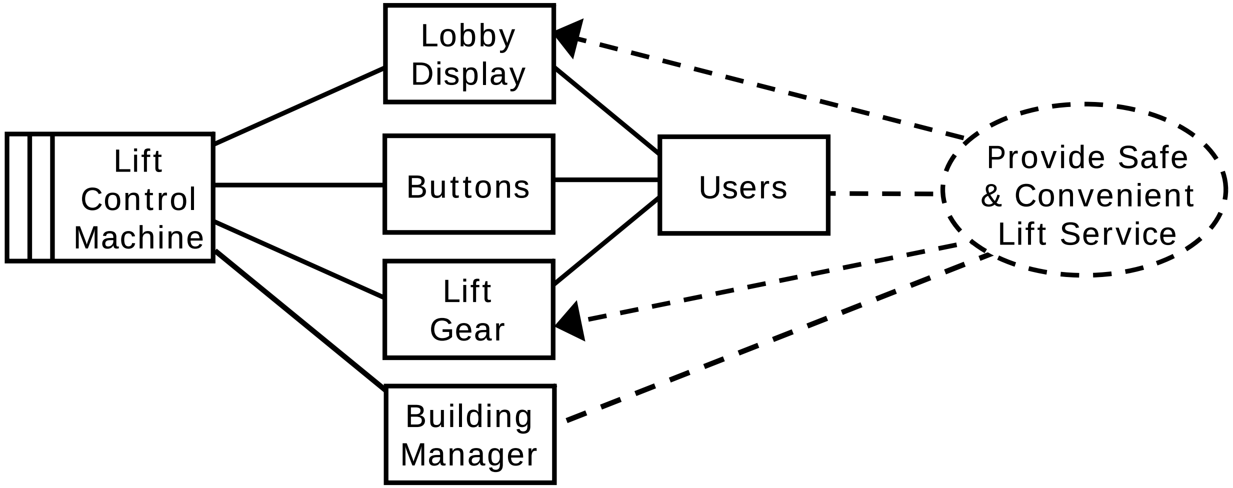
The problem world is unavoidable



- A specification expressed only in terms of A1,A2,A3 would be impossible to understand or check
  - Because the problem world is non-formal and irregular
- Specification structure must be based on requirement ... (allowing efficient movement, avoiding collision, etc)
- ... taking explicit account of problem domain properties
  - Paths through the intersection, locations of light units, vehicle and pedestrian speeds and accelerations, lines of sight, driver behaviours, room for halted vehicles, traffic densities, sensor properties, ...

Decomposing the problem

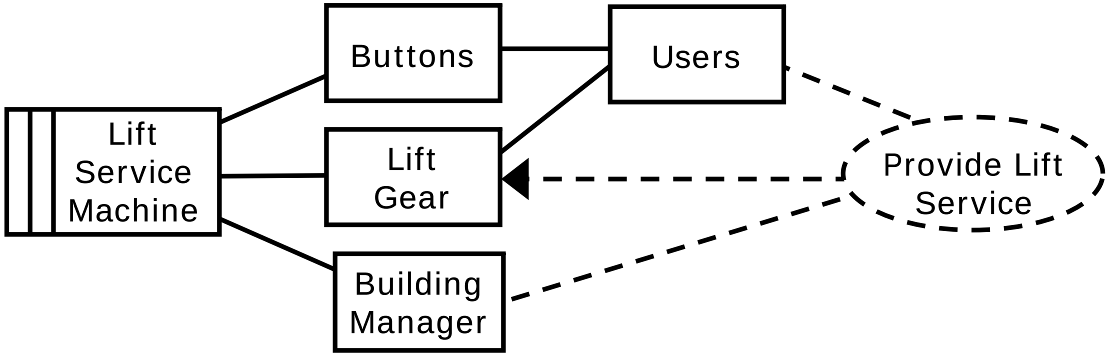
Problem:
Provide safe convenient lift service &c



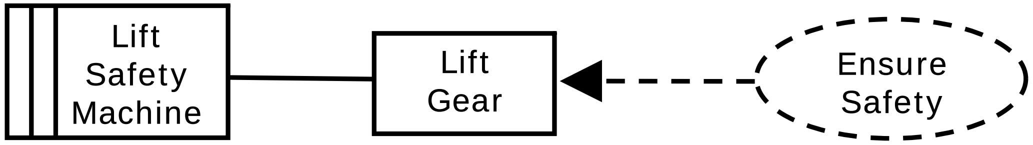
- Many functional requirements
  - Provide lift service in response to requests, with priority management (eg express lifts, rush-hour &c)
  - Ensure safety when equipment fails
  - Display lift position on indicators in hotel lobby
- An approach to decomposition
  - Each subproblem:
    - Machine[sp]+Requirement[sp]+World[sp]

