Engineering
and
Software Engineering

Michael Jackson
jacksonma@acm.org

Future of Software Engineering Symposium
ETH Zürich
22-23 November 2011

NATO Software Engineering Conferences

- In 1967, influenced by the software crisis …
  - … of low project and product dependability …
  - … a NATO study group recommended a conference

In the Autumn of 1967 the Science Committee established a Study Group on Computer Science. • • •

• • • In late 1967 the Study Group recommended the holding of a working conference on Software Engineering. The phrase 'software engineering' was deliberately chosen as being provocative, in implying the need for software manufacture to be based on the types of theoretical foundations and practical disciplines, that are traditional in the established branches of engineering.

- "Theoretical foundations and practical disciplines …
  … traditional in the established branches of engineering"

Engineering and Software Engineering

1. A lesson from traditional engineers
   - Artifact component structure
2. A view of software engineering
   - Engineering behaviour in the world
3. A challenge of software engineering
   - Identifying and combining components
4. About human understanding
   - Three pillars of wisdom

A lesson from traditional engineers

The most obvious lesson: specialisation

- Multiple dimensions of specialisation
  - Theory: control, structural, fluid dynamics, electronics, …
  - Technology: µ-electronics, welding, pre-stressed concrete …
  - Problem world: civil, mining, production, …
  - Requirement: industrial, transportation, …
  - Artifact specialisation is essential
    - End-user artifacts
      - Cars, aircraft, suspension bridges, ...
    - Specialised components
      - Car seats, jet engines, aircraft landing gear, ...
    - Cross-industry components
      - Electric motors, tyres, bolts, cables, ...

A lesson from traditional engineers

Radical design

"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 … 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

- Karl Benz 1886 Patent MotorWagen

- 3 wheels
- Solid tyres
- Bicycle-type wheels
- Open cab
- Rear-wheel brakes
- No front springs
- Rear cart springs
- Belt drive
- No gearbox
- Driver in centre
- Steered by crank
- Flywheel starter

A lesson from traditional engineers

From radical to normal design

- Specialisation allows normal design to evolve
- Radical design, then successive normal designs …

Karl Benz 1886

Ford Model A 1930

Toyota Camry 1992

"… in normal design … the engineer … 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.

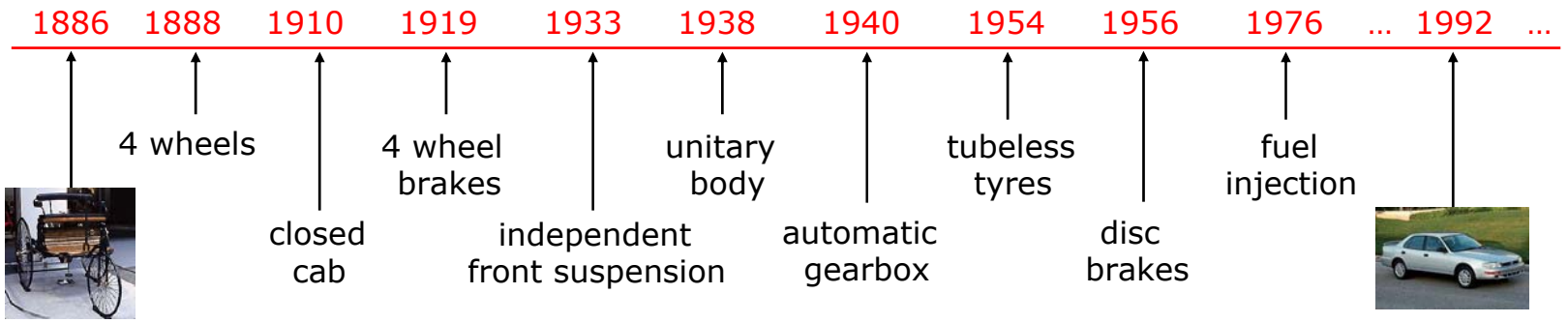
W G Vincenti; What Engineers Know and How They Know It

- Normal design and design practice evolve by learning
- Normal design and design practice support learning

A lesson from traditional engineers

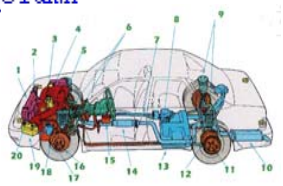
Component structure in normal design

- Evolution of standard component structure
  - Fixes improvements in functionality and efficiency
  - Structures a repository for failure avoidance lessons



