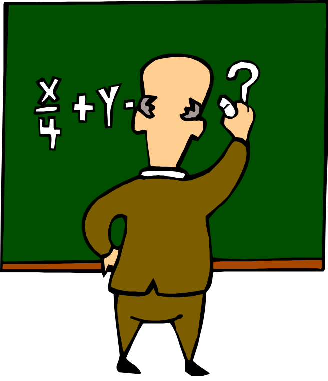
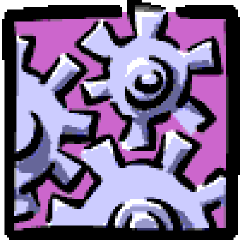
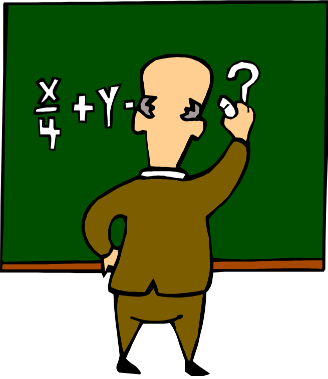


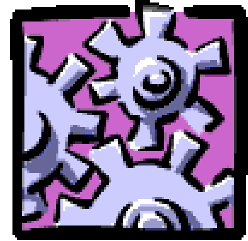


Test de Systèmes Informatiques

Burkhart Wolff
Université Paris-Sud



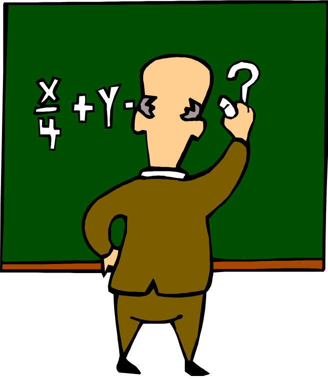
Software Testing can be formal too



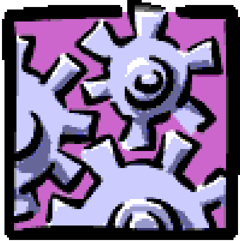
- *« We know less about the theory of testing, which we do often, than about the theory of program proving, which we do seldom »*

  Goodenough J. B., Gerhart S.,

  IEEE Transactions on Software Engineering, 1975

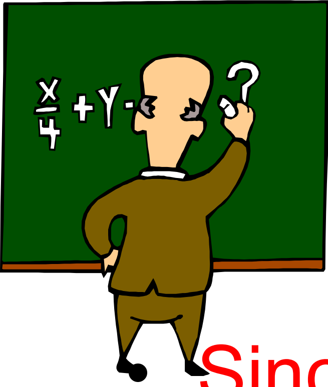


Relevance

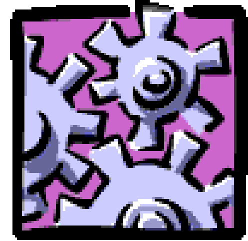


Why is it important to get software right?

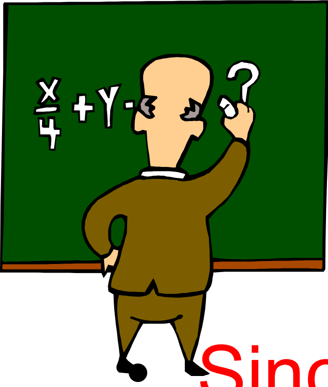
? ? ?