Decomposing the problem

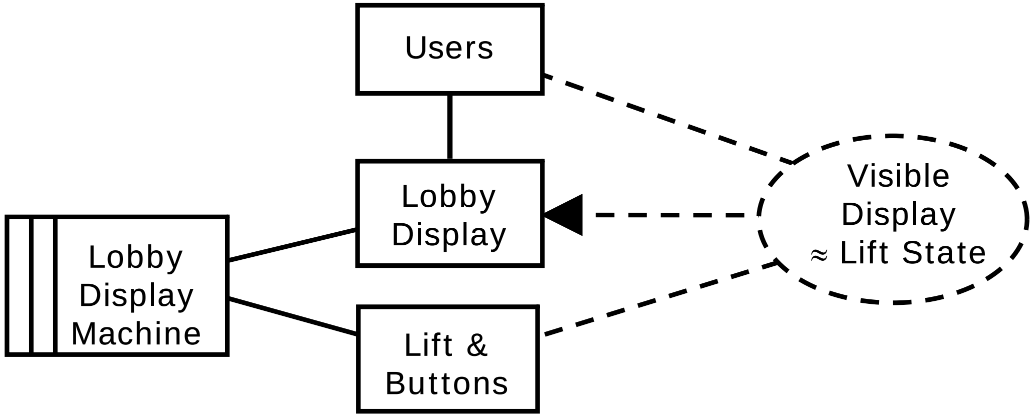
Subproblem 1:
Provide Lift
Service



Subproblem 2:
Ensure safety

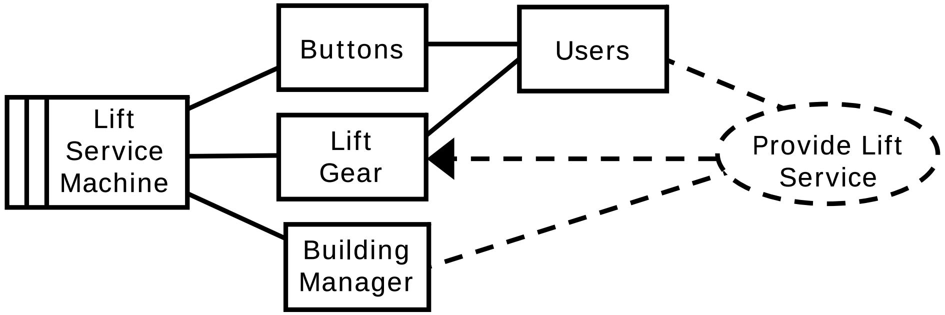


Subproblem 3:
Maintain
Lobby Display

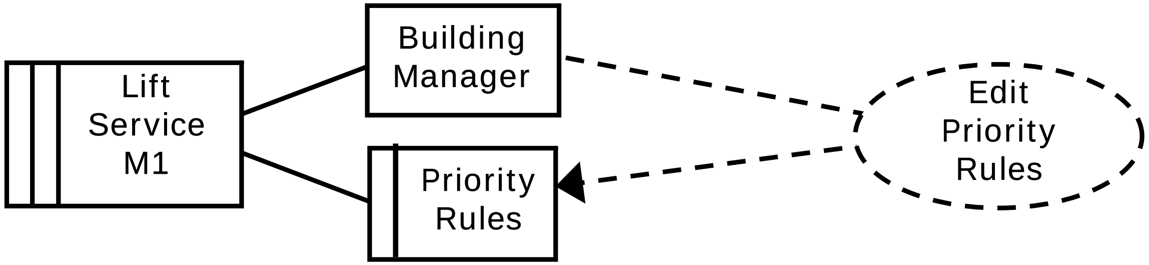


Decomposing the problem

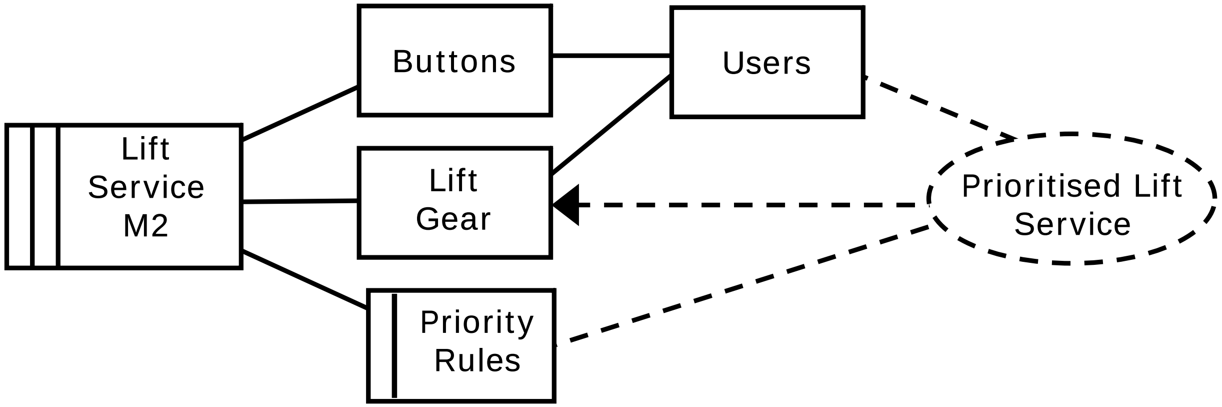