- The core: named component types in a standard structure

Battery
Brake Drum
Brake Pads
Catalytic Converter
Cross-Flow Radiator
Disc Brake Caliper
Distributor
Engine
Fuel Lines
Fuel Tank
Fuel Tank Sending Unit
Gear Shift Selector
Intake Manifold
MacPherson Strut Suspension
Muffler
Rack&Pinion Steering and Column
Radiator Hose
Radiator Pressure Cap
Rotor
Wheel Mounting Studs



A lesson from traditional engineers

Normal design: learning to avoid failures

- "Engineering advances by proactive and reactive failure analysis, and at the heart of the engineering method is an understanding of failure in all its real and imagined manifestations."

  Henry Petroski; Design Paradigms

de Havilland DH106 Comet 1 (metal fatigue)

Tacoma Narrows Bridge (wind-induced roadway oscillation)

- Avoiding failure by evolving normal design
  - Radical Comet 1: turbojet, 500mph, pressurised
    - Metal fatigue cracks started at window corners
- Avoiding failure by evolving normal design practice
  - Radical Tacoma Narrows: span:width ratio of 72:1
    - Analyse vertical roadway oscillation

A lesson from traditional engineers

Attaching lessons to artifacts

- Tacoma Narrows lesson
  - Suspension bridge … roadway … stiffness … span/width ratio … wind-induced … vertical oscillation
- Comet 1 lesson
  - Turbojet aircraft … fuselage apertures … torsional stress … pressure stress … metal fatigue

- Specialisation focuses many lessons on components
  - Normal design has standard components and … … clearly recognised design tasks for them
- Knowledge of general principles is very valuable but …
  - … harder to call to mind when it's important and …
  - … harder to apply for specific artifacts

A lesson from traditional engineers

A lack of component structure: Therac-25

- Deaths and injuries, 1985-1987
  - Overconfidence in software
  - Failure to eliminate root causes
  - Unrealistic risk assessments
  - Lack of audit trails
  - Safe versus friendly user interfaces
  - Careless software reuse
  - Confusing reliability with safety
  - Lack of defensive design
  - Poor information display
  - Complacency about radiation therapy machines
  - Inadequate module and regression testing

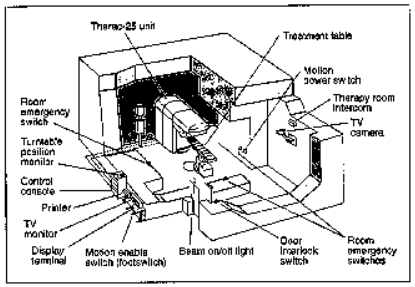


Figure 1. Typical Therac-25 facility.

Leveson and Turner July 1993

- Radiotherapy machines twenty years later, 2001-2008
- Hundreds of overdoses, several deaths

New York Times January 2010

Engineering and Software Engineering

1. A lesson from traditional engineers
   - Artifact component structure
2. A view of software engineering
   - Engineering behaviour in the world
3. A challenge of software engineering
   - Identifying and combining components
4. About human understanding
   - Three pillars of wisdom

A view of software engineering

Software engineering as systems engineering

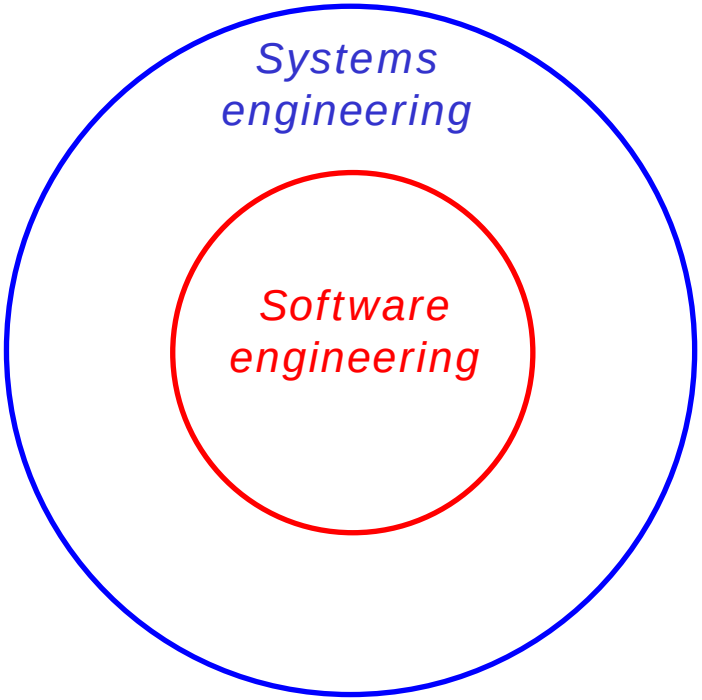
- Software that monitors and controls the physical world
  - Natural, engineered, human

Systems engineering

- Railways, aircraft, banks, power stations, hospitals, libraries, factories, passports, welfare, taxation ...

