

What Is A Good Model?

Dr J Paul Gibson

Dept. INF

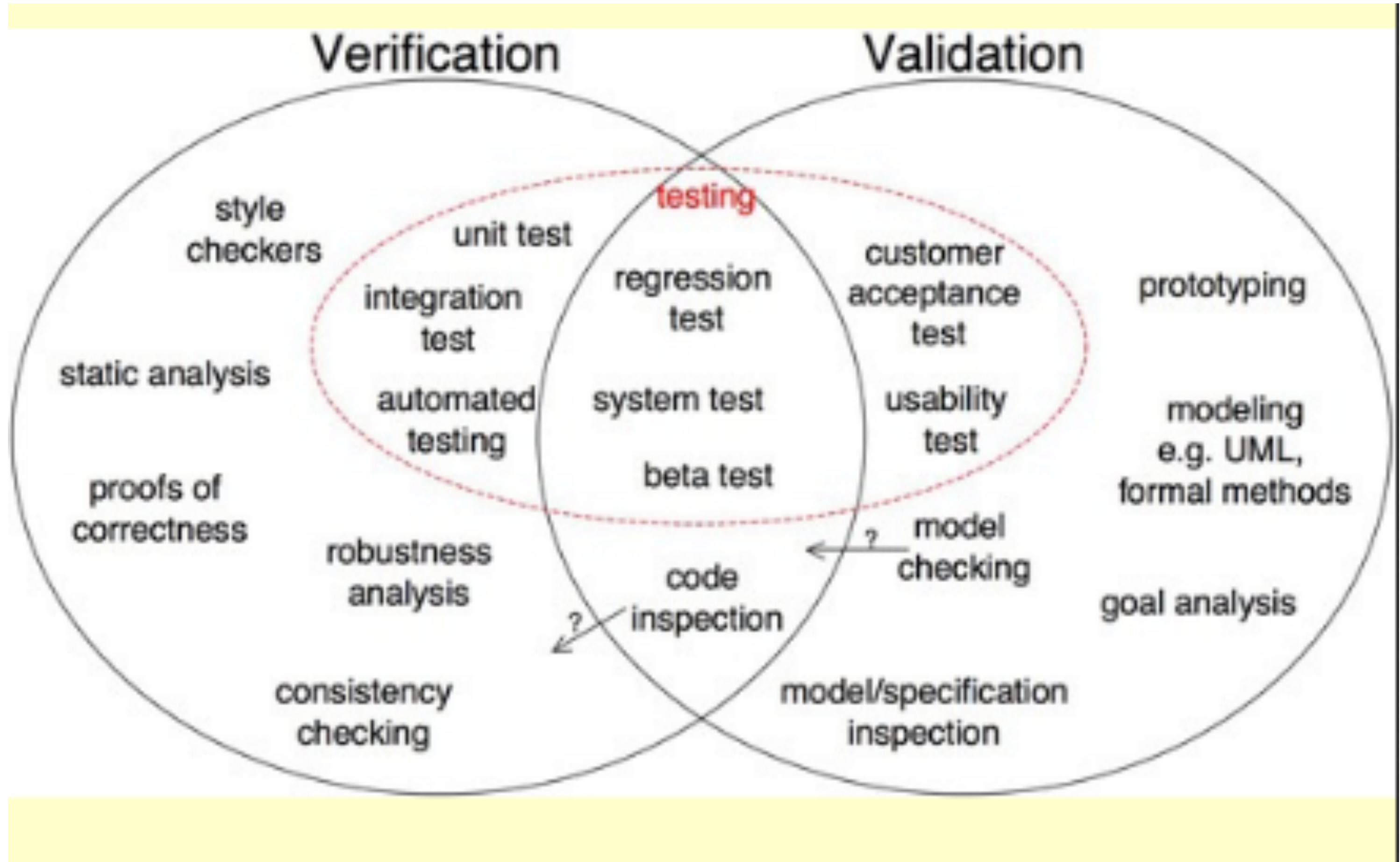
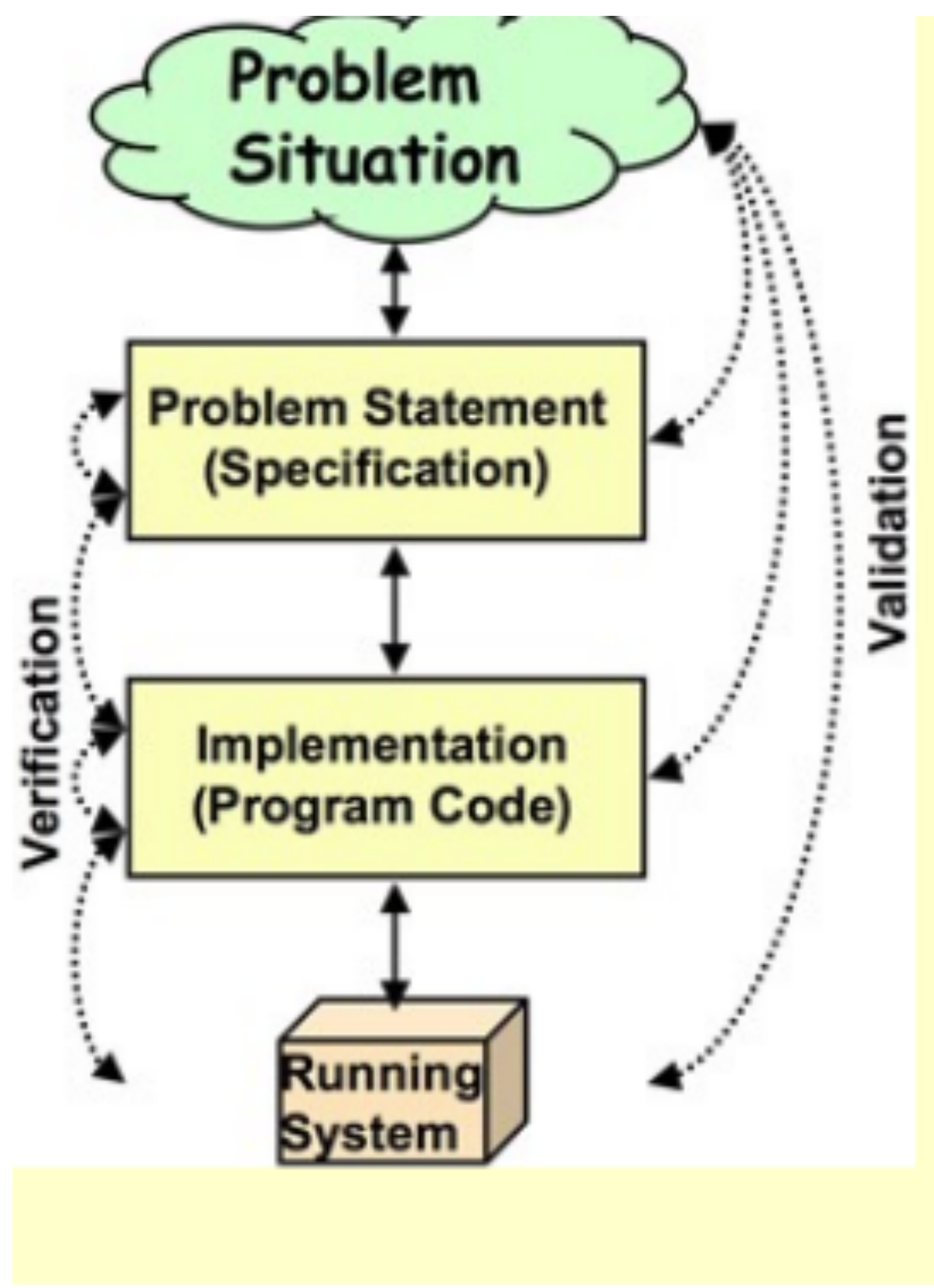
Office D311

