Old School

Techniques that still work no matter how hard we try to forget them

Slide 1
21 July 2009

Keith Braithwaite

Context

“I come as an entertainer, not as a salesman. I want you to enjoy these ideas because I enjoy them”— *Alan Watts*

If IT were a person…

It would be diagnosed with

- ADHD
- Retrograde amnesia
- OCD

If IT were a person…

It would be diagnosed with

- ADHD
  - We have difficulty retaining focus on the job at hand
  - We are very easily distracted by *ooh, shiny!*
- Retrograde amnesia
- OCD

If IT were a person…

It would be diagnosed with

- ADHD
  - We have difficulty retaining focus on the job at hand
  - We are very easily distracted by
- Retrograde amnesia
  - We don’t recall our past
- OCD

If IT were a person…

It would be diagnosed with

- ADHD
  - We have difficulty retaining focus on the job at hand
  - We are very easily distracted by
- Retrograde amnesia
  - We don’t recall our past
  - We don’t recall our past
- OCD

If IT were a person…

It would be diagnosed with

- ADHD
  - We have difficulty retaining focus on the job at hand
  - We are very easily distracted by
- Retrograde amnesia
  - We don’t recall our past
  - We don’t recall our past
- OCD
  - We follow rituals independent of their effectiveness

If IT were a person…

It would be diagnosed with

- ADHD
  - We have difficulty retaining focus on the job at hand
  - We are very easily distracted by
- Retrograde amnesia
  - We don’t recall our past
  - We don’t recall our past
- OCD
  - We <u>follow</u> **rituals** <u>independent</u> **of** <u>their</u> **effectiveness**

Tony Hoare said…

zühlke
empowering ideas

“If we could only learn the right lessons from the successes of the past we would not need to learn from the failures ”

Zombies

Zombies

Given half a chance they *will* eat your brain

- Code the works "first time"
- Structured Programming

These, and others, we should forget

Code that works “first time”

City and Guilds COBOL

- 3 attempts to compile, run and test or fail

There was a time when this sort of thing made sense

Code that works "first time"

There was a time when this sort of thing made sense

Code that works “first time”

Jerry Weinberg tells of being told that

- The computer (singular) earns more than you do, so behave accordingly

Code that works "first time"

The computer learns more than you, behave accordingly

- cost(processor time) >> cost(developer time)
- Cycle time to get feedback—hours to days

Code that works "first time"

The computer learns more than you, behave accordingly

- cost(processor time) >> cost(developer time)
- Cycle time to get feedback—hours to days

In fact, you earn much more than the computer

Code that works "first time"

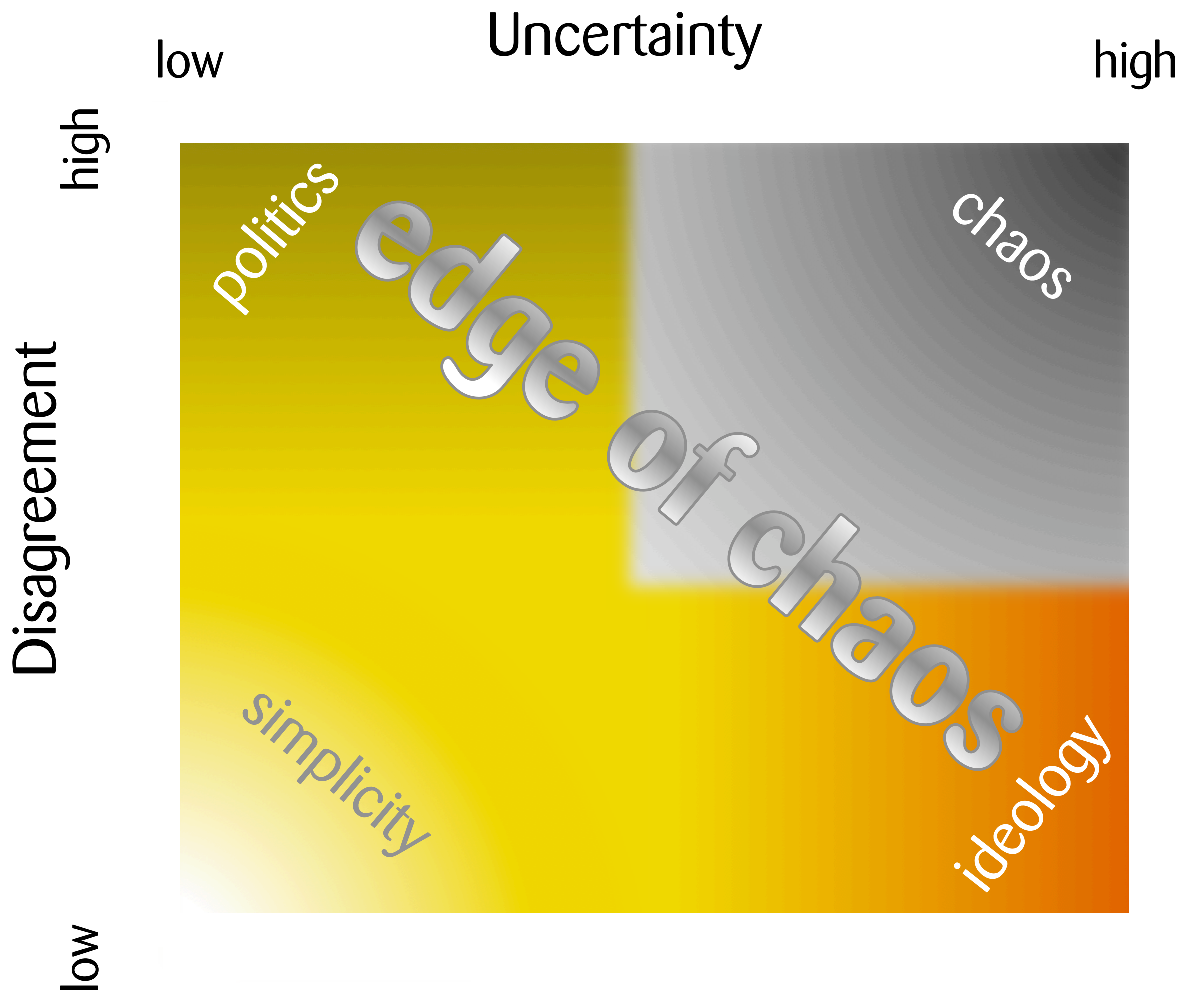
The computer learns more than you, behave accordingly

- cost(processor time) >> cost(developer time)
- Cycle time to get feedback—hours to days