Software engineering

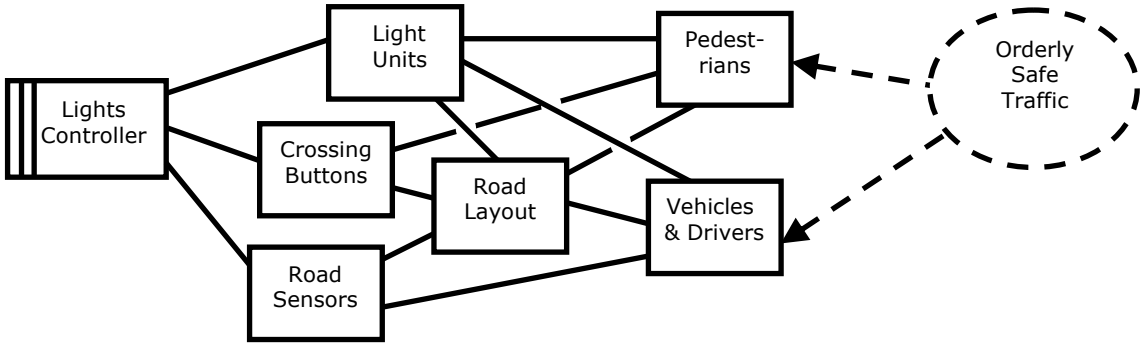
- Systems engineering taking the physical world as given
- This is a separation of concerns, not a restriction of possibilities