paul.gibson@telecom-sudparis.eu

<http://jpaulgibson.synology.me/~jpaulgibson/TSP/Teaching/CSC4104/CSC4104-WhatIsAGoodModel.pdf>

NOTE - All work will be with paper/pen and interactively on the whiteboard

Engineers SOLVE *PROBLEMS*

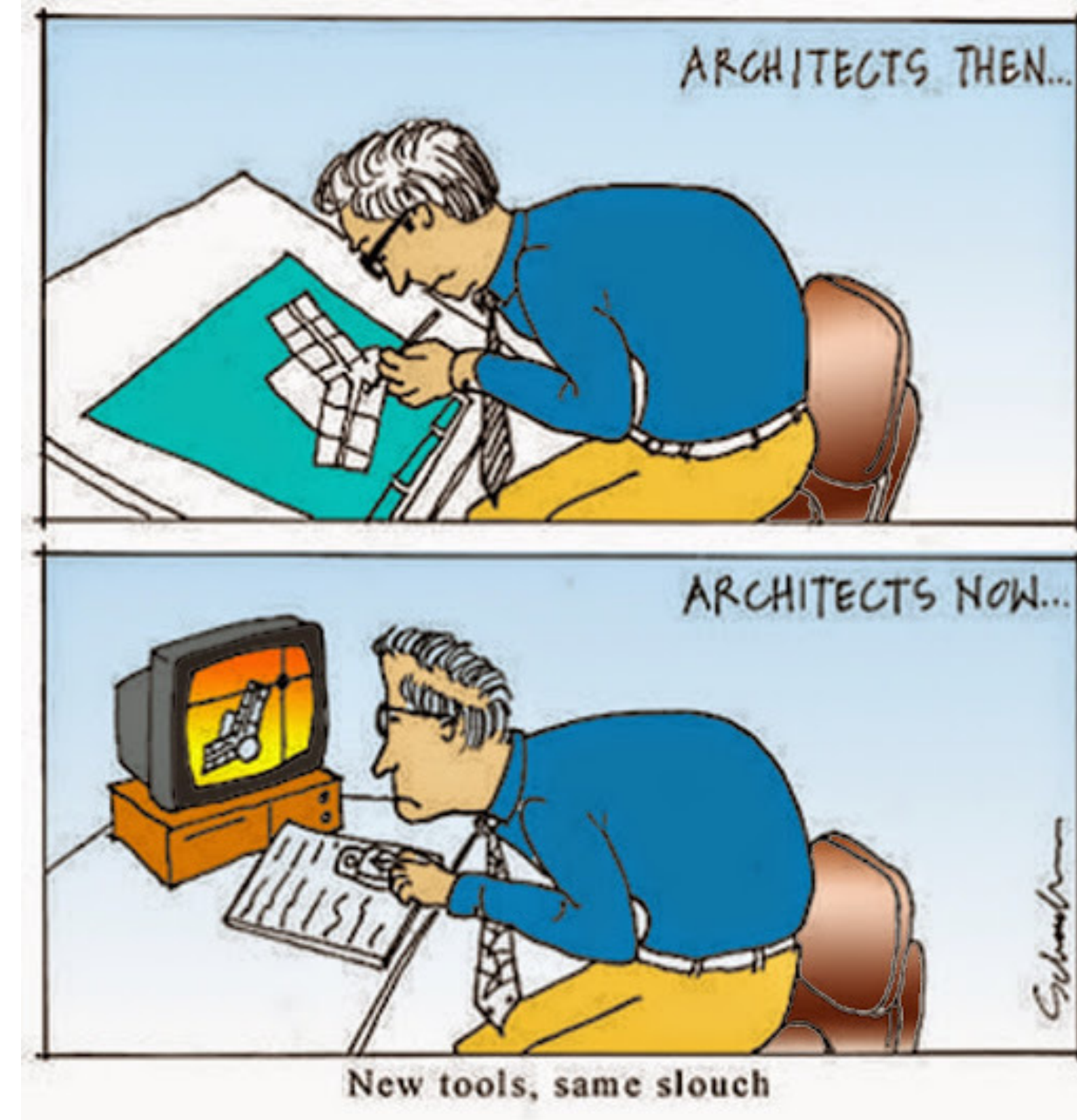


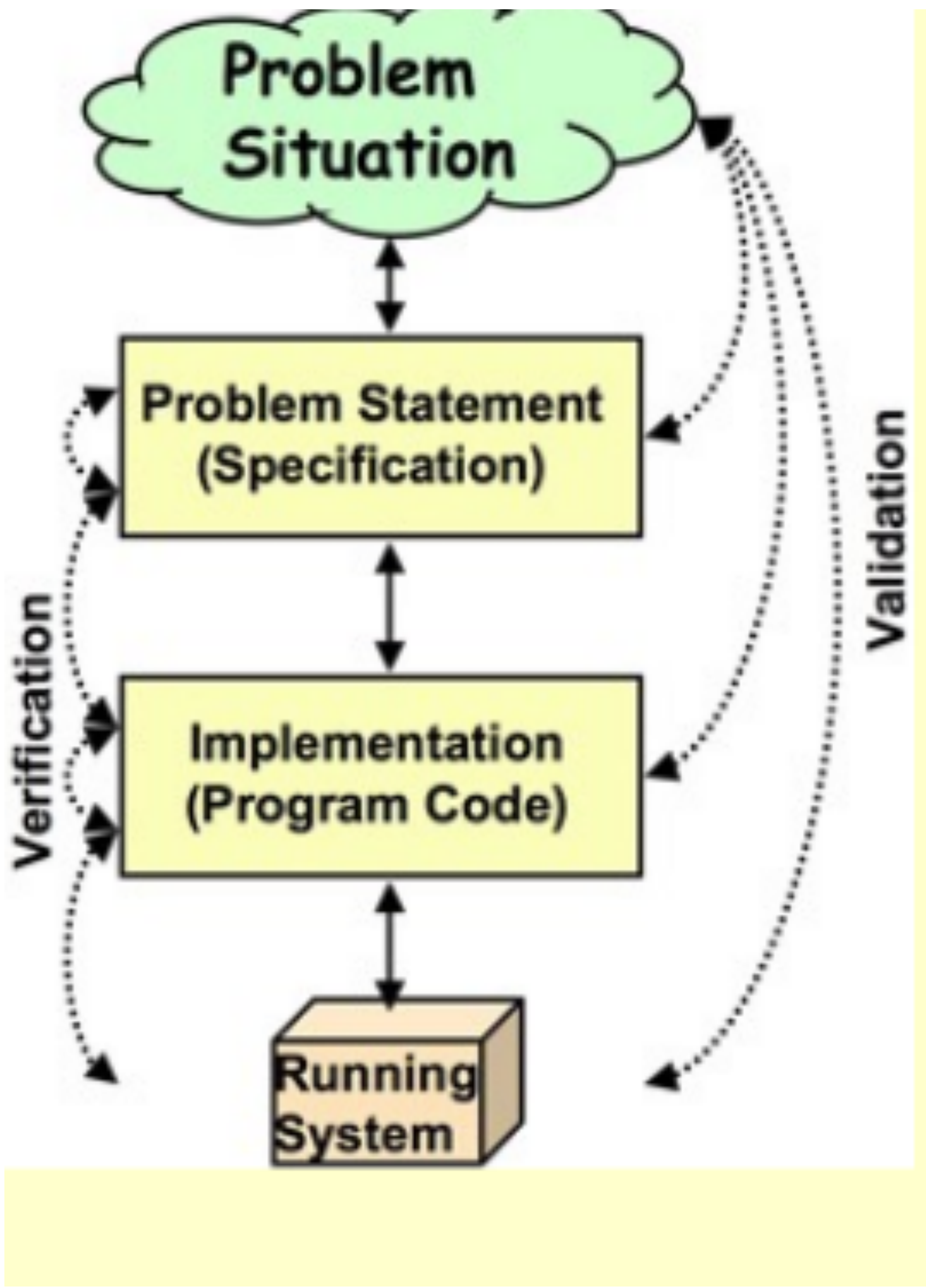
and CHECK (proposed) *SOLUTIONS*

Engineers work with models (of problems and solutions)

Engineering is based on science – **scientists** (try to) **build models** of things in the real world, **engineers** (try to) build **things** in the real world from models

Architects build models of problems and solutions – they are engineers and scientists