Subproblem 1:
Provide Lift
Service



Subproblem 1a:
Edit Priority
Rules

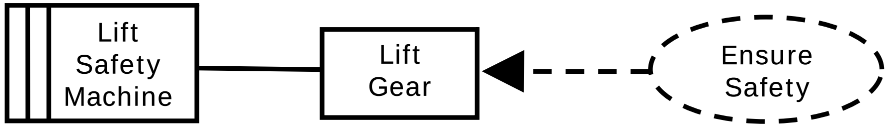


Subproblem 1b:
Prioritised
Lift Service

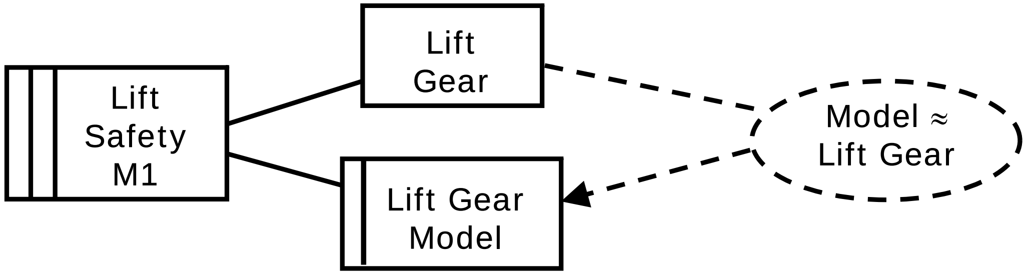


Decomposing the problem

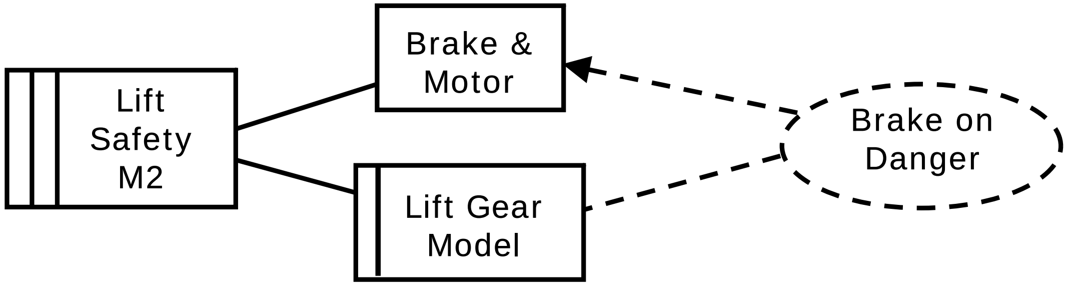
Subproblem 2:
Ensure Safety



Subproblem 2a:
