Abstraction – is it teachable?

or

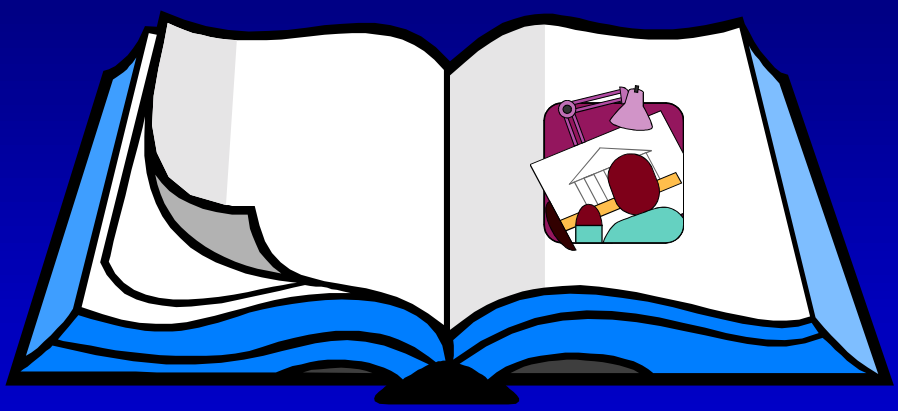
the devil is in the detail

Jeff Kramer

Distributed Software Engineering
Department of Computing

Imperial College
London

Chapter 1. Teaching experience



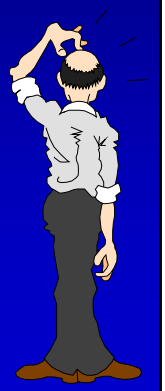
Teaching experience

Courses:

- software engineering,
- distributed systems,
- distributed algorithms,
- programming,
- concurrency,
- .....

Skills:

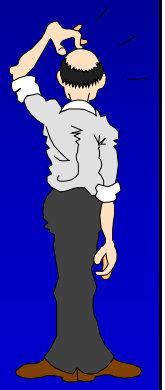
- problem solving,
- conceptualization,
- modelling,
- analysis,
- .....



Experience: the better ones....

Some students are able to produce elegant designs and solutions.

Generally the same students are also able to comprehend the complexities of distributed algorithms, the applicability of the various modelling notations, and so on.

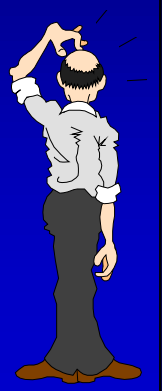


Experience: the others ....

A number of others are not so able.

They tend to find distributed algorithms very difficult, do not appreciate the utility of modelling, find it difficult to know what is important in a problem, produce convoluted solutions which replicate the problem complexities, ...

Why ?



I believe .....

... that the heart of the problem lies in a difficulty in dealing with

Abstraction

Chapter 2. What is it? Why is it so important?

Definitions

- the act of withdrawing or removing something
- the act or process of leaving out of consideration one or more properties of a complex object so as to attend to others

=> Remove detail (simplify and focus)

- a general concept formed by extracting common features from specific examples
- the process of formulating general concepts by abstracting common properties of instances

=> generalisation (core or essence)

Abstraction in other disciplines

Art – Matisse

Music – jazz

Maps – London Underground map

Matisse – guess what ….

representation
of the essence
of the subject
&
removal of
detail

Jazz

jazz musician –

"It is easy to make something simple sound complex, however its more difficult to make something complex sound simple".

1930 – London Underground map

Aspect of focus?

Relationship between stations and interchanges, not actual distances

1933 – Harry Beck (1st schematic image map)

2001 – Fit for purpose

Why is abstraction important in **Software Engineering**?

Software is abstract!