A view of software engineering

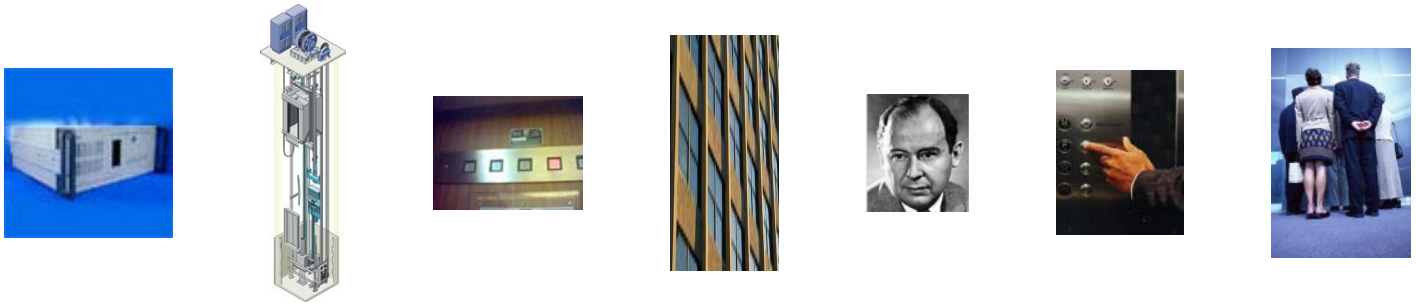
Two software engineering problems

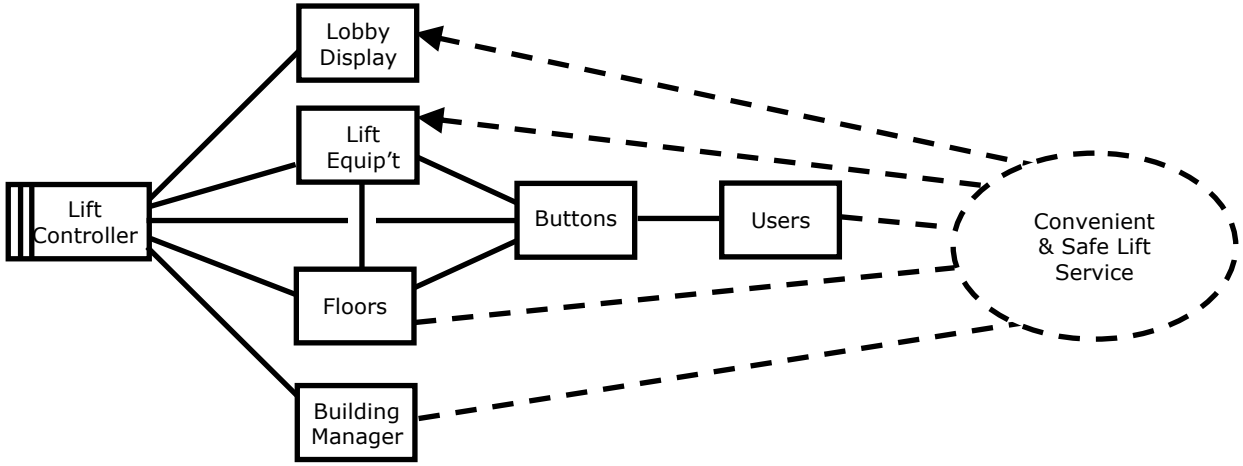
- Software that monitors and controls the physical world
  - Natural, engineered, human domains
- Controlling traffic at a complex road crossing



Two software engineering problems

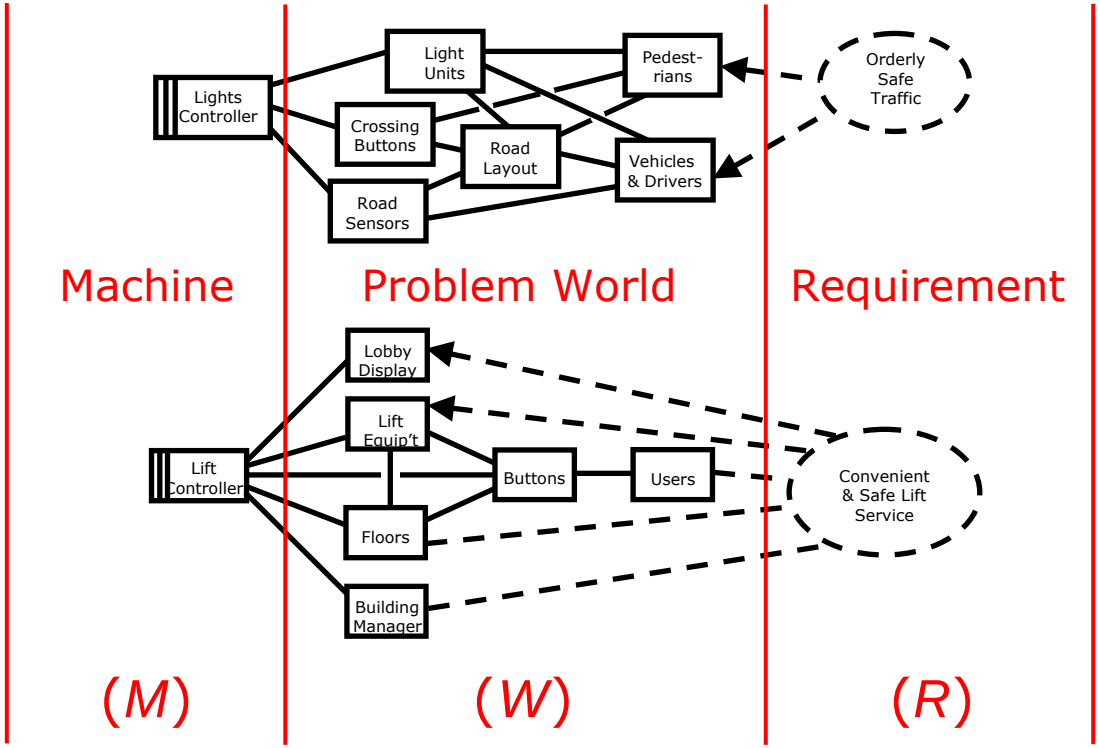
- Software that monitors and controls the physical world
  - Natural, engineered, human domains
- Controlling lifts in a large multiple-use building





A view of software engineering

Two software engineering problems



- Our product is not the behaviour of the machine ...
  - ... it's the behaviour of the problem world
- Formalised SE problem: develop *M* such that $M, W \models R$