Build a Requirements Model

Build a Design Model

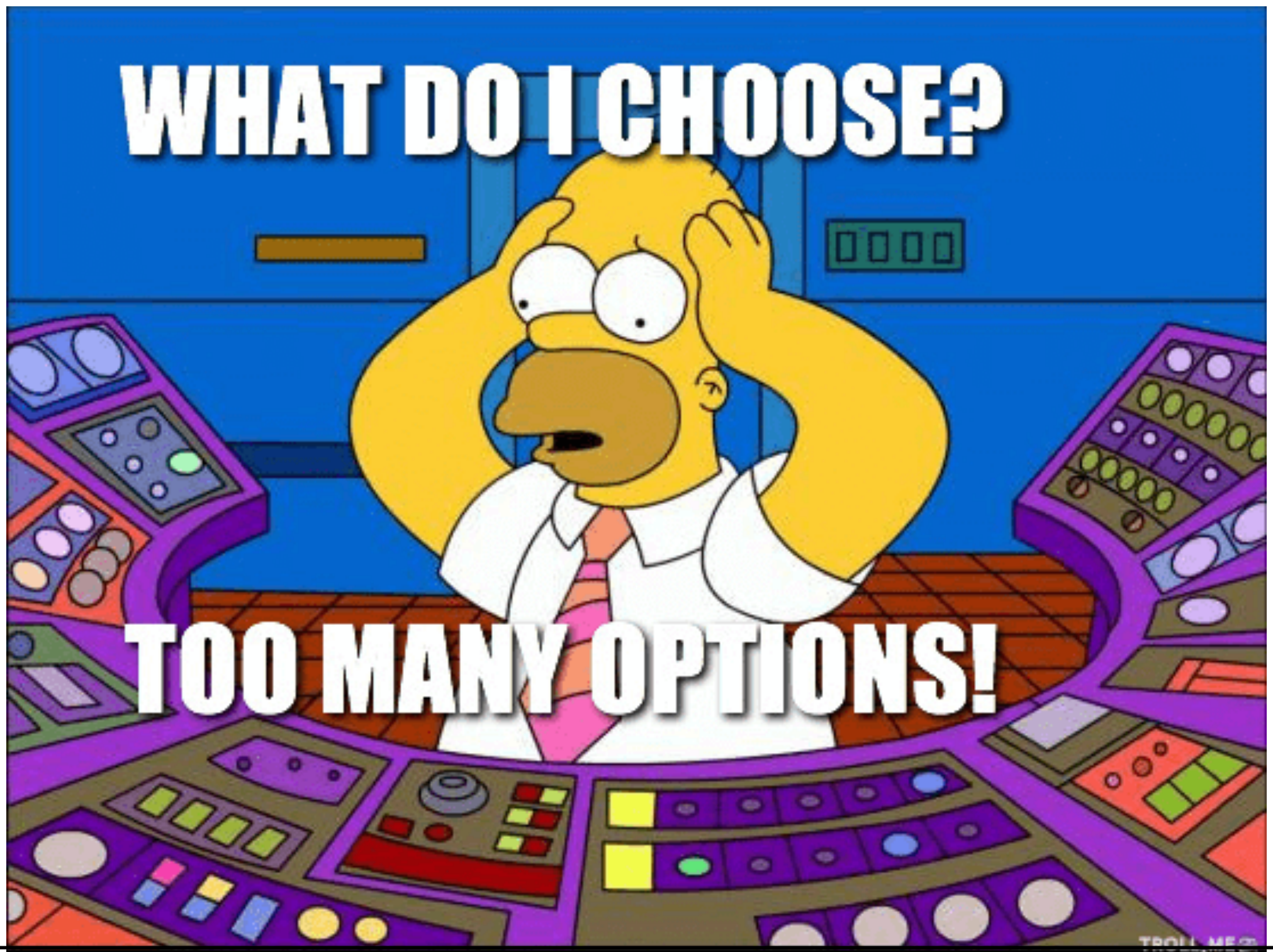
Build an Implementation Model

Build Test Model

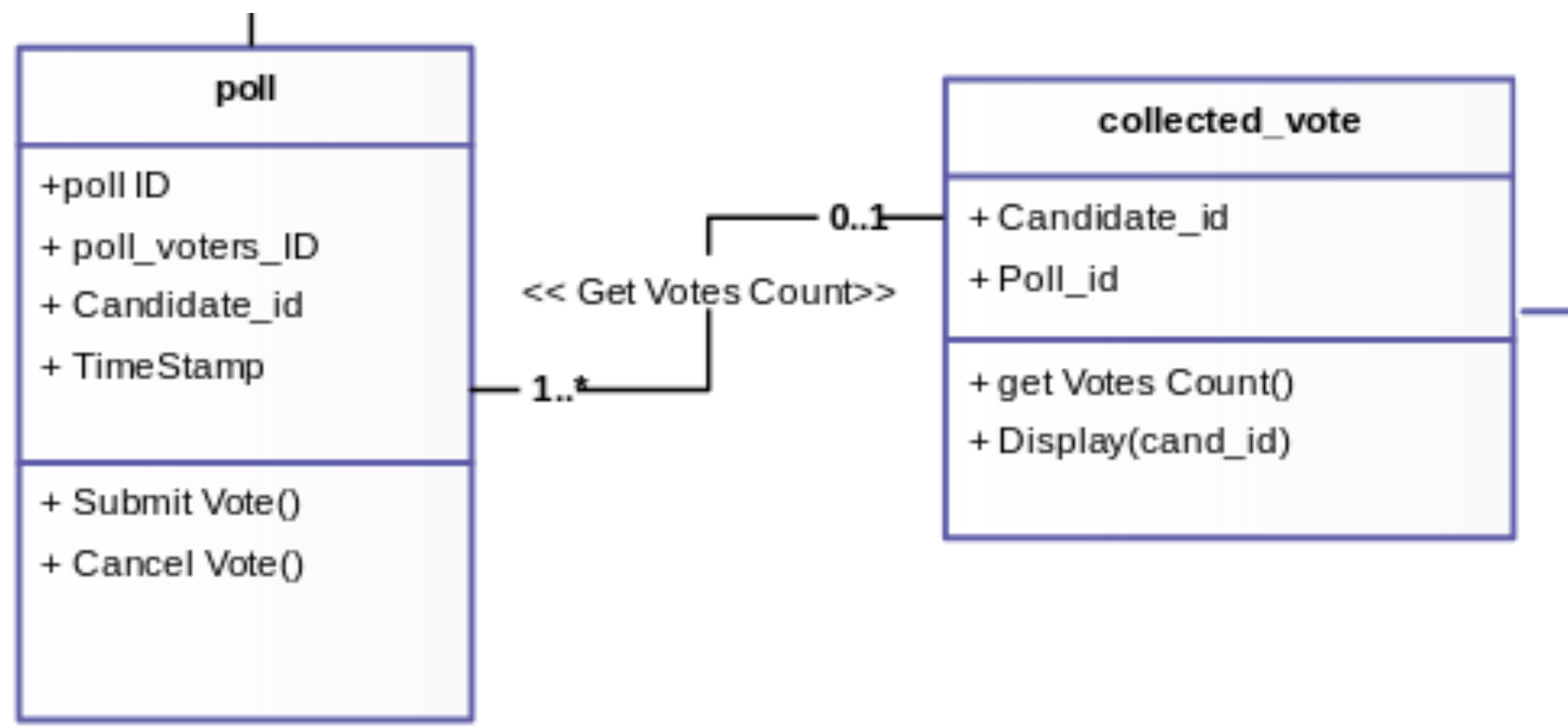
What is a good model ?

What is a good modelling language ?

What is a good modelling method ?



↓
UML



↓
Java

```
public class Poll { ... }
```

↓
Java Byte Code

```

118      new #68 <Class javax/realtime/PeriodicParameters>
121      dup
122      aload      4
124      aload      5
126      aload      8
128      aload      5
130      aconst_null
  
```

↓
Machine Code

110000010100011101000110

Building a Model

Modelling Method - any technique concerned with the construction and/or analysis of mathematical models which aid the development of computer/information systems

Some *toy* modelling languages will help us explore the fundamental concepts - consistency, completeness, coherency, validation, verification, testing ...

We will not be using UML/Java but the lessons are the same !!

Typographical Re-write Systems (TRS)

A TRS is a formal system based on the ability to generate a set of strings following a simple set of syntactic rules.