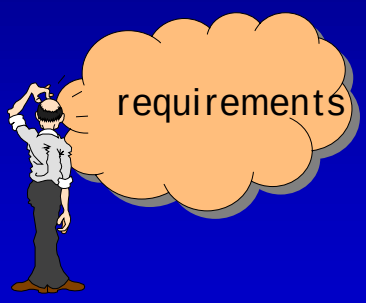
Requirements

Design

Programming

Why is it important? **requirements engineering**

Requirements - elicit the critical aspects of the environment and required system while neglecting the irrelevant.



goals
scenarios
assumptions
constraints
properties

"The act/process of leaving out of consideration one or more properties of a complex object so as to attend to others"

Why is it important? **design**

Design - articulate the software architecture and component functionalities which satisfy functional and non-functional requirements while avoiding unnecessary implementation constraints.

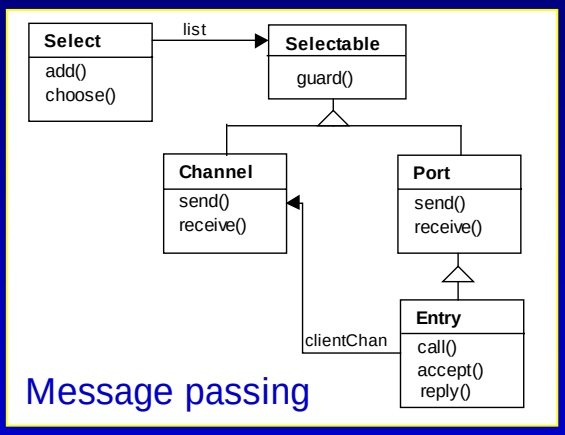
eg. Compiler design (Ghezzi):

- *abstract syntax* to focus on essential features of language constructs;
- design to generate intermediate code for an *abstract machine*

"The act/process of leaving out of consideration one or more properties of a complex object so as to attend to others"

Why is it important? **programming**

Programming - use data abstraction and classes so as to generalize solutions.



"the process of formulating general concepts by abstracting common properties of instances "

Why is it important? advanced topics

Abstract interpretation for program analysis - map concrete domain to an abstract domain which captures the semantics for the purpose at hand.

eg. Rule of signs for multiplication *

0*+ = 0*- = +*0 = -*0 = 0

+*+ = -*- = +

+*- = -*+ = -

Hankin

"the process of formulating general concepts by abstracting common properties of instances"

Abstraction

Abstraction is fundamental to Engineering in general, and to Software Engineering in particular !

Do our students' powers of abstraction depend on their **genes** ?

Can we improve their abilities ? …and if so, how ?

Is it possible to teach abstraction ?

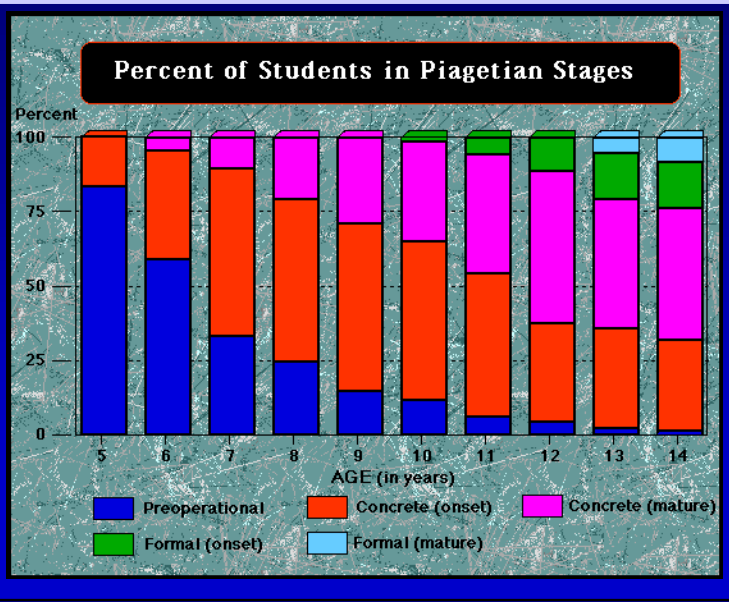
Cognitive Development

Changes in thinking by which mental processes become more complex and sophisticated.