Some kinds of failure in $M, W \models R$

- R is not what's really required ('requirements elicitation')
  - "Overriding priority for pedestrian requests"
  - "Respond directly to earliest lift request first"
- W does not hold in the problem world ('modelling')
  - "Conservation of vehicles in crossing area"
  - "Initially lift is at ground floor with doors open"
- M does not hold of the machine ('programming')
  - "Vehicle phasing continues during pedestrian phases"
  - "Floor requests persist when lift direction reversed"
- $M, W \models R$ does not hold ('formal reasoning')
  - "No traffic flow direction is starved"
  - "Accepted lift requests are satisfied in current pass"

Engineering and Software Engineering

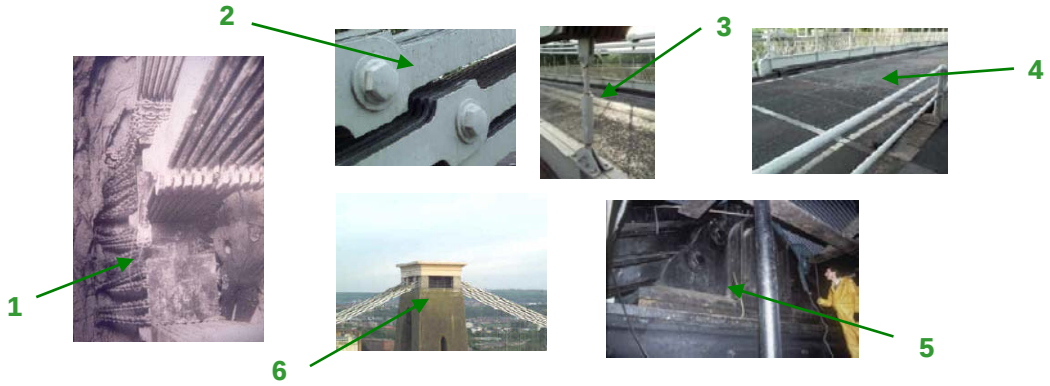
1. A lesson from traditional engineers
   - Artifact component structure
2. A view of software engineering
   - Engineering behaviour in the world
3. A challenge of software engineering
   - Identifying and combining components
4. About human understanding
   - Three pillars of wisdom

A challenge of software engineering

Physical components

- Artifact types have characteristic component structures
- Standard bridge types: suspension, arch, swing, truss, …
  - Each has its own operational principle
  - Each has a standard structure of named component types

1 Anchorage
2 Chain
3 Hanger
4 Roadway
5 Saddle
6 Tower

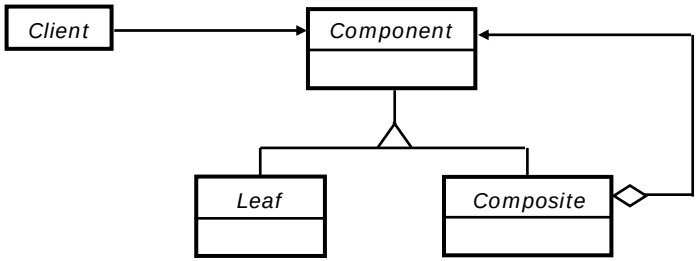


- Functional-physical mapping is nearly 1-1
  - Failure can often be explained in terms of components and component interactions

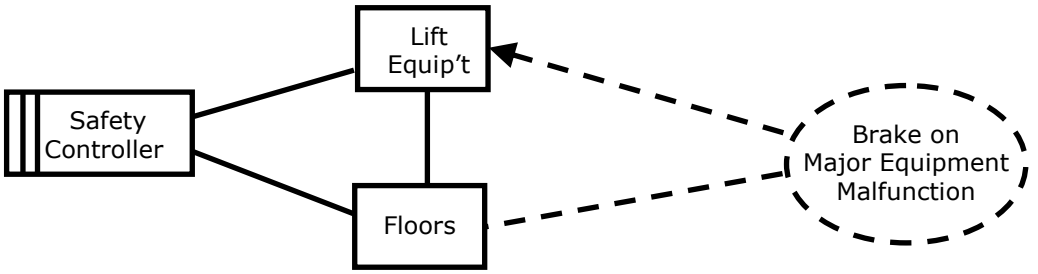
A challenge of software engineering

Components in software-engineered systems

- In programming, the whole product is software
  - A component is a software object or an object assembly



- In SE, the whole product is a problem world behaviour
  - A component is a behaviour projection ...
  - ... of the machine and the relevant problem world parts