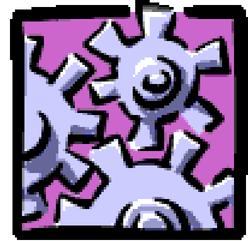
Relevance



- Since information technology becomes more pervasive, the risks become more important
  - Reliability, Safety and Security becomes more critical :
    - transport systems (Cars, Métros, TGV), aviation controls, aerospace, ...
    - critical industriel processes, nuclear power plants, weapons
    - medical technologies: tele-surgery, radiation control…
    - critical telecommunication infrastuctures and networks,
    - electronic commerce (SAP)



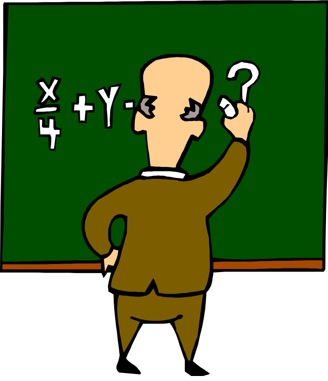
Relevance



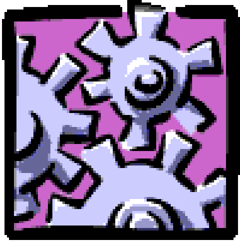
Since information technology becomes more pervasive, the risks become more important

- Reliability, Safety and Security becomes more critical :
  - transport systems (Cars, Métros, TGV), aviation controls, aerospace, …
  - critical industriel processes, nuclear power plants, weapons
  - medical technologies: tele-surgery, radiation control…
  - critical telecommunication infrastuctures and networks,
  - electronic commerce (SAP, ATOS)

This should be the most important reason, but actually, it isn't.



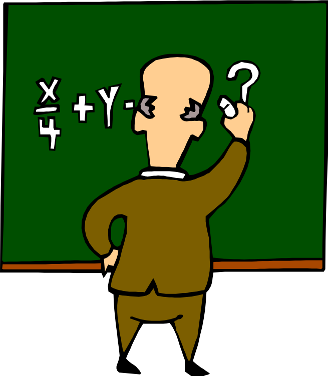
Relevance



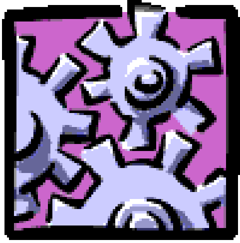
- The more likely reason is:

  it is so expensive if you don't !!!
  (It's the economy, stupid !)

  50 % of the overall costs were spent for test and verification in large software projects ... So, if the development of MS Vista cost 8 billion $ ...



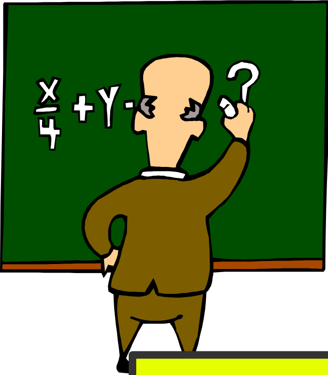
Relevance



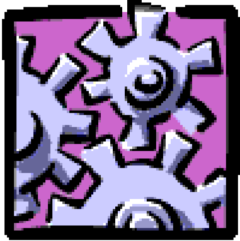
- Another reason is:

  We want to build more complex systems, and validation and verification techniques are a limiting factor!

  We simply can't do it without !