Jean Piaget's fours stages of cognitive development:

- 1st & 2nd: sensorimotor and preoperational (0-7yrs)
- 3rd stage: concrete operational thought (7-12yrs)
  - no abstract thought
- 4th stage: formal operational period (12–adult)
  - think abstractly (logical use of symbols related to abstract concepts), systematically and hypothetically

Cognitive Development – formal operational thought



4
3
3

Huitt & Hummel

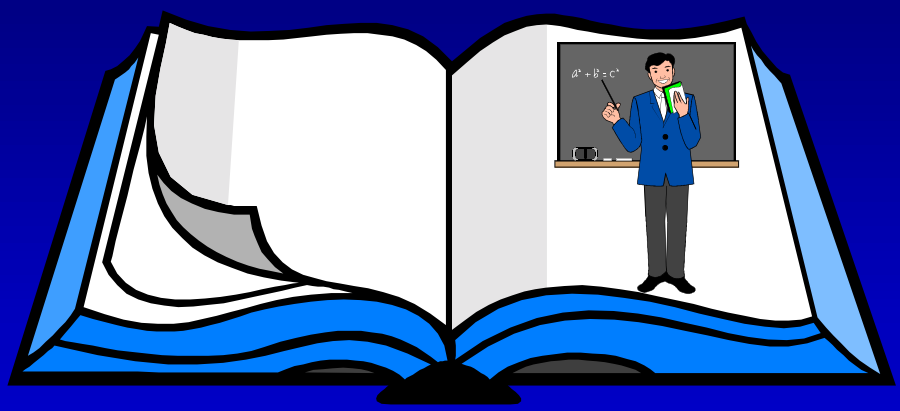
Cognitive Development

Changes in thinking by which mental processes become more complex and sophisticated.

Jean Piaget's fours stages of cognitive development:

- 1st & 2nd: sensorimotor and preoperational (0-7yrs)
- 3rd stage: concrete operational thought (7-12yrs)
  - good news: Some ability for abstraction with training
- 4th stage: formal operational period (12–adult)
  - bad news: Not reached by all individuals. Only 30% to 40% of teenagers exhibit ability for abstact thought, some adults never do!

Chapter 3. Teaching abstraction?



Courses on Abstraction?

1st Year (all required):

Declarative Programming I
Databases 1
Declarative Programming II
Discrete Mathematics 1
Discrete Mathematics 2
Hardware
Programming I
Logic
Reasoning about Programs
Programming II
Computer Systems
Mathematical Methods and Graphics

2nd Year (most required):

Algorithms, Complexity and Computability
Architecture II
Compilers
Artificial Intelligence I (optional)
Operating Systems II
Computational Techniques (optional)
Software Engineering - Design I
Concurrent Programming (optional)
Statistics
Networks and Communications
Software Engineering - Design II

Imperial College MEng in Software Engineering

Courses on Abstraction?

3rd Year (most optional):

Advanced Databases
Advanced Computer Architecture
Advances in Artificial Intelligence
Computational Finance
Computational Logic
Custom Computing
Distributed Systems
Introduction to Bioinformatics
Knowledge Management Techniques
Decision Analysis
Operations Research
Graphics
Quantum Computing
Management - Organisation and Finance (required)
Simulation and Modelling
Multimedia Systems
Software Engineering - Methods (required)
Performance Analysis
The Practice of Logic Programming
Robotics
Type Systems for Programming Languages

4th Year (most optional):

Advances in Artificial Intelligence
Advanced Graphics and Visualization
Advanced Issues in Object Oriented Programming Automated Reasoning
Advanced Operations Research
Complexity
Computer Vision
Computing for Optimal Decisions
Intelligent Data and Probabilistic Inference
Domain Theory and Exact Computation
Modal and Temporal Logic
Grid Computing
Models of Concurrent Computation
Knowledge Representation
Natural Language Processing
Management - Economics and Law
Network Security
Multi-agent Systems
Program Analysis
Parallel Algorithms
Software Engineering - Environments

Courses on Abstraction?