Maintain Lift
Gear Model Domain

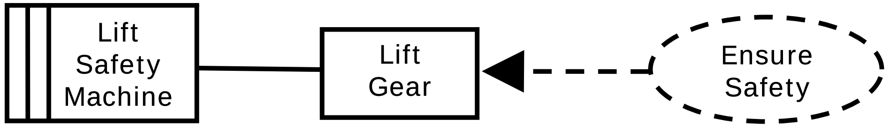


Subproblem 2b:
On Danger, Brake
on & Motor off



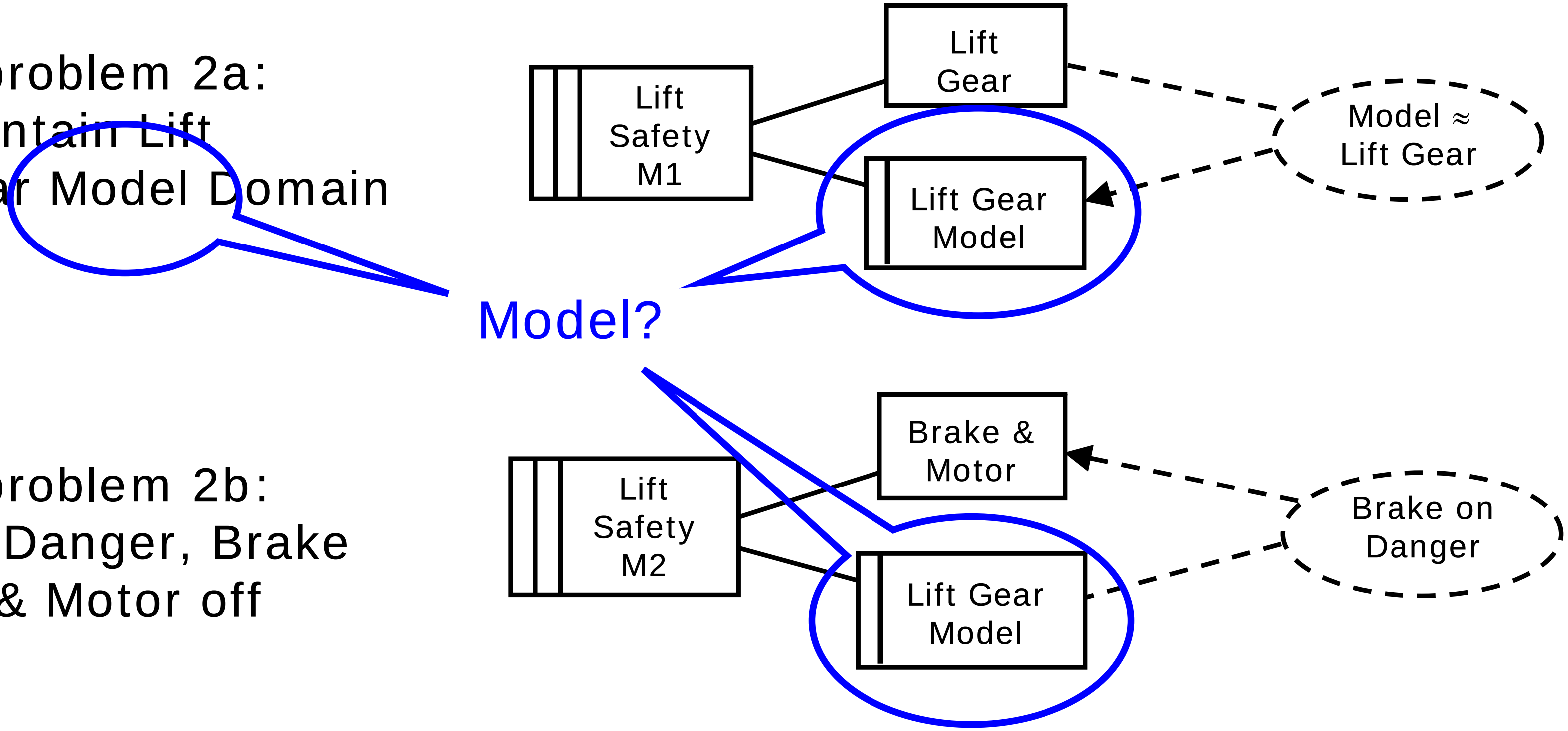
Decomposing the problem

Subproblem 2:
Ensure Safety



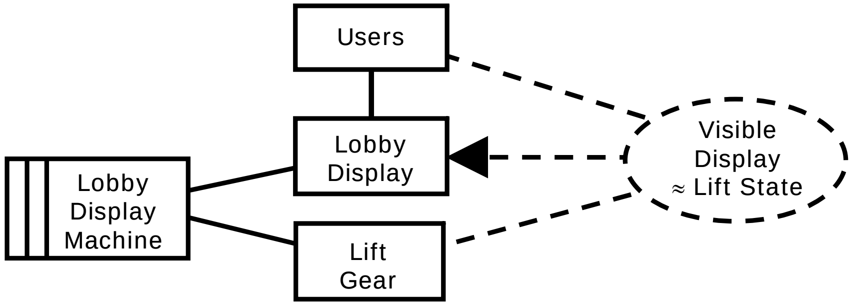
Subproblem 2a:
Maintain Lift
Gear Model Domain