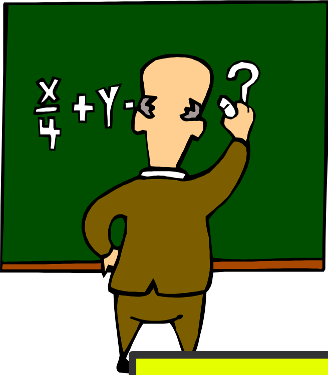


Relevance

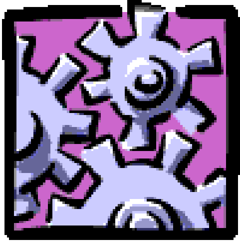


Requirement Analysis → Design → Coding Phase → Integration → Deployment

Requirement Analysis
Design
Coding Phase
Integration
Deployment



Relevance



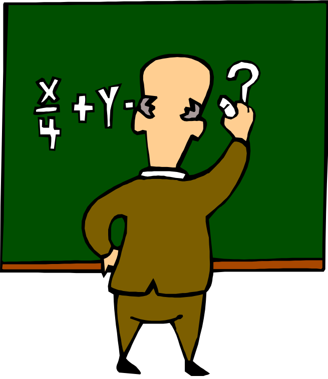
Requirement Analysis

Deployment

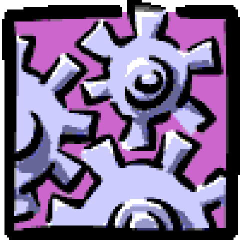
Design

Integration

Coding Phase

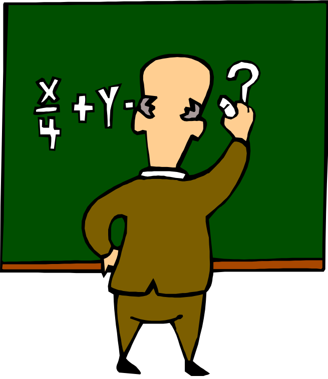


Relevance

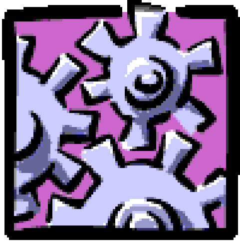


- Yet another reason is:
  - In an industrial setting, you might be bound by laws to provide reasonable (i.e. formal) ducuments …

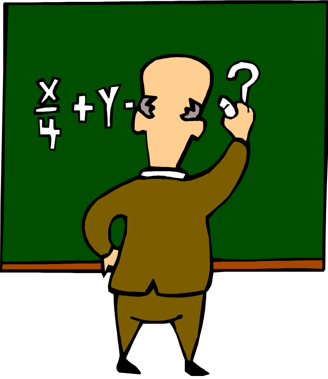
    See Microsoft Windows Server Monopoly Case...



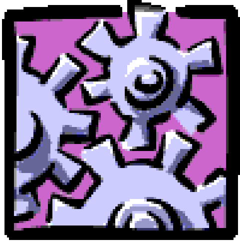
Relevance



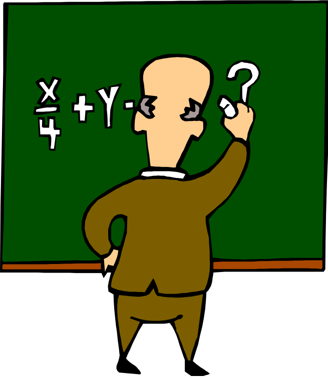
- Reports on industrial Practice: eg.
  W Grieskamp:„Microsoft's Protocol Document Program: A Succes-Story for Model-Based Testing“. TestCom/Fates 09
  - 222 protocols/technical documents tested
  - 22,847 pages studied and converted into formal requirements
  - 250 man years effort
    - 250 test engineers in Hyderabad
    - 100 test engineers in Beijing



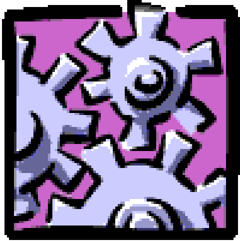
Outline of the course



- Introduction into Model-based Testing
  - Formal specifications
  - Black-box testing
  - Putting them together
- Application to *Specifications in HOL*
- Application to *Finite State Machines*
- Application to *Labelled Transition Systems with inputs and outputs*

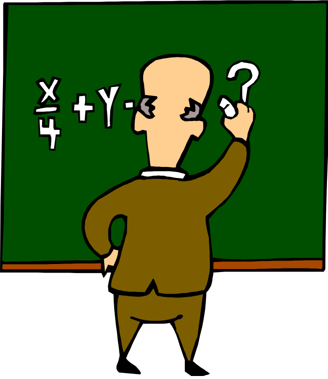


Outline of the course

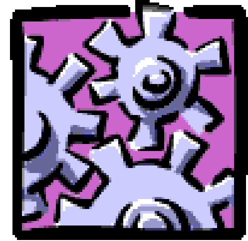


- Introduction
  - White-box testing
  - Putting them together

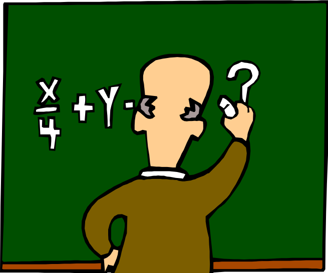
- Application to *IMP (in HOL)*



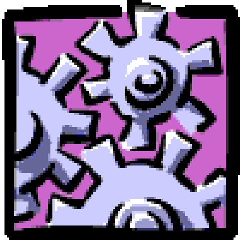
Formal Specification?