Which courses rely on or utilise the powers of abstraction to

- explain
- model
- specify
- reason
- solve .... ?

List of courses which do NOT make use of Abstraction?

So ....?

Abstraction is essential but has to be taught indirectly.

So ....?

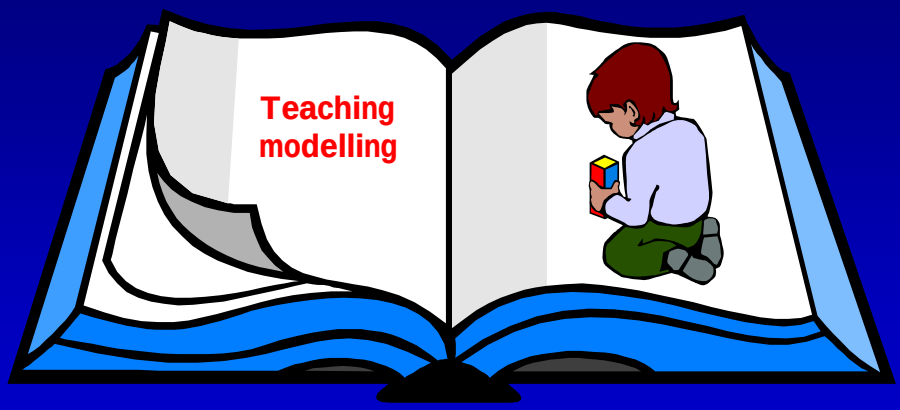
How should we ensure that students can understand and make use of abstraction ?

1. Teach enough Mathematics
2. Teach (formal) modelling and analysis

   Caveat: Must be tool supported

   Must feel the benefit
3. Other techniques ?

Chapter 4. Modelling and analysis



Models and Modelling?

- A model is a description from which detail has been removed in a systematic manner and for a particular purpose.
- A simplification of reality intended to promote understanding.
- Models are the most important engineering tool; they allow us to understand and analyse large and complex problems.

Finkelstein

Ockam's Razor

- William of Ockam (1285) formulated the famous "Rule of the Razor":

  Entia non sunt multiplicanda sine necessitate.

  Entities should not be multiplied without necessity.

- In other words a model should be as simple as possible, but no simpler - it should discard elements of no interest.
- "Fit for purpose".

formal methods experience

Attempts to teach software engineering students how to use formal models as part of their software development process have generally been unsuccessful.

The models often

- **do not integrate well into the software development process,**
- **are too difficult to learn and use,**
- **provide inadequate tool support**
- **are not well motivated**

Much pain with little gain!

software engineering students

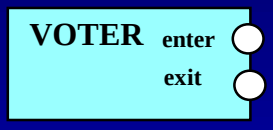
The challenge is to make modelling and analysis accessible and useful to software engineering students.

teaching Concurrency – models and programs

- Concepts
  - we use a model-based approach for the design and construction of concurrent programs
- Models
  - we use finite state models to represent concurrent behaviour (FSP and LTS), and model checking for analysis (LTSA).
- Practice
  - we use Java for constructing concurrent programs.

component VOTER - behaviour

Component:



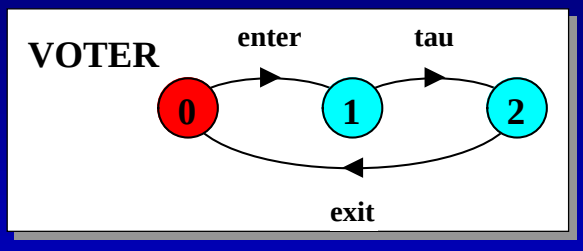
Process specification in FSP:

```
VOTER = (enter -> vote -> exit -> VOTER
        ) @{enter,exit}.
```

Actions {enter,exit} are exposed, vote is hidden.