Each rule is calculable --- the generation of a new string from an old string by application of a rule always terminates

A TRS may produce an infinite number of strings

TRSs can be as powerful as any computing machine (IS)

TRSs are simple to implement (simulate)

Case Study 1 --- The MUI TRS

Thanks to Hofstadter -
https://en.wikipedia.org/wiki/Gödel,_Escher,_Bach

Alphabet = {M,I,U}

Strings: any sequence of characters found in the alphabet

Axiom: MI

Generation Rules: for all strings such that x and y are strings of MUI or ' ' :

- 1) xI can generate xIU
- 2) Mx can generate Mxx
- 3) $xIIIIy$ can generate xUy
- 4) $xUUy$ can generate xy

A **theorem** of a TRS is any string which can be generated from the axioms (or any other theorem)

A **proof** of a theorem corresponds to the set of rules which have been followed to generate that theorem

The model is
executable - a
“program”

Input -
a string

Output -
true/false
(with optional
proof)

Case Study 1 --- The MUI TRS

Alphabet = {M,I,U}

Strings: any sequence of characters found in the alphabet

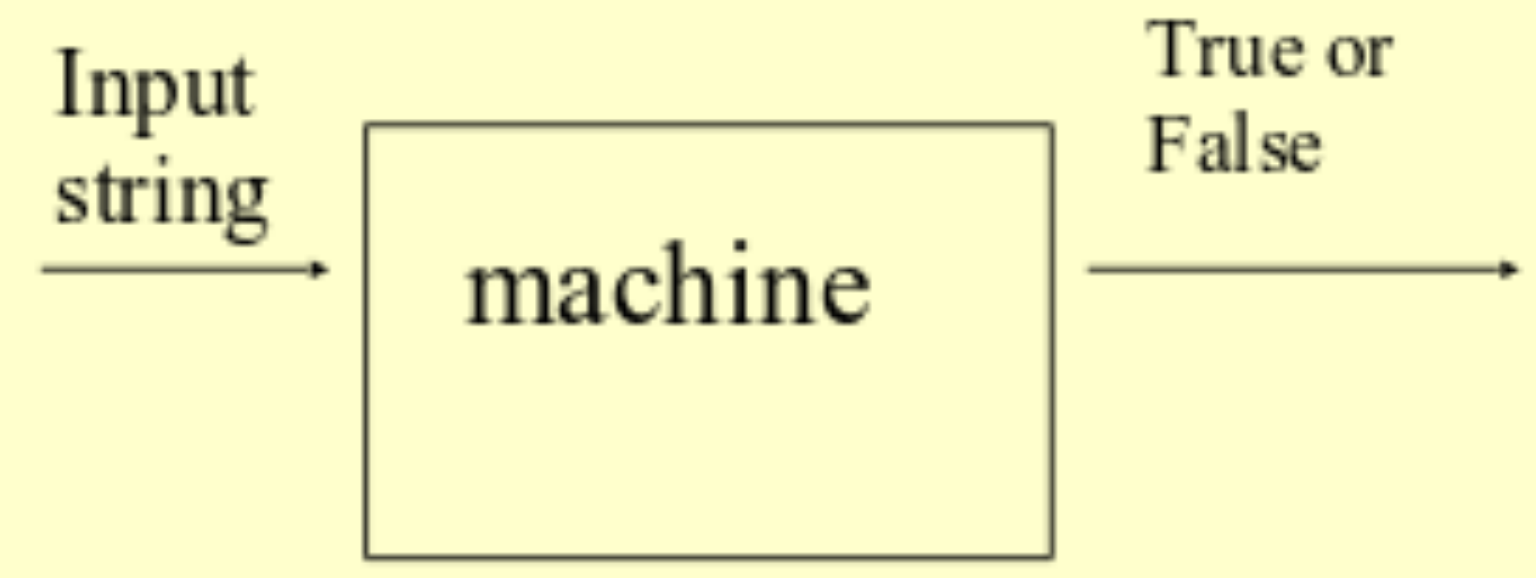
Axiom: MI

Generation Rules: for all strings such that x is a string of MUI or x = "":

- 1) xI can generate xIU
- 2) Mx can generate Mxx
- 3) xIIly can generate xUy
- 4) xUUy can generate xy

Question: can you prove the theorem MUIIU?

Question: can we automate the process of testing for theoremhood of a given string in a finite period of time?



Such a machine would be a **decision procedure** of MUI

Case Study 1 --- The MUI TRS

Alphabet = {M,I,U}

Strings: any sequence of characters found in the alphabet

Axiom: MI

Generation Rules: for all strings such that x is a string of MUI or x = '':

- 1) xI can generate xIU
- 2) Mx can generate Mxx
- 3) xIIly can generate xUy
- 4) xUUy can generate xy

Question: is IIIIUUUUIIUUI a theorem of the system?

Question: before we move on ... is MU a theorem of MUI ?