- **There is a notation**
  - Pieces of text
    - HOL, Z, VDM, B, CSP, Lotos, …
  - Annotated diagrams
    - Finite State Machines, Petri Nets, « Automata-derived » diagrams (LTS, Kripke structures), Statecharts, …
- **There is a formal semantics**
  - Algebras, Predicate transformers, Sets, Traces and Failures…
- There is a **formal system** (proofs) or a **verification method** (model-checking), or both.



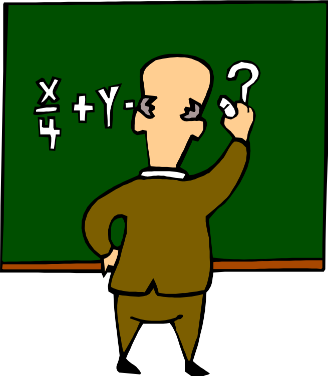
Example 1: HOL



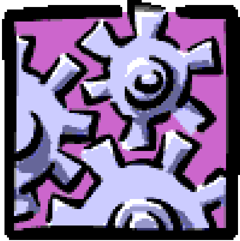
theory CONTAINER
  imports Main
begin
  datatype *Container* ::= [] | _::_(*nat ; Container*)
consts *isin : [Nat, Container] ⇒ bool*
  consts *remove: [Nat, Container] ⇒ Container*
  axioms

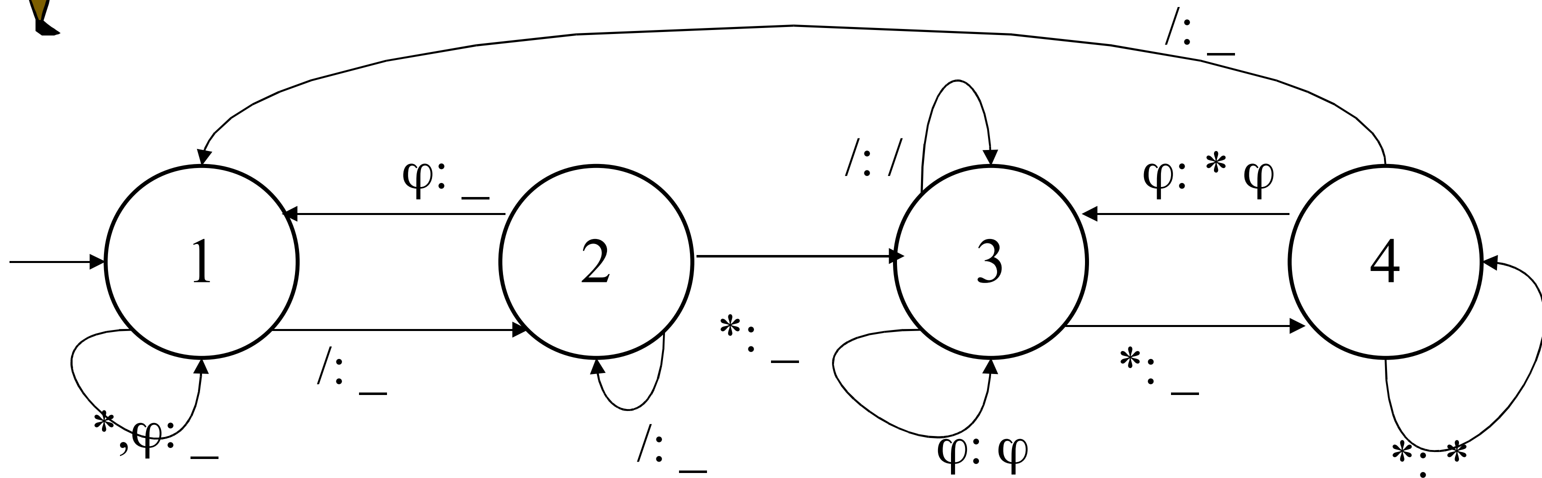
- *isin(x, []) = False*
- *eq(x, y) = True ⇒ isin(x, y::c) = True*
- *eq(x, y) = False ⇒ isin(x, y::c) = isin(x,c)*
- *remove(x, []) = []*
- *eq(x, y) = True ⇒ remove(x, y::c) = c*
- *eq(x, y) = False ⇒ remove(x, y::c) = y::remove(x,c)*