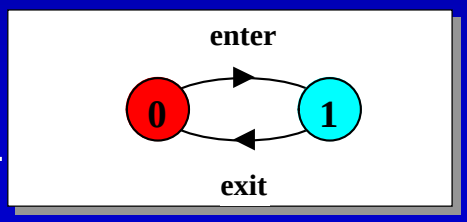
component USER - behaviour

Labelled transition system LTS:



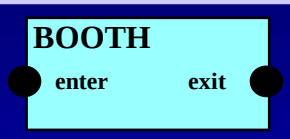
LTS Animation can be used to step through the actions to test specific scenarios.

VOTER can be minimised with respect to Milner's observational equivalence.



component BOOTH - behaviour

Component:



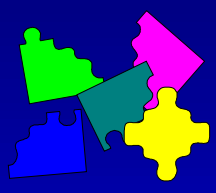
Process specification in FSP:

```
const Max = 3
range Int = 0..Max

BOOTH(N=Max) = BOOTH[N],
BOOTH[v:Int] =(when(v>0) enter->BOOTH[v-1]
              |when(v<Max)exit->BOOTH[v+1]
              ).
```


Modelling concurrent systems

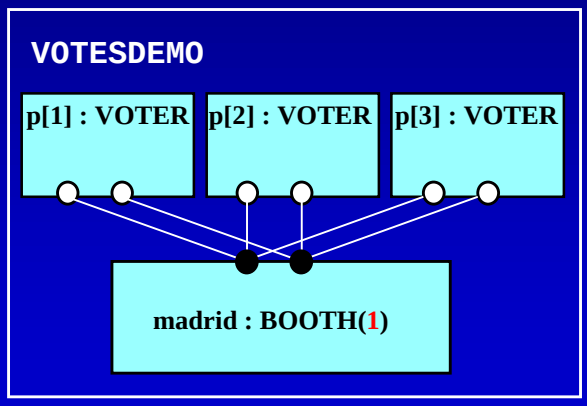
Primitive components



Composite components

Composite component behaviour

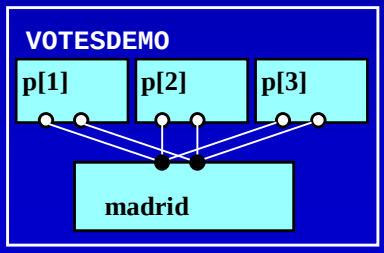
Three voters **p[1..3]** use a shared booth, **madrid,** to register their vote. To ensure mutual exclusion .....



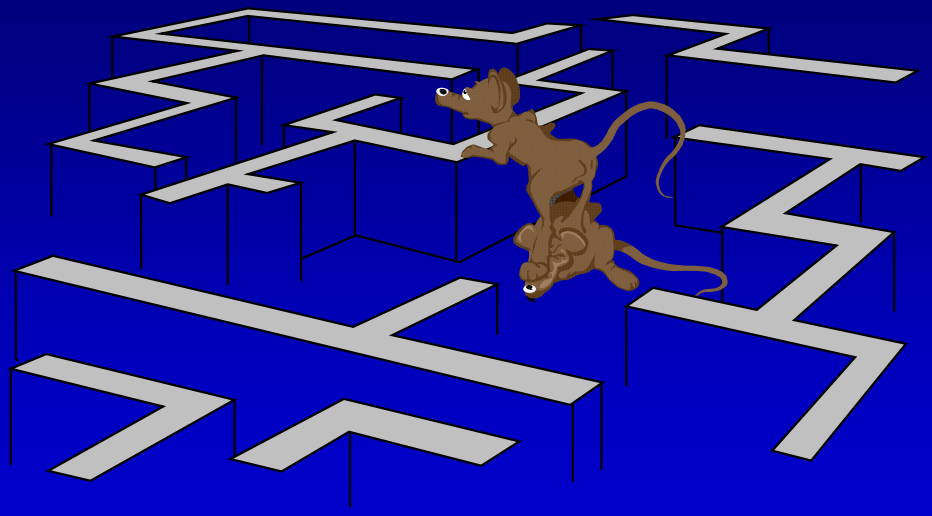
*... the number of spaces available in the booth must be **1**.*