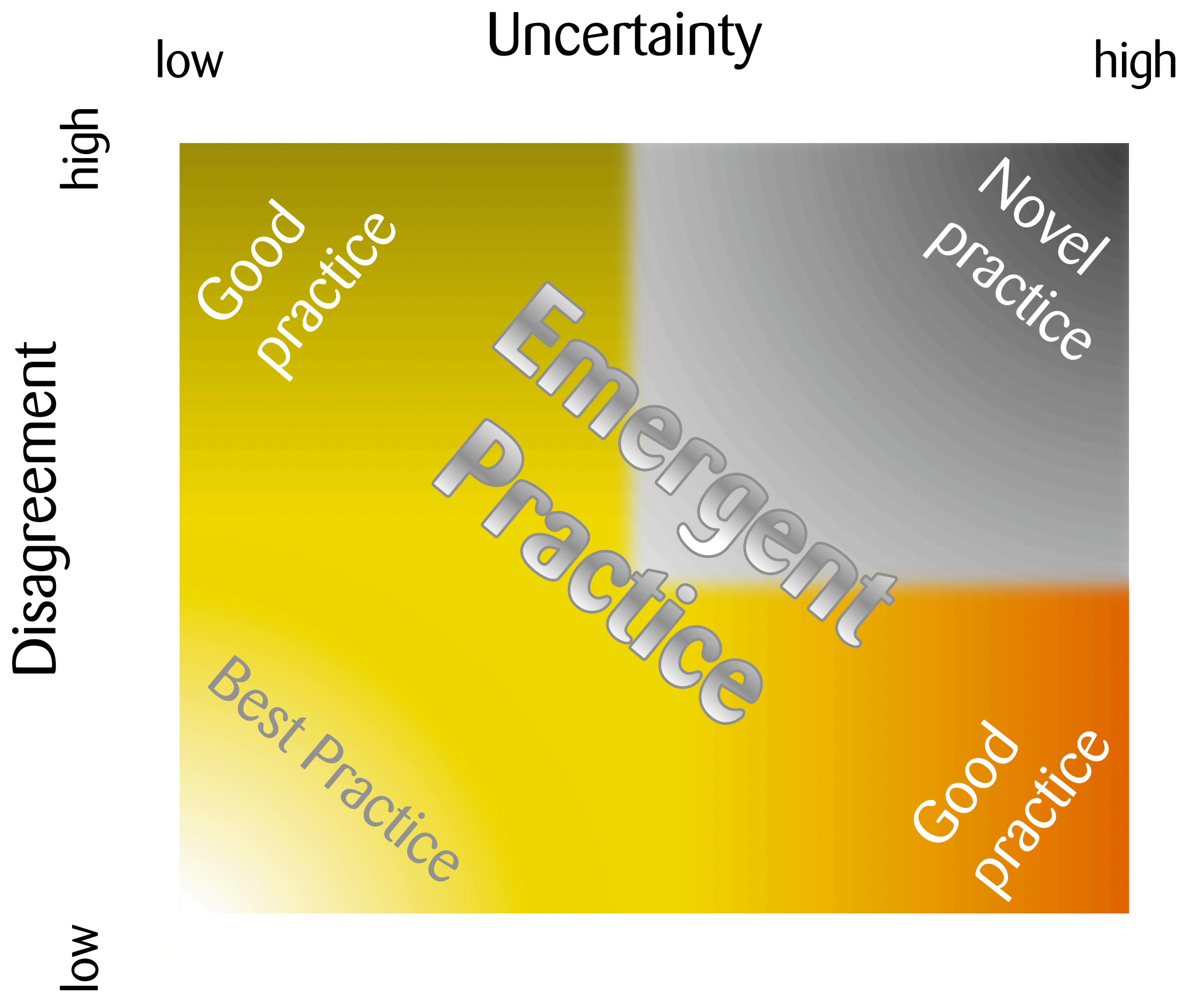
You earn much more than the computer, behave accordingly

- cost(processor time) << cost(developer time)
- Cycle time to get feedback—milliseconds to minutes
- A top-end dev workstation amortised over 3 years
  - £1 per day
  - 2 or 3 *orders of magnitude* cheaper than a programmer

Code that works "first time"

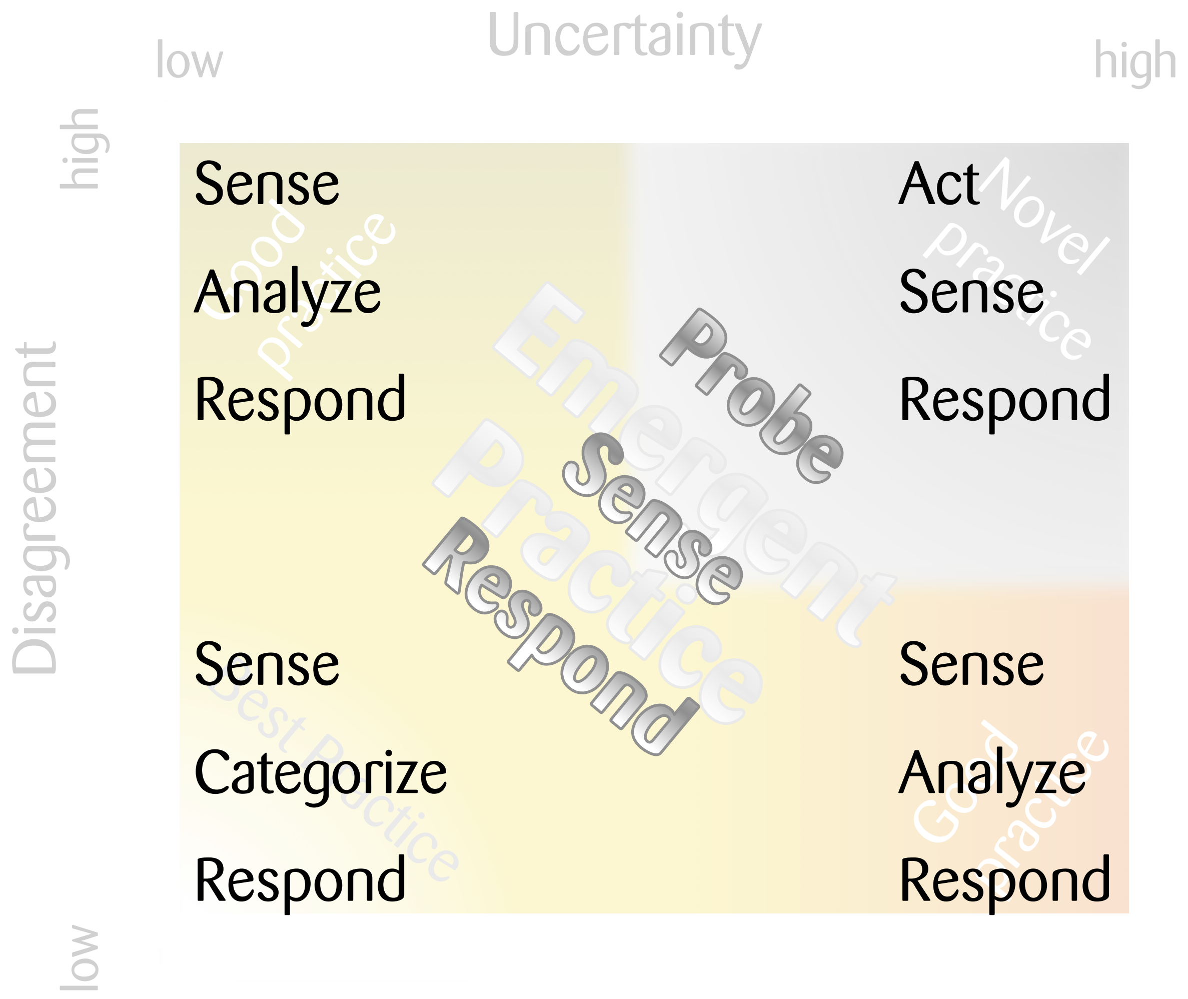


Code that works “first time”



Code that works “first time”

zühlke
empowering ideas



Structured Programming

A “sub-program” had:

- One entry point
- Sequence
- Iteration
- Alternation
- One exit point

Structured Programming

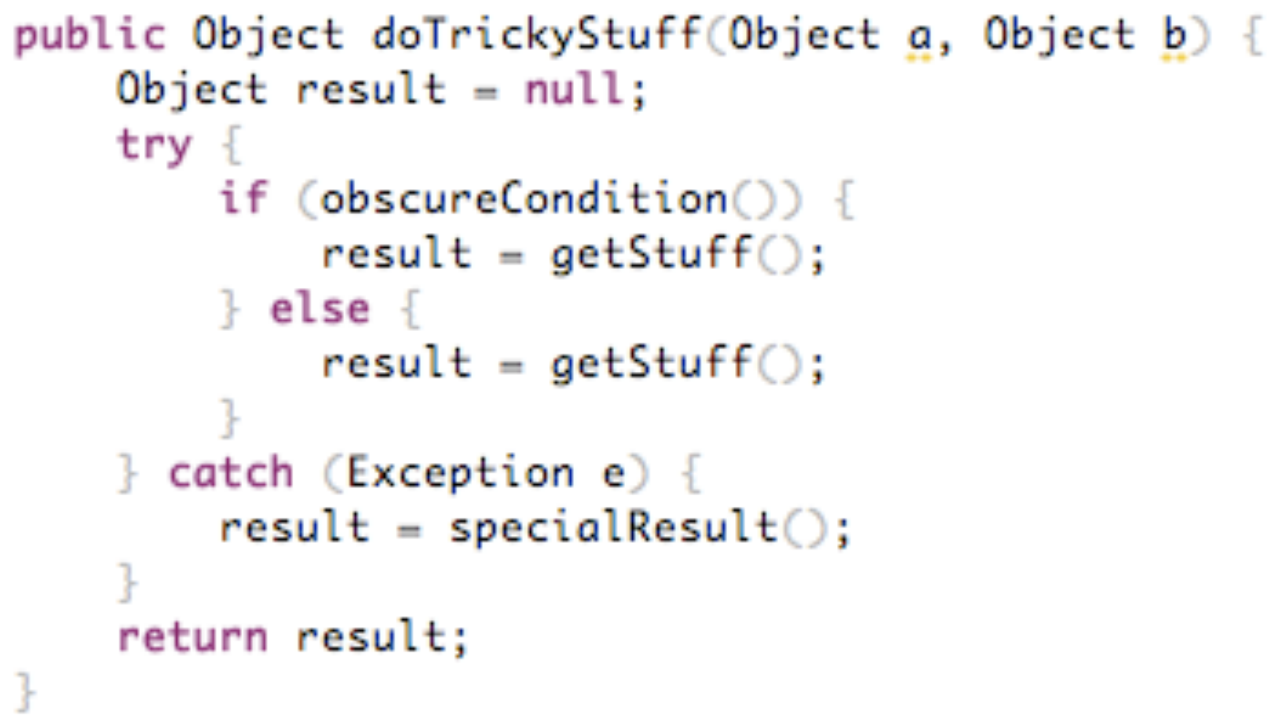
That was a big improvement over spaghetti code

- Especially when flow of control was DIY

```
10 IF (SUM .LE. LLIMIT) THEN
        NUMBER = NUMBER + 1
        SUM = SUM + NUMBER
    GO TO 10
    END IF
```

Structured Programming

But this sort of thing makes no sense:

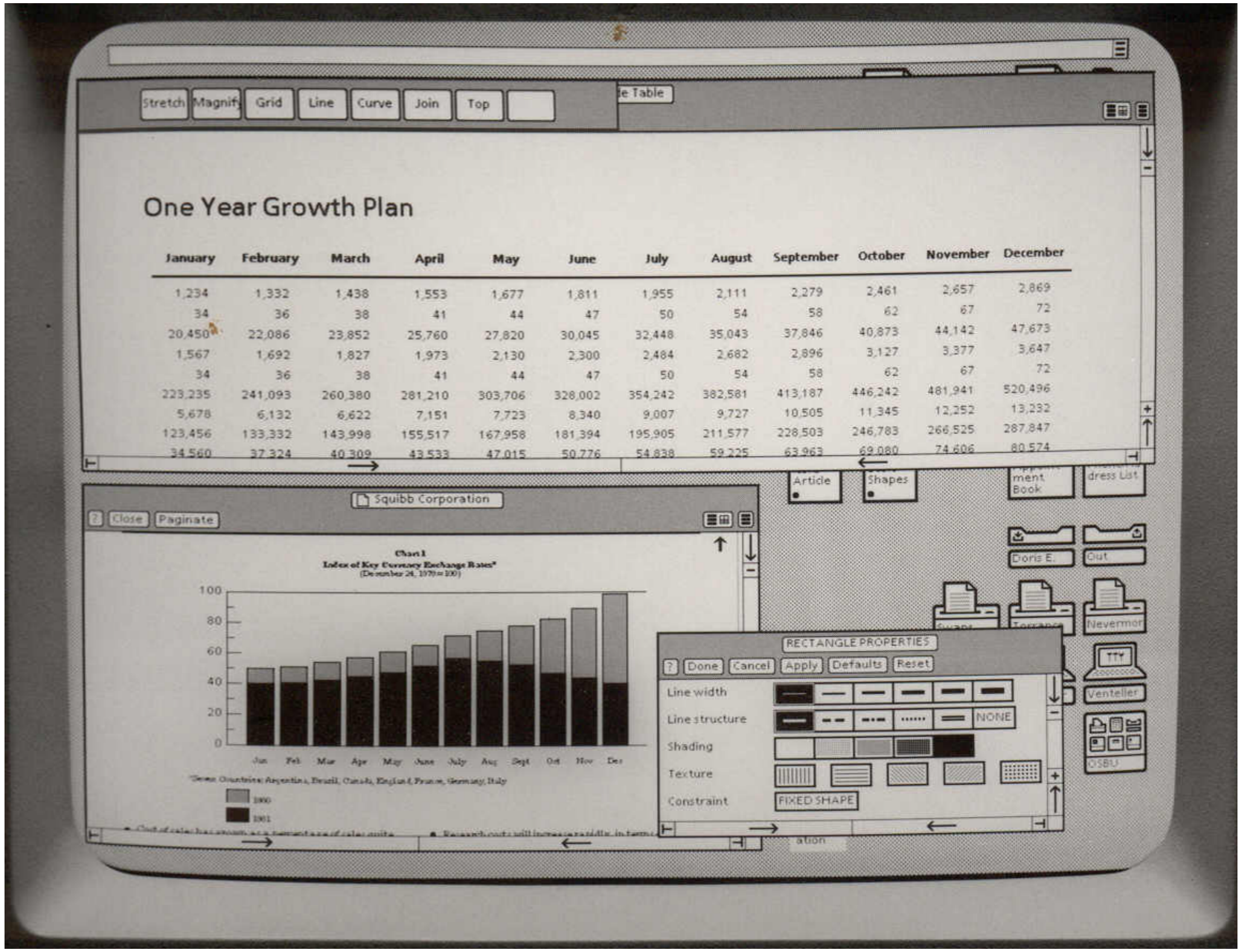


```
public Object doTrickyStuff(Object a, Object b) {
    Object result = null;
    try {
        if (obscureCondition()) {
            result = getStuff();
        } else {
            result = getStuff();
        }
    } catch (Exception e) {
        result = specialResult();
    }
    return result;
}
```

Old School: things we got right

http://commons.wikimedia.org/wiki/File:Vincent_Series_C_Black_Shadow_1950.jpg

Old School: things we got right



One Year Growth Plan

January	February	March	April	May	June	July	August	September	October	November	December
1,234	1,332	1,438	1,553	1,677	1,811	1,955	2,111	2,279	2,461	2,657	2,869
34	36	38	41	44	47	50	54	58	62	67	72
20,450	22,086	23,852	25,760	27,820	30,045	32,448	35,043	37,846	40,873	44,142	47,673
1,567	1,692	1,827	1,973	2,130	2,300	2,484	2,682	2,896	3,127	3,377	3,647
34	36	38	41	44	47	50	54	58	62	67	72
223,235	241,093	260,380	281,210	303,706	328,002	354,242	382,581	413,187	446,242	481,941	520,496
5,678	6,132	6,622	7,151	7,723	8,340	9,007	9,727	10,505	11,345	12,252	13,232
123,456	133,332	143,998	155,517	167,958	181,394	195,905	211,577	228,503	246,783	266,525	287,847
34,560	37,324	40,309	43,533	47,015	50,776	54,838	59,225	63,963	69,080	74,606	80,574