Case Study 2 --- The pq- TRS

Alphabet = $\{p,q,-\}$

Axiom: for any such x such that x is a possibly empty sequence of ‘-’s,

$xp-qx-$ is an axiom

Generation Rules: for any x,y,z which are possibly empty sequences of ‘-’s,

if $xpyqz$ is a theorem then $xpy-qz-$ is a theorem

Question: is there a decision procedure for this formal system?

Hint: all re-write rules lengthen the string so ...?

Alphabet = {p,q,-}

Axiom:

for any such x such that x is a possibly empty sequence of '-'s, xp-qx- is an axiom

Generation Rules:

for any x,y,z which are possibly empty sequences of '-'s, if xpyqz is a theorem then xpy-qz- is a theorem

Case Study 2 --- The pq- TRS interpretation

If we interpret

- p as plus
- q as equals
- and a sequence of n '-'s as the integer n

then we have

a means of checking $x+y=z$ for all non-negative integers x,y and z

We say that pq- is **consistent** (under the given interpretation) because all theorems are true after interpretation

We say that pq- is **complete** if all true statements (in the domain of interpretation) can be generated as theorems in the system.

We say that the interpretation is **isomorphic** to the system if the system is both complete and consistent

Modellers strive for consistency and completeness

Case Study 2 --- The pq- TRS extension

The pq- system is isomorphic to a very limited domain of interpretation (but maybe that is all that is required!)

Normally, to widen a domain we can

add an axiom

add a generating rule

For example, what happens if we add the axiom:

$$xp - qx.$$

Using this, we can generate many new theorems!

Question: with this new axiom what about completeness and consistency?

Case Study 2 --- The extended pq- TRS reinterpreted

After extension,

--p--q--- is now a theorem but $2+1=2$ is not true

To solve this problem we can re-interpret for consistency ---

interpet q as " \geq "

However, we have now lost completeness ---

" $2+5 \geq 4$ " is true (in our domain of interpretation) but

--p-----q----- is a non-theorem

Note: this is a big problem of mathematics (c.f Church) ---

it is not possible to have a complete, decidable system of mathematical properties which is consistent

if all the theorems that can be checked are consistent then there are some things which we would like to be able to prove as theorems which the system is not strong enough for us to do

Question :
Impact on
Requirements
Modelling ?

Case Study 3 --- A tq- TRS

Question:

- can you define a TRS for modelling the multiplication of two integers
- can you show that it is complete and consistent

Interpretation:

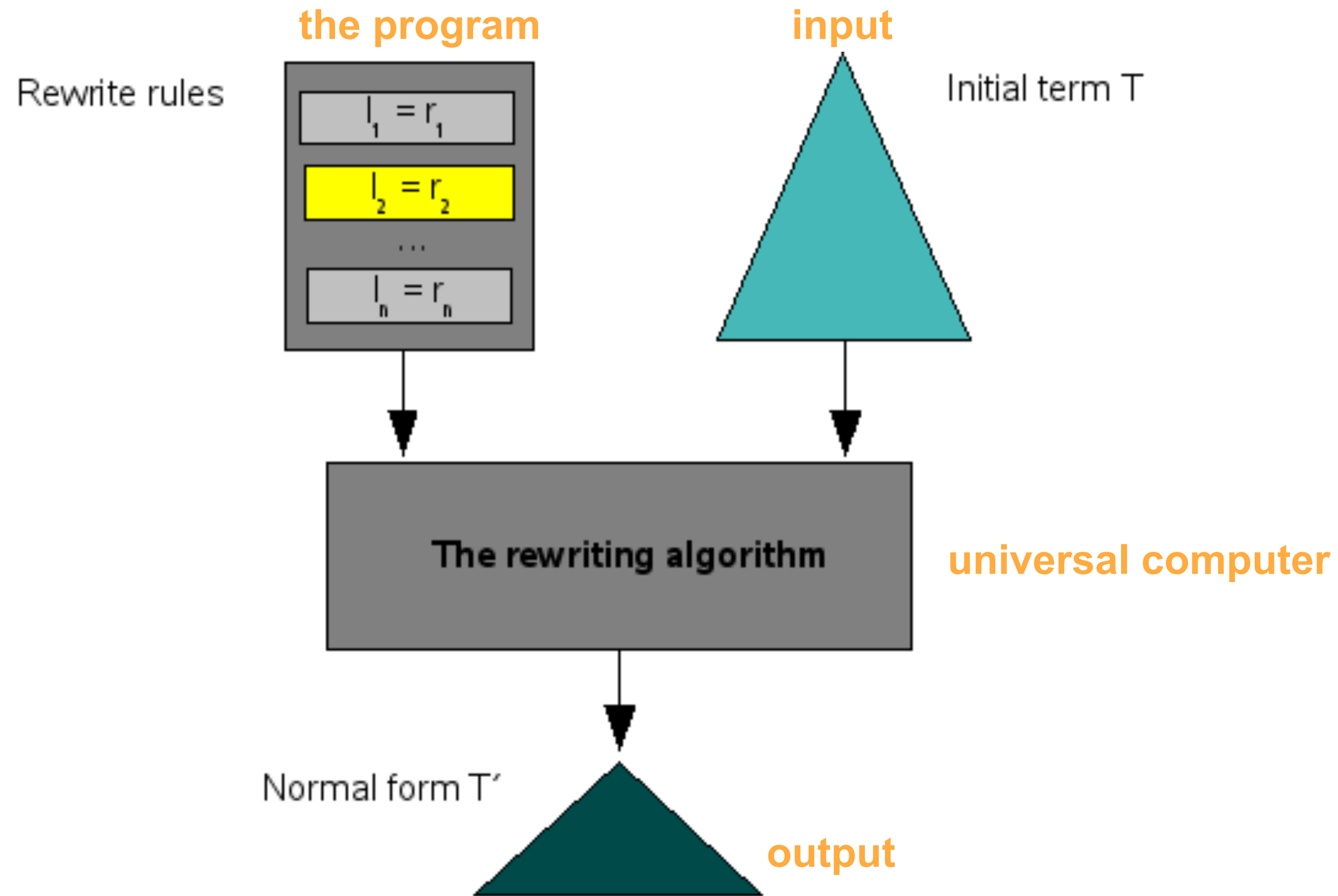
- t as times
- q as equals
- sequences of ‘-’s as integers