Composite component behaviour

```
||VOTESDEMO = (  p[1..Max]:VOTER
              || madrid:BOOTH(1)
              )
      /{p[1..Max].enter/ madrid.enter,
        p[1..Max].exit/ madrid.exit}.
```

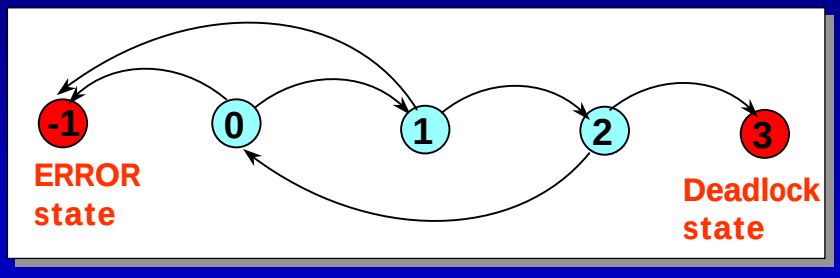


Benefit - behaviour analysis



Reachability analysis for checking models

Searches the system state space for **deadlock** states and **ERROR** states arising from property violations.



Deadlock - state with no outgoing transitions.

ERROR (π) state -1 is a trap state. Undefined transitions automatically mapped to the **ERROR** state.

Safety - property automata

Safety properties are specified by deterministic finite state processes called property automata. Invalid behaviour transitions to an **ERROR**.

```
property EXCLUSION =  (p[i:1..3].enter
                    -> p[i].exit
                    -> EXCLUSION ).

||CHECK = (VOTESDEMO || EXCLUSION).
```

Safety properties are are violated if ***ERROR*** is reachable in the composed system.

...if the initial value of the booth is 2 ? ...or 0?

Liveness - progress properties

We support a limited class of liveness properties, called progress, which can be checked efficiently:

$$[]\Diamond a$$

$$[]\Diamond a \Rightarrow []\Diamond b$$

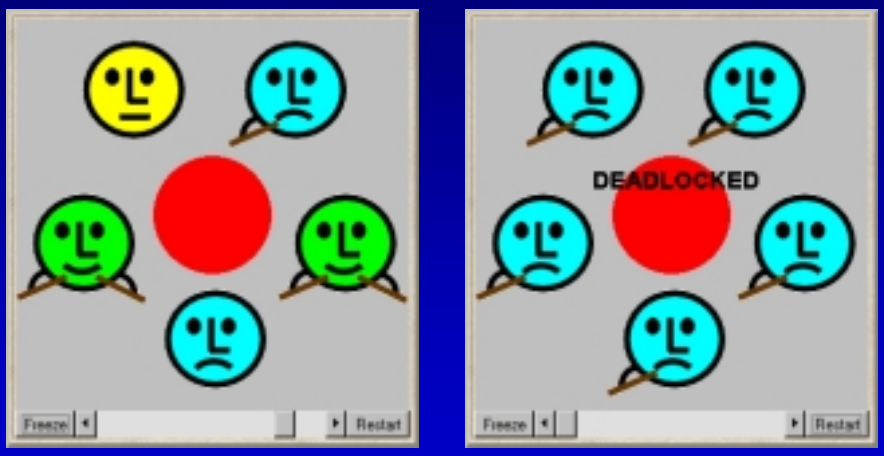
i.e. Progress properties check that, in an infinite execution, particular actions occur infinitely often.

For example:

```
progress OKtoVOTE[i:1..3] = {p[i].enter}
```

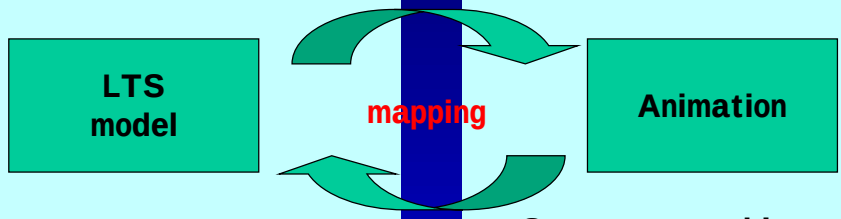
...if we give priority to two of the voters?