Subproblem 2b:
On Danger, Brake
on & Motor off

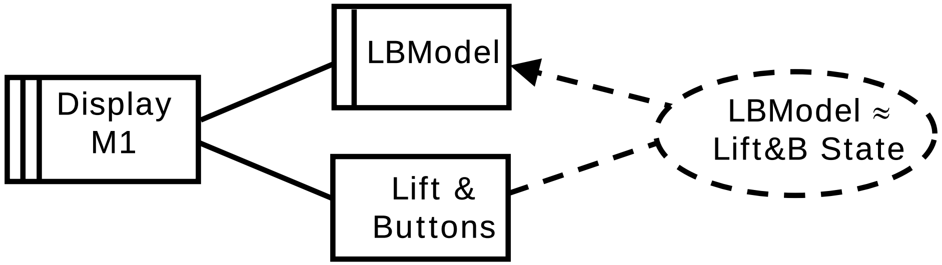


Decomposing the problem

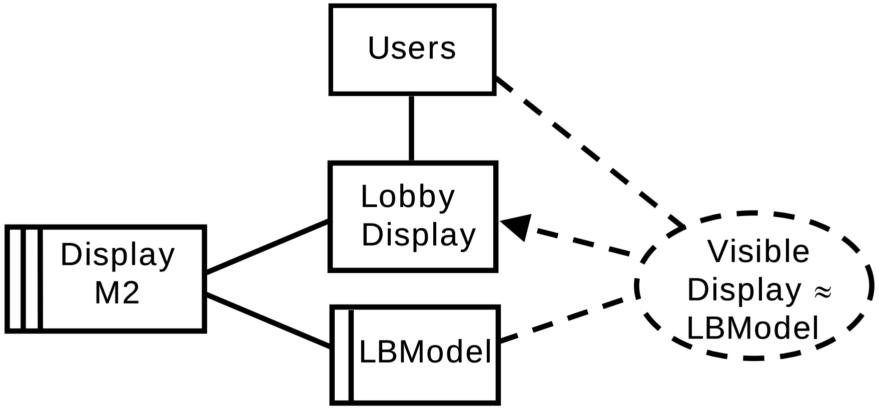
Subproblem 3:
Maintain
Lobby Display



Subproblem 3a:
Maintain Lift
Gear Model Domain



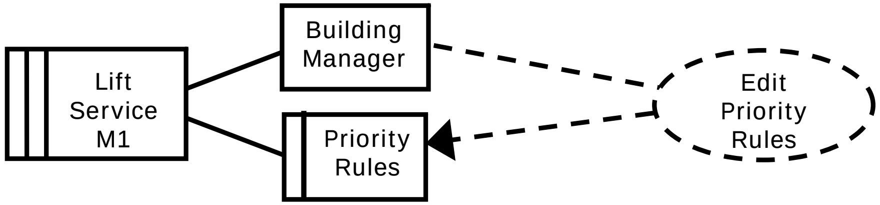
Subproblem 3b:
Align Visible
Display with Model



Decomposition as normal design

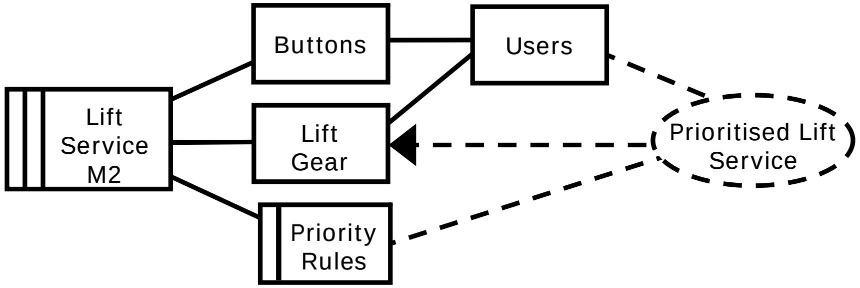
- Subproblems of known classes
  - eg: “Edit Rules” is a workpieces problem

Subproblem 1a:
Edit Priority
Rules



- Subproblems may be considered in isolation
  - “Apply Rules” is a separate (behaviour) problem
- Composition may be considered separately

Subproblem 1b:
Prioritised
Lift Service



Normal design evolves towards dependablity

Toyota 1992

- 106 years of development from 1886 (Karl Benz)
  - 4 Wheels (year 2)
  - Enclosed Cab (year 24)
  - 4-wheel Brakes (year 33)
  - Independent FS (year 47)
  - Unitary Body (year 52)
  - …

"The engineer engaged in such design knows at the outset how the device in question works, what are its customary features, and that, if properly designed along such lines, it has a good likelihood of accomplishing the desired task."

W G Vincenti;
What Engineers Know and How They Know It;
Johns Hopkins 1993

Radical design gives low expectations

Karl Benz 1886

- Year 0 of a radical design
  - 3 Wheels
  - Open Cab
  - Rear-wheel Brakes
  - Cart springs
  - Driver in centre
  - Steered by Tiller

"In radical design, how the device should be arranged or even how it works is largely unknown. The designer has never seen such a device before and has no presumption of success. The problem is to design something that will function well enough to warrant further development."