A challenge of software engineering

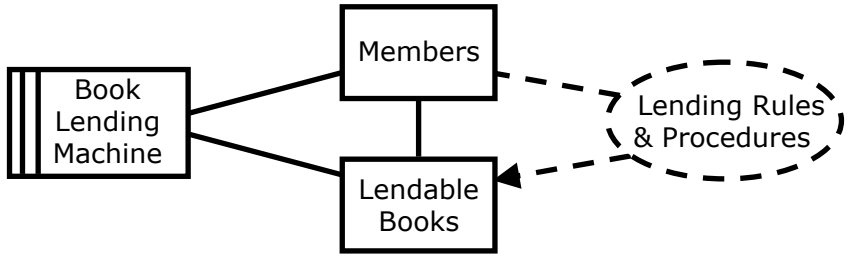
Complexity in system behaviour

- Many ‘stakeholders’
  - Operators, users, customers, regulators, ...
  - Requirement conflicts
  - Multiple system features and external interfaces
- Many system operation contexts (’modes’?)
  - Weekday business hours, weekends, fire brigade control, maintenance engineer control, test inspector control, major conference on disability, minor equipment failure, major equipment failure, ...
- Many behaviour projection components
  - Complex component structures
  - Parallelism, concatenation, iteration, choice
  - Shared state within and between problem domains
  - Complexity within and between behaviours

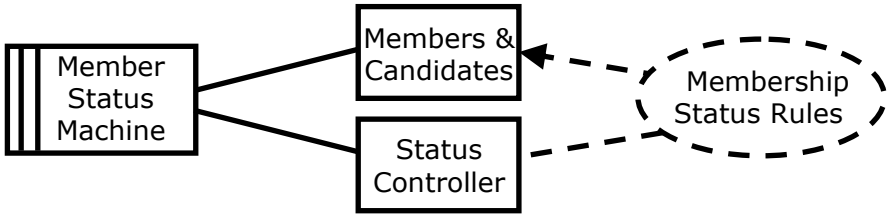
A challenge of software engineering

Component interactions

- Two components in a library system



- Borrowing
  - Reserve, collect, borrow, renew, return, shelve, damage, repair, lose, find, inter-library loan, ...



- Membership
  - Join, pay, renew, resign, cc refusal, change name or address, promote to senior, die, emigrate, go bankrupt, go to prison, ...

- Complexity of each component: intrinsic plus interaction

A challenge of software engineering

Many possible component views

- Functions (behaviours)
  - Component is machine plus problem world
- Problem world entities
  - Component is one book, one member, etc
- Software modules
  - Component defined by programming language
- Software deployment
  - On client, on server

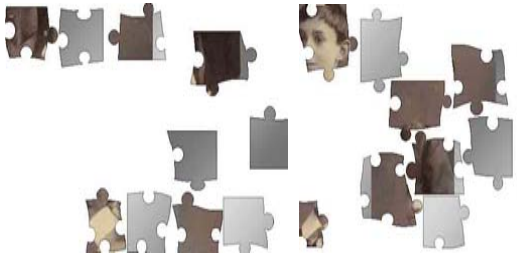
- None of these structures can be ignored
  - Full ‘seamless development’ is impossible

- Functional (behavioural) structure remains essential

A challenge of software engineering

Abandoning behavioural components?

- Avoid defining and combining behavioural components?
  - Reduce to highest common factor
    - Single events with actions
  - Use invariants and pre- and post-conditions
  - Behaviours become implicit and obscure
- Is there a cost?
  - Yes: loss of human understanding
  - A collection of fragments whole picture



... is not the ...

  - How can we even know that we have all the pieces?

Engineering and Software Engineering

1. A view of software engineering
2. Learning from traditional engineers
3. Challenges of software engineering
4. About human understanding

About human understanding

The three pillars of our understanding

- Pillars of our understanding
  - Intuition
    - The logic of contrivance
    - Knowing how and knowing that
  - Experience
    - Learned lessons as a communal possession
    - Specialisation nourishes communal learning
  - Calculation
    - Formal reasoning and analysis
    - Within structure set by intuition and experience

About human understanding

The three pillars of our understanding

- Pillars of our understanding
  - Intuition
    - The logic of contrivance
    - Knowing how and knowing that
  - Experience
    - Learned lessons as a communal possession
    - Specialisation nourishes communal learning
  - Calculation
    - Formal reasoning and analysis
    - Within structure set by intuition and experience

Thank you!