RECTANGLE PROPERTIES

Done Cancel Apply Defaults Reset

Line width

Line structure

Shading

Texture

Constraint FIXED SHAPE

Old School: things we got right

Analysis

Architecture

Modelling

http://commons.wikimedia.org/wiki/File:Vincent_Series_C_Black_Shadow_1950.jpg

15 years and Counting

zühlke
empowering ideas

I've been a professional programmer for about 15 years

- And an amateur for years before that

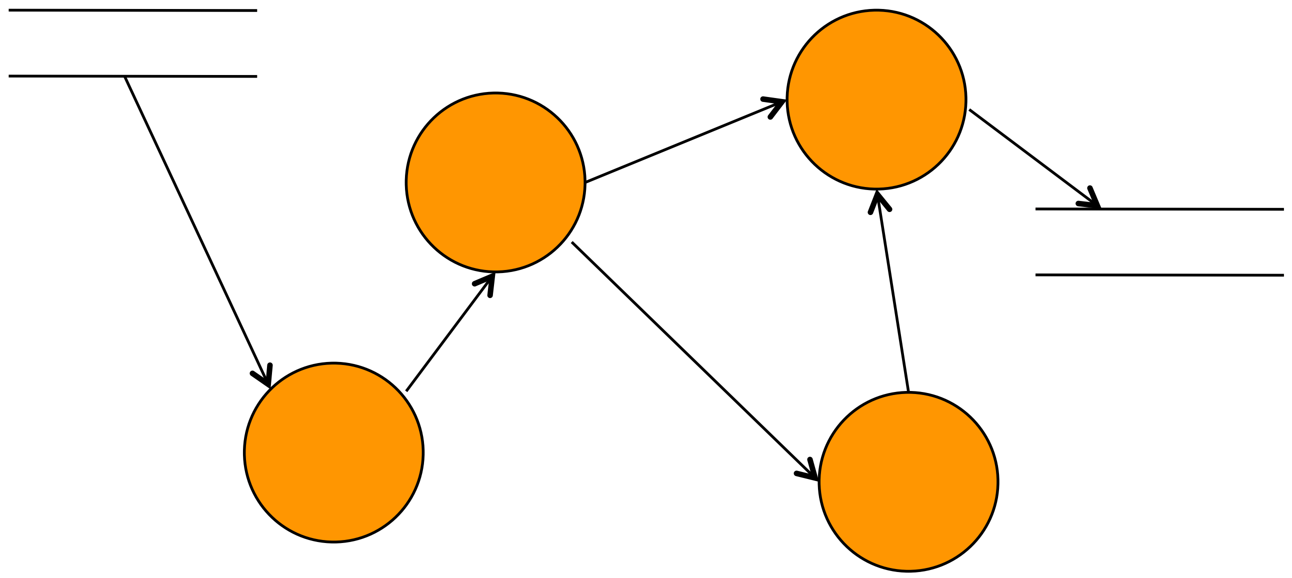
What follows are ideas that I learned very early on

- And still use day-by-day

Analysis

There used to be this thing called Systems Analysis

- It used to be a core skill
- But it got a bad name



Analysis

So, we stopped doing it

- Agile gave some of us an excuse

Analysis

We had to re-invent *understanding*

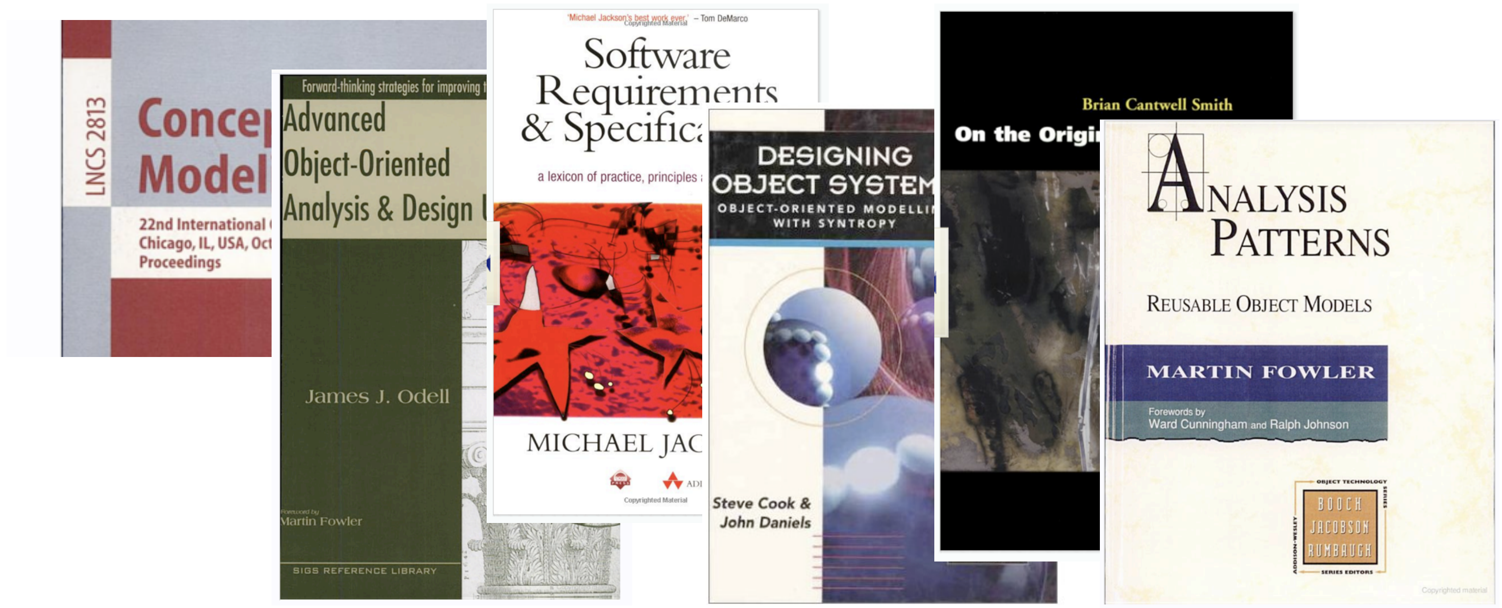
- behaviour driven development
  - (AKA TDD the way you were always supposed to do it)
- Domain Driven Design
  - “Until I started working in "enterprise IT" I didn't realise that people *didn't* do this. I suppose that this is an important book, but it's depressing that this is so”—Nat Pryce

Analysis

zühlke
empowering ideas

Domain Driven Design

"Leading software designers have recognized domain modeling and design as critical topics for at least 20 years, yet surprisingly little has been written about what needs to be done or how to do it."—Evans



Analysis

These days we "conquer and divide"