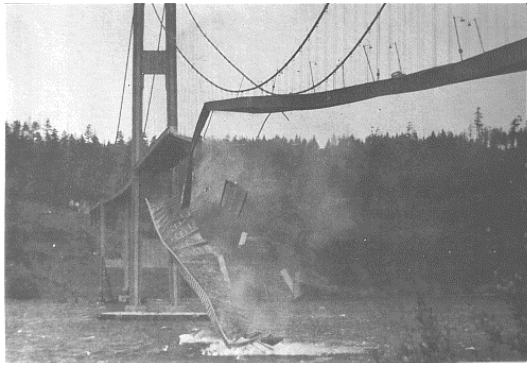
W G Vincenti;
What Engineers Know and How They Know It;
Johns Hopkins 1993

The significance of normal design

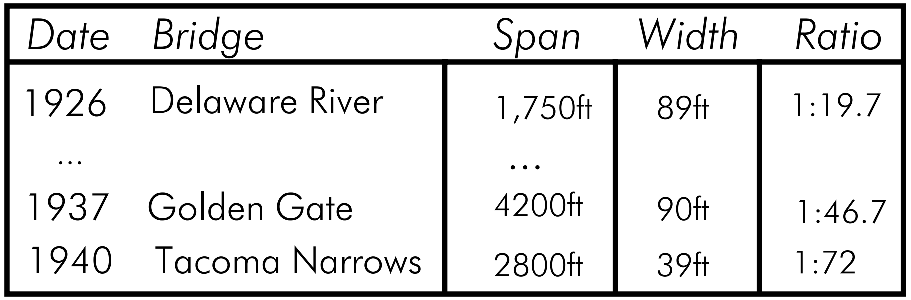
- Normal design develops products
  - From well-understood components (‘devices’)
  - Combining components in well-understood ways
- Well-understood components
  - Developed by normal design at a finer granularity
  - Address known component concerns
- Combining components
  - Subproblem composition
  - Composition has its own concerns
- An aspect of accumulated engineering knowledge
  - Which concerns are important
  - How to address them effectively

The vital importance of normal design

Failure of the Tacoma Narrows Bridge 1940



Theodore Condron's table of ratios



Date	Bridge	Span	Width	Ratio
1926	Delaware River	1,750ft	89ft	1:19.7
...		...		
1937	Golden Gate	4200ft	90ft	1:46.7
1940	Tacoma Narrows	2800ft	39ft	1:72

- Condron's recommendation: widen the roadway to 52ft
  - It would have worked, but it was ignored

C. Michael Holloway; From Bridges and Rockets, Lessons for Software Systems
Proc 17th International System Safety Conference, 1999

Subproblem concerns

- Concerns are mostly about things going wrong
  - Avoiding failures
  - Graceful degradation in presence of failure
- Subproblem and domain classes raise typical concerns
  - Initialisation concern for Lift Service subproblem
    - State of problem world when program starts
  - Breakage concern for Lift Gear domain
    - eg: motor off for $\geq n$ msec between up and down
  - Senseless commands for Edit Rules subproblem
    - eg: delete when nothing selected
  - Identities concern for Maintain Lobby Display
    - Machine interacting with floor[f], light[f]?
    - (Patient Monitoring is a serious example)

An initialisation concern

- US Army handheld ‘plugger’*
- User set missile target; saw battery was low; replaced battery; sent target instructions to B52

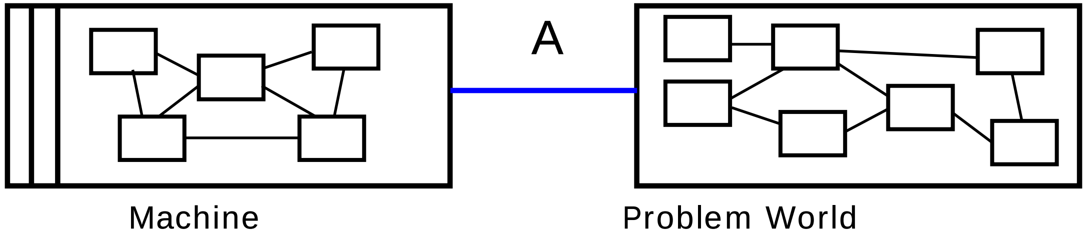
* Personal Lightweight GPS Receiver

- Missile killed the user: 4 ‘friendly deaths’
- US Army response
  - “Military personnel training must be improved”
- The design error was elementary
  - Errors of ignoring a concern often seem elementary
  - The designers won’t make this mistake again

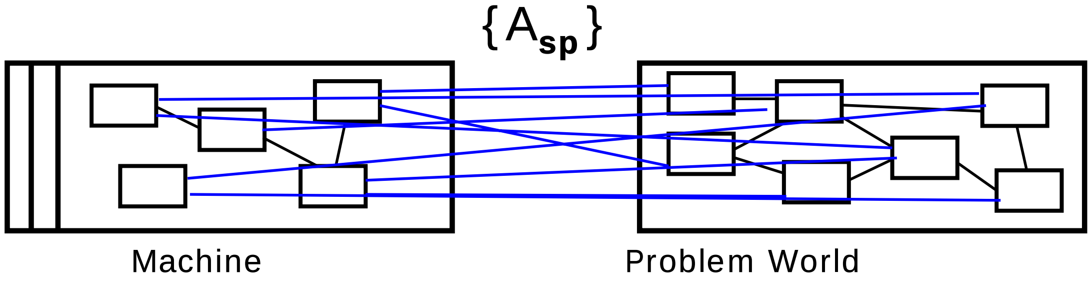
Washington Post
March 24, 2002

Subproblems as system components

- Machine and problem world decompositions are not separate
  - Decomposition is not like this …