Problem 1 -

Define a TRS that can decide if a natural number is composite

OPTIONAL (DIFFICULT) - Define a TRS that can decide if a natural number is prime

Term Re-writing - another view/mechanism - towards ADTs



This is the computational model behind many Abstract Data Types (ADTs)

<http://www.meta-environment.org/doc/books/extraction-transformation/term-rewriting/term-rewriting.html>

ADTs are a powerful specification technique which exist in many forms/languages

They are often used to model requirements, and to specify abstract classes in OO models

A simple ADT specification of integer addition

TYPE integer SORTS integer, boolean

OPNS

0:-> integer

succ: integer -> integer

eq: integer, integer -> boolean

+: integer, integer -> integer

EQNS forall x,y: integer

0 eq 0 = true; succ(x) eq succ(y) = x eq y;

0 eq succ(x) = false; succ(x) eq 0 = false;

0 + x = x; succ(x) + y = x + (succ(y));

ENDTYPE

How do we show, for example,

“1+2 = 3” is “true”

By a sequence of rewrite rules

“succ(0) + succ(succ(0)) eq succ(succ(succ(0)))”

“0 + succ(succ(succ(0))) eq succ(succ(succ(0)))”

“succ(succ(succ(0))) eq succ(succ(succ(0)))”

“succ(succ(0)) eq succ(succ(0))”

“succ(0) eq succ(0)”

“0 eq 0”

“true”

A simple ADT specification of integer addition

TYPE integer SORTS integer, boolean

OPNS

0:-> integer

succ: integer -> integer

eq: integer, integer -> boolean

+: integer, integer -> integer

EQNS forall x,y: integer

0 eq 0 = true; succ(x) eq succ(y) = x eq y;

0 eq succ(x) = false; succ(x) eq 0 = false;

0 + x = x; succ(x) + y = x + (succ(y));

ENDTYPE

Question: how do we show

• **$3+2 = 4+1$**

• **$2+2 \neq 3+2$**

A simple ADT specification of integer addition

TYPE integer SORTS integer, boolean

OPNS

0:-> integer

succ: integer -> integer

eq: integer, integer -> boolean

+: integer, integer -> integer

EQNS forall x,y: integer

0 eq 0 = true; succ(x) eq succ(y) = x eq y;

0 eq succ(x) = false; succ(x) eq 0 = false;

0 + x = x; succ(x) + y = x + (succ(y));

ENDTYPE

Question:

Extend the model to include multiplication

An ADT for a set of integers

TYPE Set SORTS integer, boolean

OPNS

empty:-> Set

str: Set, integer -> Set

add: Set, integer -> Set

contains: Set, integer -> boolean

EQNS forall s:Set, x, y :integer

contains(empty, x) = false;

x eq y => contains(str(s,x), y) = true;

not (x eq y) => contains(str(s,x), y) = contains(s,y);

contains(s,x) => add(s,x) = s;

not(contains(s,x)) => add(s,x) = str(s,x)

ENDTYPE

Note: the new syntax for preconditions

Question:

How do you interpret each of the operations and equations?

Is this a valid interpretation for a set of integers?

An ADT for a set of integers

TYPE Set SORTS integer, boolean

OPNS

empty:-> Set

str: Set, integer -> Set

add: Set, integer -> Set

contains: Set, integer -> boolean

EQNS forall s:Set, x, y :integer

contains(empty, x) = false;

x eq y => contains(str(s,x), y) = true;

not (x eq y) => contains(str(s,x), y) = contains(s,y);

contains(s,x) => add(s,x) = s;

not(contains(s,x)) => add(s,x) = str(s,x)

ENDTYPE

Question:

add operations for --

- remove
- union
- equality

Problem 2 -

Write an ADT specification for a stack of integers

Write an ADT specification for a queue of integers

Compare and contrast the 2 models