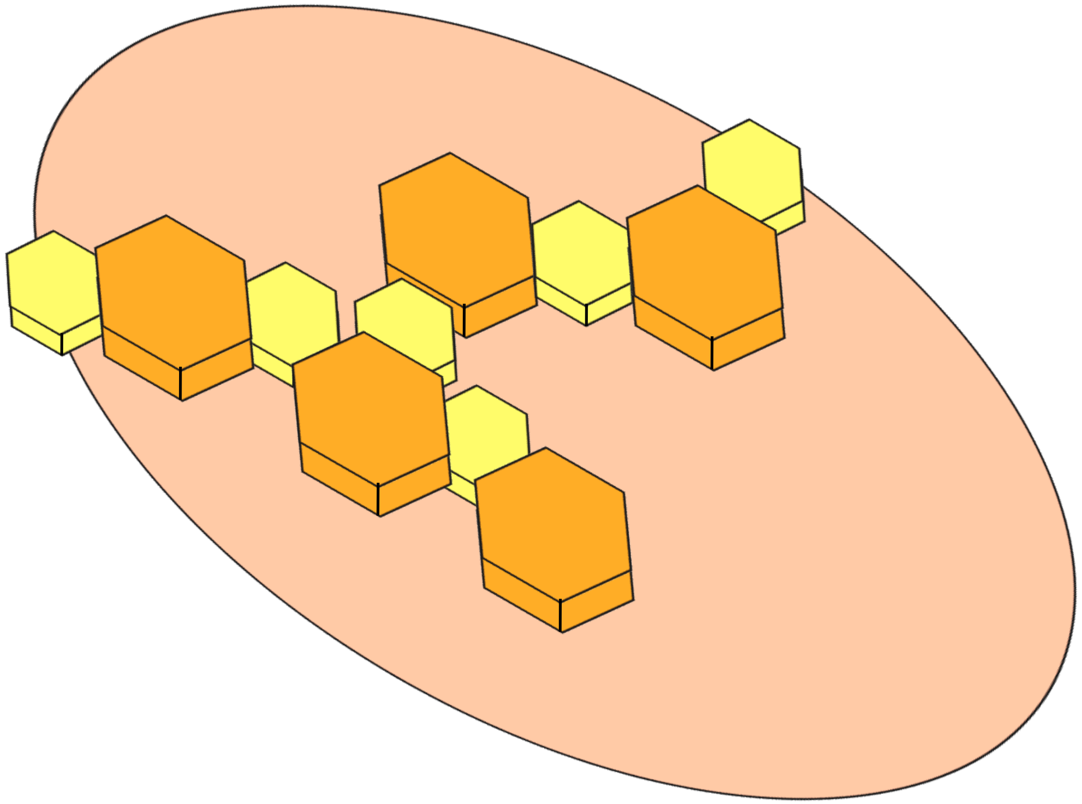
- We can discover the domain
- Which is great!

Analysis

But still…

- Objects in the world have states that they move between
  - We might want to talk explicitly about them
- Some events must occur in certain orders
  - We might want to talk explicitly about that

Consider a system built out of domains with various intents



Analysis

Then we would know where to put the analysis

There’s an echo of this in Enterprise stacks

Architecture

zühlke
empowering ideas

No-one quite agrees on what this is

So it has become everything and nothing

Architecture

Seems as if it should have something to do with:

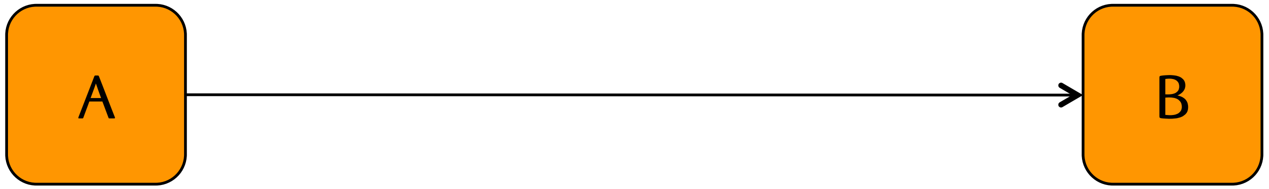
- Compromise
- Communication
- Habitability
- Reconciliation
- Comfort
- Ease of construction

And not:

- Which stack to fit in between the web server and database

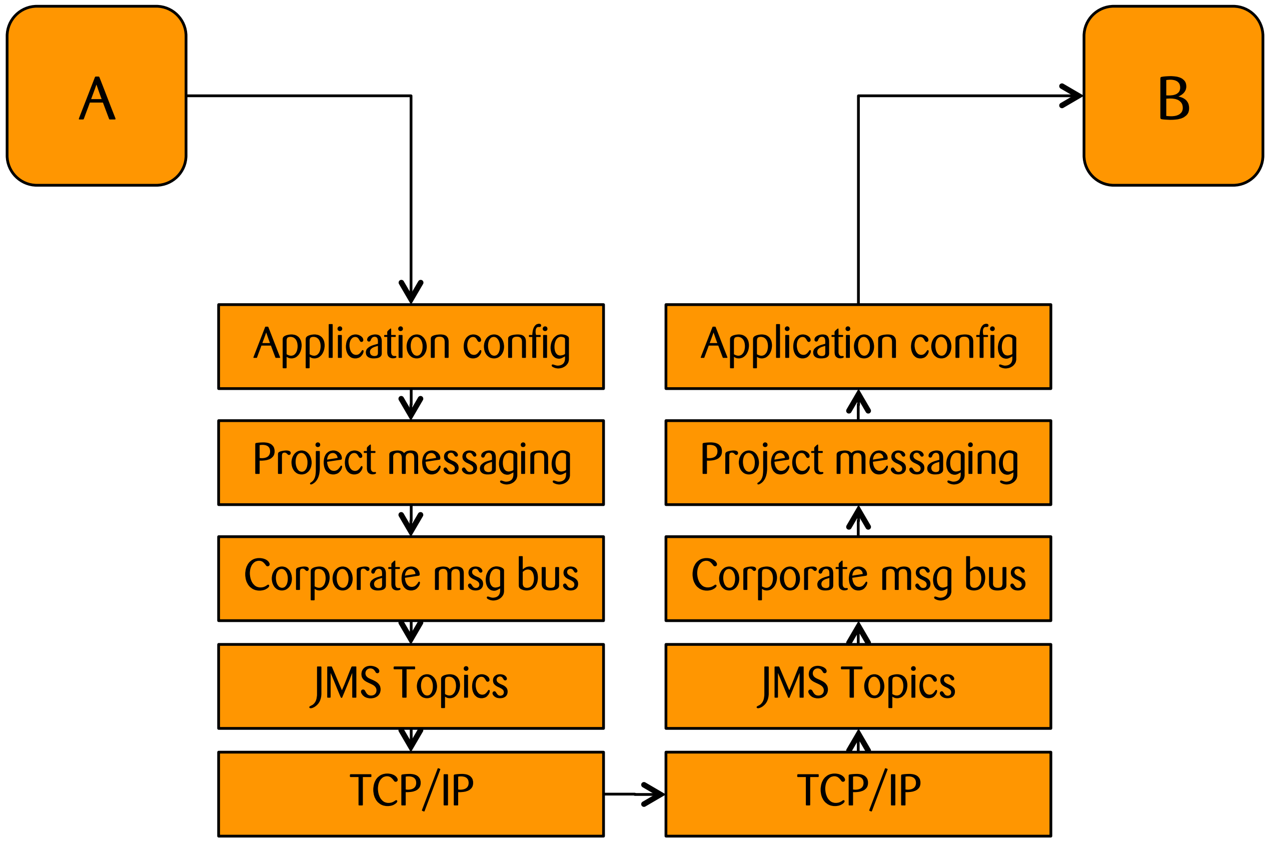
Architecture

“Stacks” an over-used metaphor

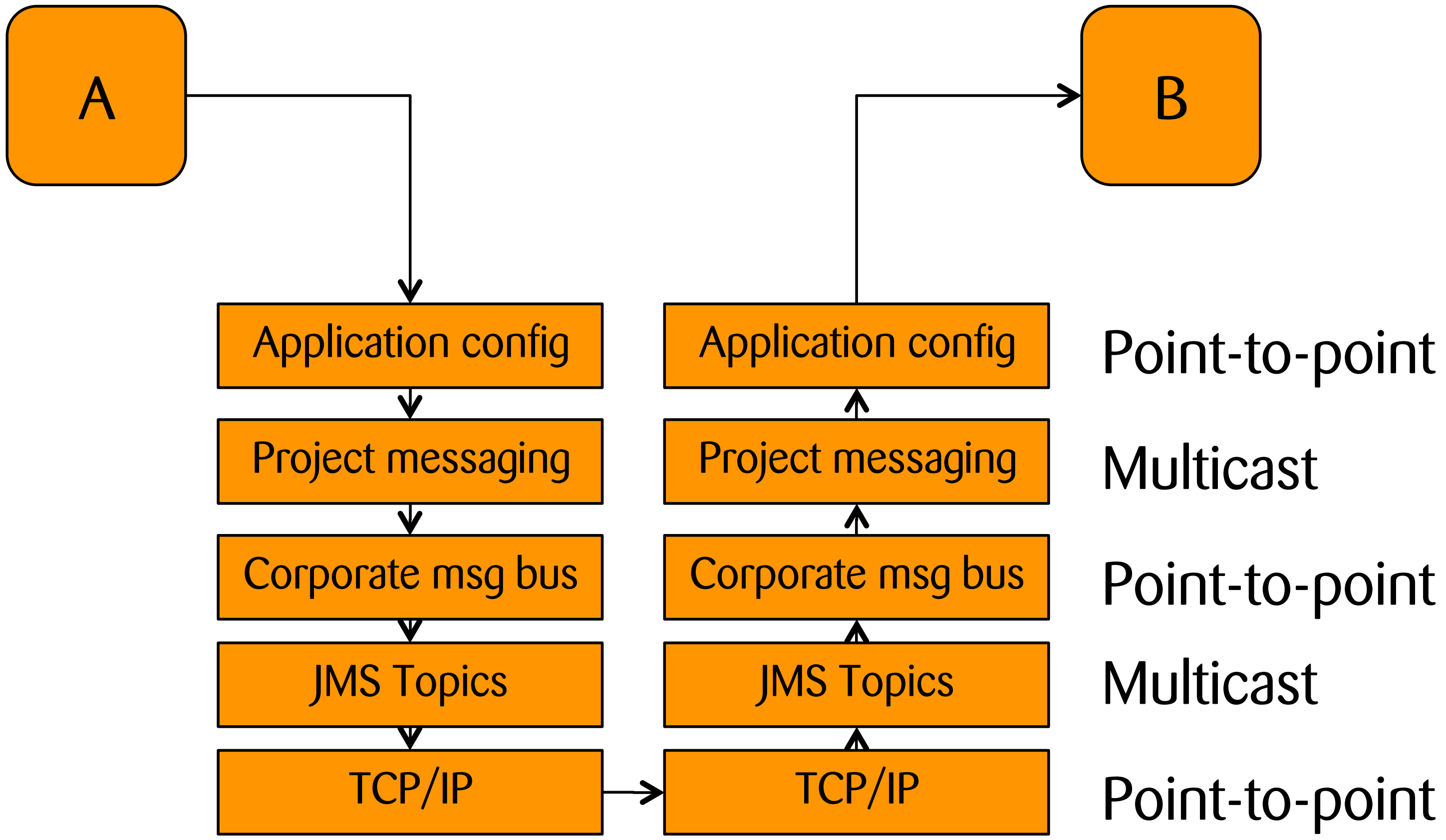


Architecture

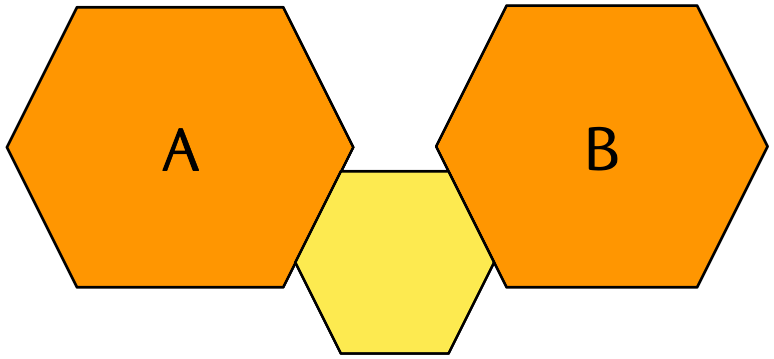
“Stacks” an over-used metaphor



“Stacks” an over-used metaphor

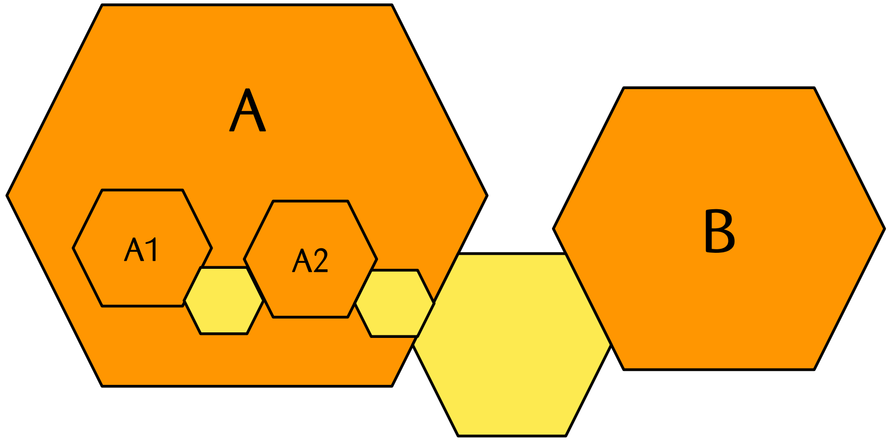


Architecture

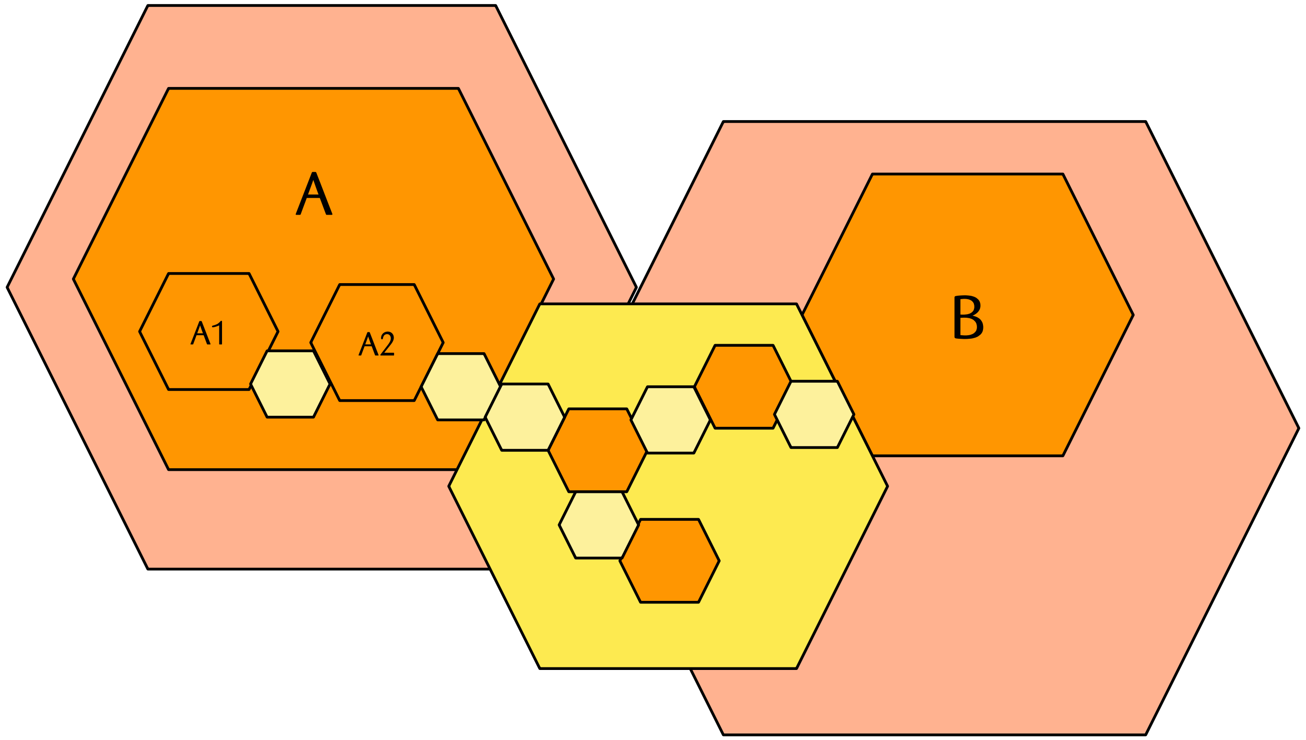


Architecture

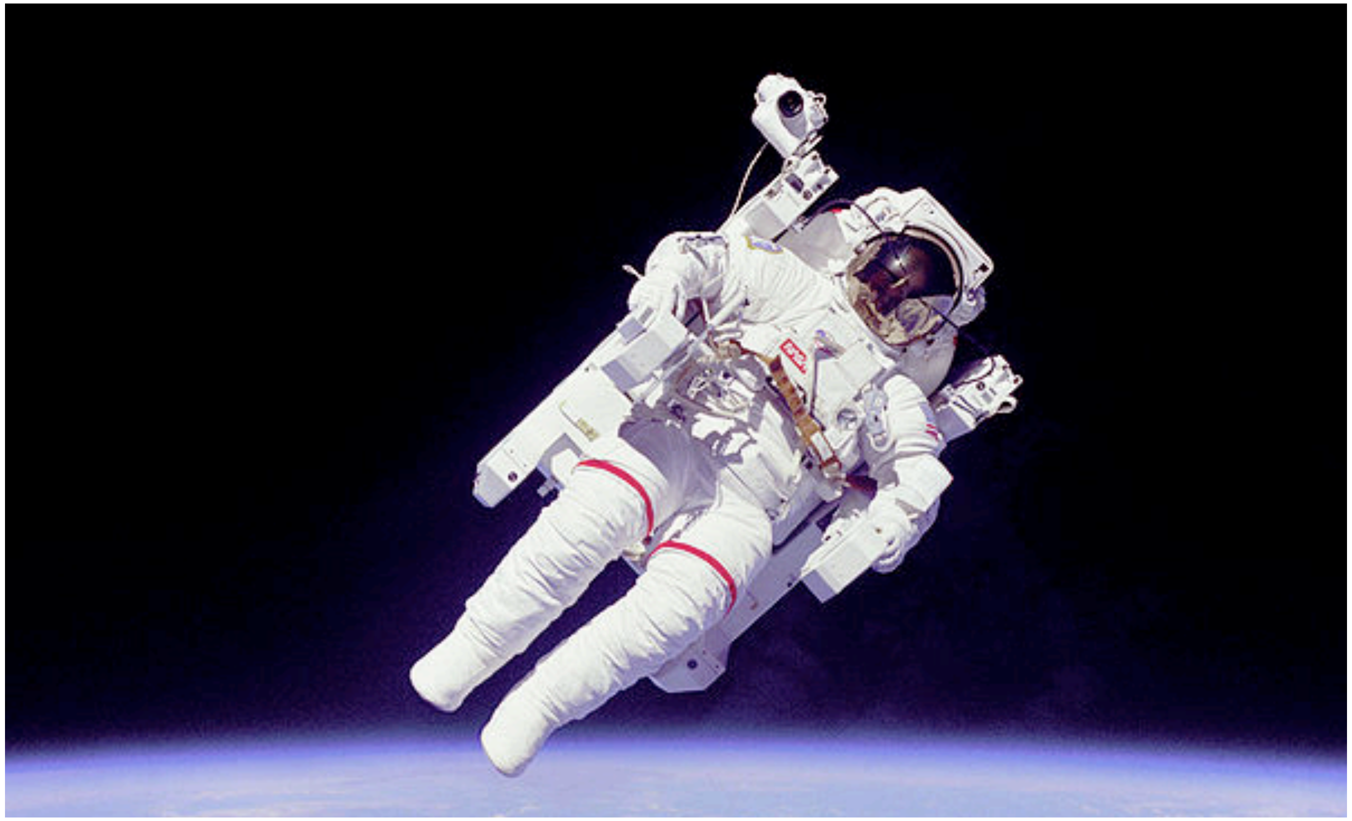
zühlke
empowering ideas



Architecture



Architecture got a bad name



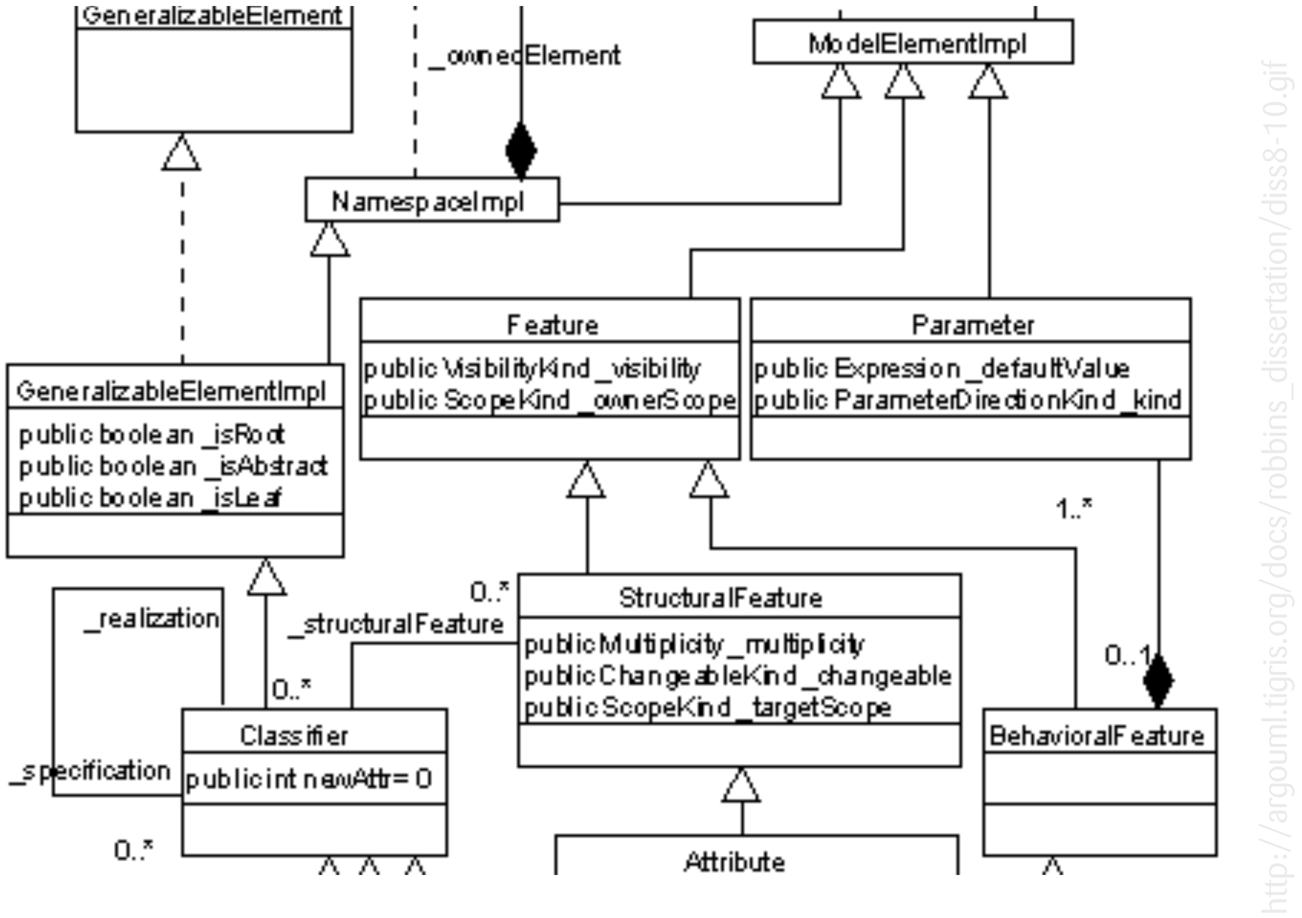
Architecture

Non–the–less, every system has an architecture

- It might be worth knowing how to talk about that

Modelling

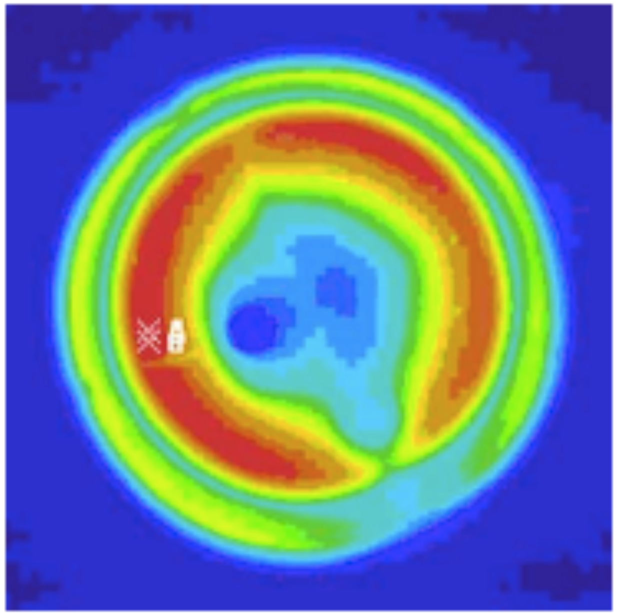
We really lost the plot on this one



GeneralizableElement
_ownedElement
ModelElementImpl
NamespaceImpl
Feature
public VisibilityKind _visibility
public ScopeKind _ownerScope
Parameter
public Expression _defaultValue
public ParameterDirectionKind _kind
GeneralizableElementImpl
public boolean _isRoot