  - … but like this:



- Subproblems interact directly within the machine …
- … and via common or connected problem domains
  - Subproblems alter each other's domain states

Separating composition concerns

- Why separate composition concerns?
    - Composition concerns need explicit consideration
        - The feature interaction problem
    - Familiar subproblem classes are harder to recognise when complicated by composition concerns
    - Subproblems are harder to understand when complicated by composition concerns
    - Considering composition is hard if you don’t know what you are composing
    - Considering some compositions may be hard if you have to consider each component separately

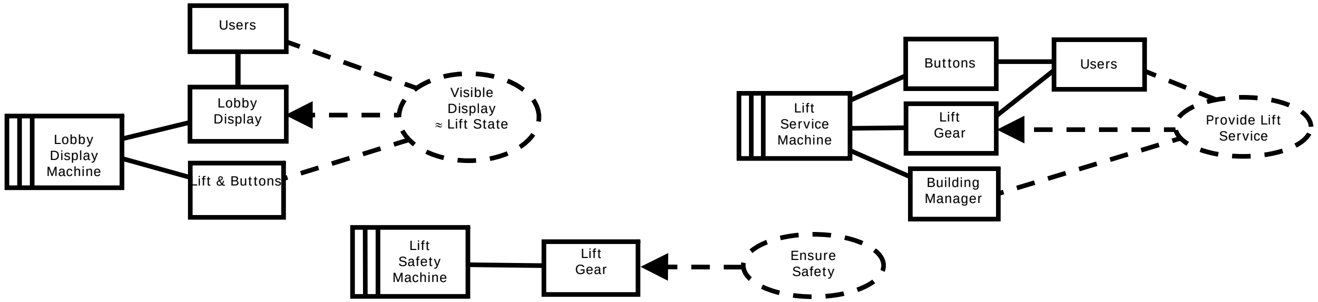
Composition concerns

- Interleaving (eg edit/use Priority Rules)
  - Reader sees common domain as static
  - Writer sees common domain as dynamic
- Requirements conflict (eg MotorOn/MotorOff)
  - Lift Service requires MotorOn
  - Lift Safety requires MotorOff
- Redundancy (eg two similar Lift Models)
  - Lobby Display models fault-free floor sensors
  - Lift Safety models faulty $\vee$ fault-free floor sensors
- Precedence (eg Lift Safety, Lobby Display)
  - Errors in Lobby Display subproblem must not spoil execution of Lift Safety subproblem

Subproblem composition

- When subproblems have common problem domains
  - eg: edit Priority Rules and use them in Lift Service
  - eg: maintain Lift Gear Model and use for Lobby Display
- Subproblem machine relationships include
  - Writer and Reader (eg of a model domain)
  - Free parallel execution (eg Lift Service and Lobby Display)
  - Sequential (eg Maintain Lift Gear Model, Brake on Danger)
  - Overriding (eg Lift Service, Brake on Danger)
- Composition may be achieved by
  - Adjusting subproblem function for composition
  - A composition machine controlling subproblems
  - A composition machine arbitrating between subproblems
  - Textual merging of subproblem machines
  - Aspect weaving mechanisms
  - …

Functional correctness of a system — version 2



- $\mathcal{S},\mathcal{W} \models \mathcal{R}$ for each subproblem, but …
  $\mathcal{W}$ is now a ‘rely condition’, not an assertion
- Different subproblems have different ‘views’
  - Lift Gear is fault-free for Prioritised Lift Service
  - Lift Gear is potentially faulty for Lift Safety
- Composition positives: required interactions are realised
  - eg: Lift Service does use Rules as edited by Manager
- Composition negatives: undesired interactions avoided
  - eg: Rules changeover doesn’t deadlock Lift Service
- Non-formal world ⇒ unbounded set of negatives
  - Normal design identifies the most likely and important