Deadlock – analysis Vs intuition



Dining Philosophers

Model interpretation ⟺ animations

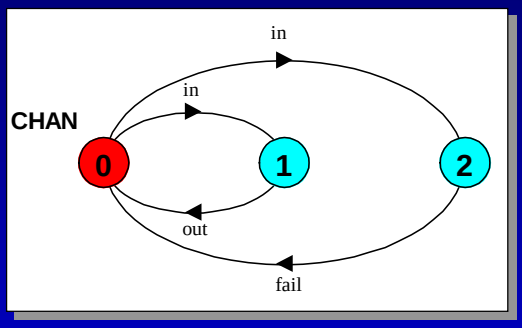


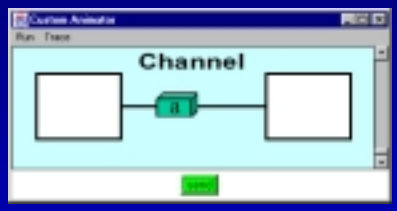
LTS Model checking

- safety properties
- progress properties
- compositional reachability
- abstraction & minimisation

Separate graphic animation model which preserves the behaviour of the model and has sound semantics based on Timed Automata.

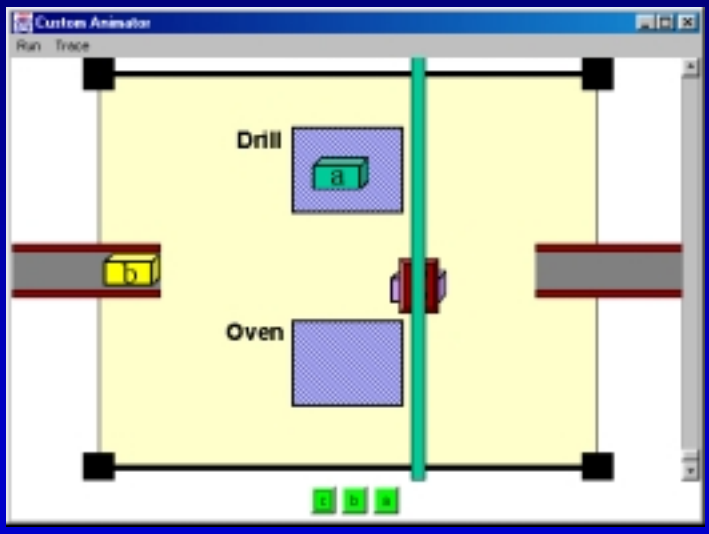
abstract models ⟺ concrete animations





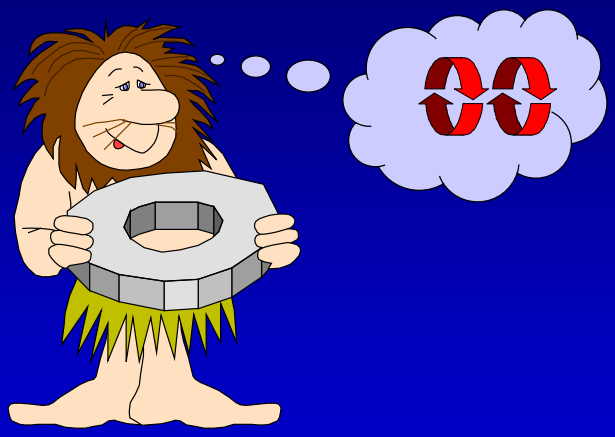

```
CHAN = (in -> out  -> CHAN
       |in -> fail -> CHAN
       ).
```


Flexible Manufacturing Cell



Animated models can be composed to form complex models.