public boolean _isAbstract
public boolean _isLeaf
1..*
0..*
_realization
_structuralFeature
StructuralFeature
public Multiplicity _multiplicity
public ChangeableKind _changeable
public ScopeKind _targetScope
0..1
0..*
Classifier
public int newAttr = 0
BehavioralFeature
_specification
0..*
Attribute
http://argouml.tigris.org/docs/robbins_dissertation/diss8-10.gif

Modelling

Engineers Model



Modelling

Engineers Model

- Models are useful for what they *leave out*
- Faster, cheaper than building a prototype

Models Answer Questions

- More quickly and easily than the real thing would

Modelling

Engineers Model

- Models are useful for what they *leave out*
- Faster, cheaper than building a prototype

Models Answer Questions

- More quickly and easily than the real thing would

Modelling

We got this wrong

- We tried to make out models useful by *adding* stuff
- Our models are often harder to build, and slower
- Out models too often don't answer questions

$STATE ::= patients \mid fields \mid setup \mid ready \mid beam\_on$

$EVENT ::= select\_patient \mid select\_field \mid enter \mid start \mid stop \mid ok \mid intlk$

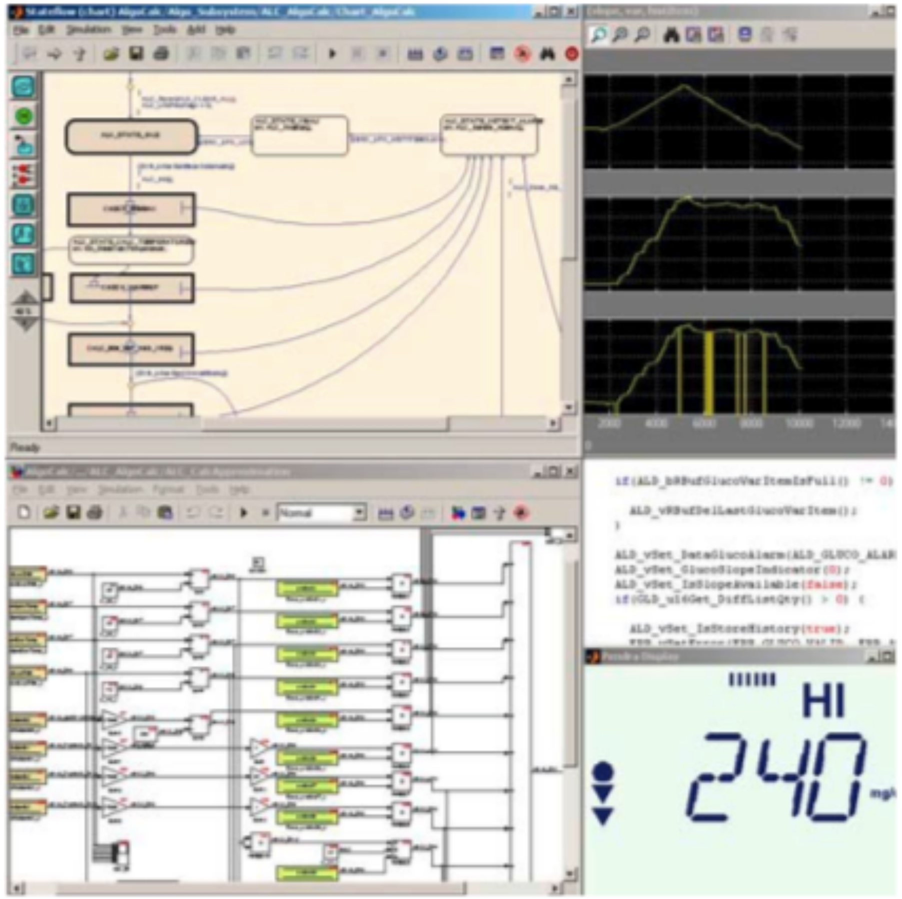
$FSM == (STATE \times EVENT) \nrightarrow STATE$

```
│ no_change, transitions, control: FSM
├──────────────────────────────────
│ control = no_change ⊕ transitions
│ no_change = { s: STATE; e: EVENT • (s, e) ↦ s }
│ transitions = { (patients, enter) ↦ fields,
│     (fields, select_patient) ↦ patients, (fields, enter) ↦ setup,
│     (setup, select_patient) ↦ patients, (setup, select_field) ↦ fields, (setup, ok) ↦ ready,
│     (ready, select_patient) ↦ patients, (ready, select_field) ↦ fields, (ready, start) ↦ beam_on, (ready, intlk) ↦ setup,
│     (beam_on, stop) ↦ ready, (beam_on, intlk) ↦ setup }
```

Modelling

zühlke
empowering ideas

There is a way forward, when appropriate



Modelling

But life, sadly, turns out to be too short

Conclusion

These *were* good ideas

- They still are good ideas

We turned against them because they were misapplied

- We can do better than that