Unit testing and integration testing

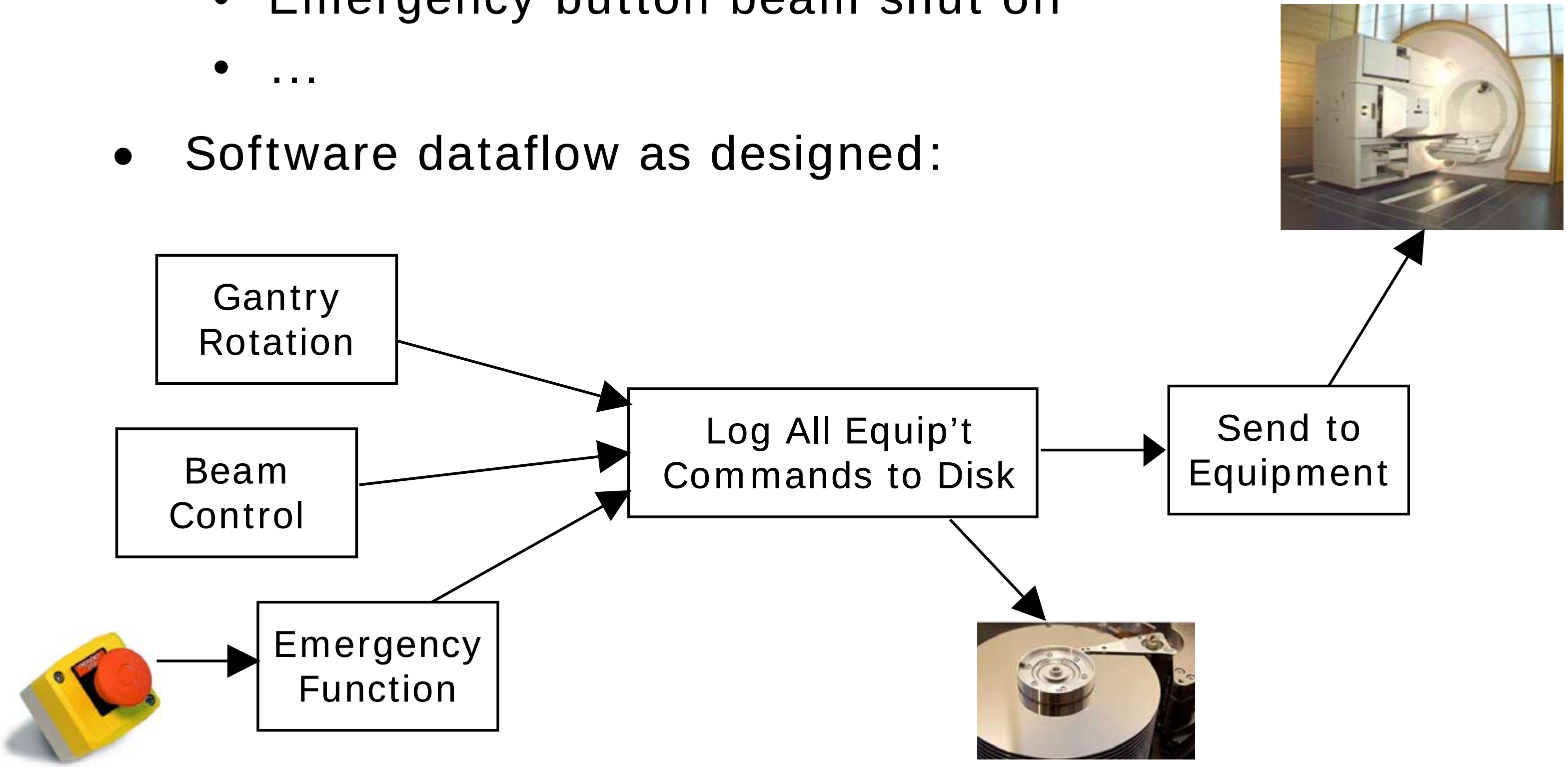
- An obvious conjecture:
  - Unit test is testing a subproblem
    - Test editing Priority Rules in isolation
    - Test using Priority Rules in isolation
  - Integration test is testing subproblem composition
    - Examine Rules composition design and test it
- Testing in isolation
  - $\mathcal{W}$ for a subproblem machine is its ‘rely condition’
    - Different domain $\mathcal{W}$s for different subproblems
  - $\mathcal{W}$ must hold for the unit test
    - By problem domain simulation
    - By restricted choice of test cases

The significance of normal design in testing

- Normal design encompasses
    - Standard designs and practices
    - Knowledge of concerns and likely faults
- Knowledge of likely faults in programming
    - eg: writing assignment for equality test in C
    - eg: off-by-1 errors in loops
- Knowledge of likely faults in specifications
    - Subproblem concerns
        - eg: Plugger initialisation
        - eg: Patient Monitoring identities
    - Composition concerns
        - eg: unexpected subproblem state combination
        - eg: faulty precedence in a Therapy System

Faulty precedence: an example

- Therapy machine functions
  - Deliver dosage
  - Log all commands to equipment
  - Emergency button beam shut off
  - …
- Software dataflow as designed:



Envoi

- Testing: an activity of searching?
  - Searching for faults
- Search should be guided
  - Look for faults where faults are likely
    - eg: “if (n=0) { … }”
    - eg: use of uninitialised variable
- Functional structure gives some guidance
  - Subproblem concerns
  - Composition concerns
- Experience gives guidance …
  - … if it is related to a structure of normal development