Model based design of concurrent programs



http://www-dse.doc.ic.ac.uk/concurrency/

from requirements to models

Requirements

- goals of the system
- scenarios (Use Case models)
- properties of interest

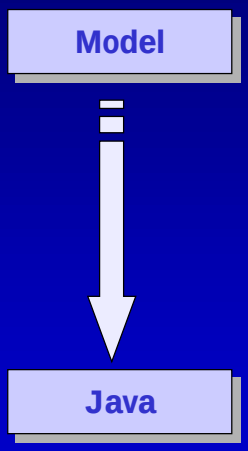
Any appropriate design approach can be used.

- identify the main events, actions, and interactions
- identify and define the main processes
- identify and define the properties of interest
- structure the processes into an **architecture**

Model

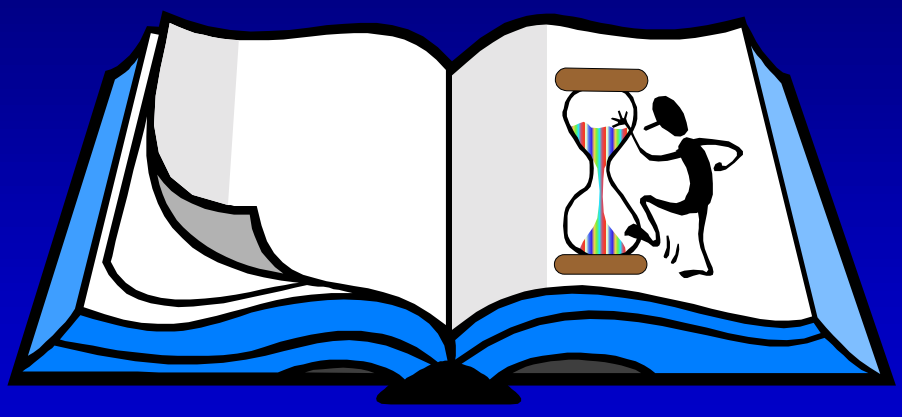
- check traces of interest
- check properties of interest

from models to implementations



- identify the main active entities
  - to be implemented as threads
- identify the main (shared) passive entities
  - to be implemented as monitors
- identify the interactive display environment
  - to be implemented as associated classes
- structure the classes as a class diagram

Chapter 5. Conclusions ...



How ....?

How can we teach abstraction ?

1. Teach enough Mathematics
2. Teach (formal) modelling and analysis

   Caveat: Must be tool supported

   Must feel the benefit
3. Other techniques ?

Experience .....

- Generally very good - the students find the models relatively intuitive and helpful in clarifying the problem.
- Comprehension is facilitated by model animation, model checking and simulation.

- **However** – some still seem to find constructing models themselves, ab initio, to be very difficult!

Modelling

- It is not enough to think about what they want to model, they need to think about how they are going to use that model.
- … fit for purpose (Occam's Razor)

Other techniques?

Learn from Cognitive Development ?

➨ Give students an opportunity to explore many hypothetical questions

➨ Encourage students to explain how they solve problems.

➨ Whenever possible, teach broad concepts, not just facts, using materials and ideas relevant to the students.

Other techniques?

More emphasis on …

- Active learning - key to development and learning is activity
- Social context of learning - learning climate, community's expectation, teachers' perceptions, …

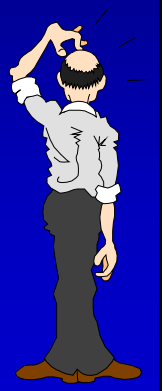
Teach respect for Clarity and Simplicity

"It has been my experience with literary critics and academics in this country, that clarity looks a lot like laziness and ignorance and childishness and cheapness to them. Any idea which can be grasped immediately is for them, by definition, something they knew all the time."

Kurt Vonnegut

I believe that …

- Abstraction is fundamental to Software Engineering.
- Abstraction has to be taught indirectly.
- Students who can understand, appreciate and utilise abstraction produce the most elegant models.



Abstraction – is it Teachable?

If the devil is in the detail, perhaps salvation is in Abstraction ?!