end



Example 2: FSM

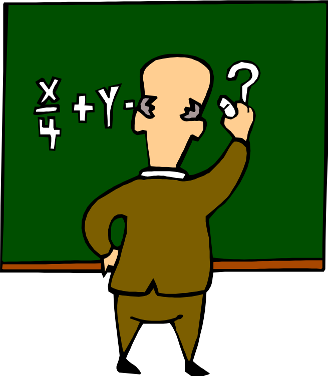




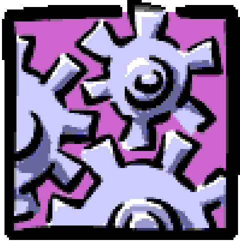
φ = any character but * and /

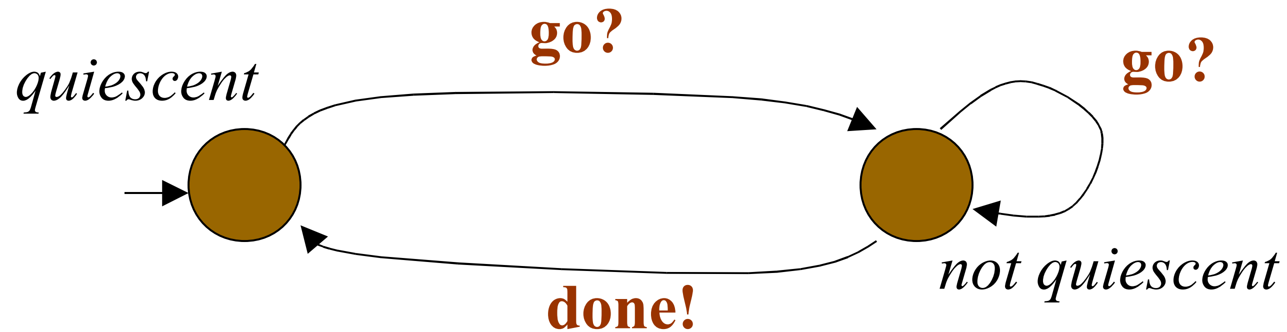
This is not a comment /* all that / *
is ** a comment */ this is no more a comment.

This FSM removes from the input text all that is not a comment

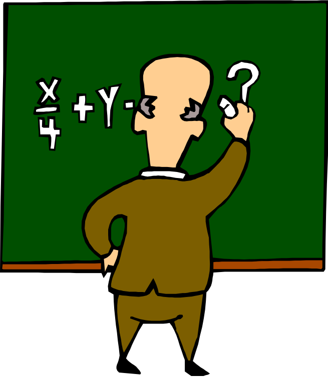


Example 3: IOTS

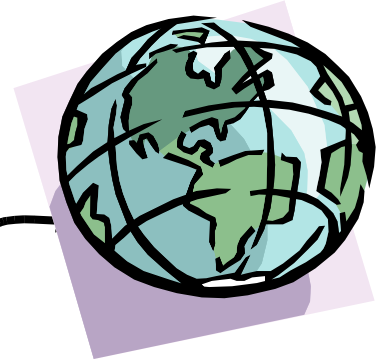




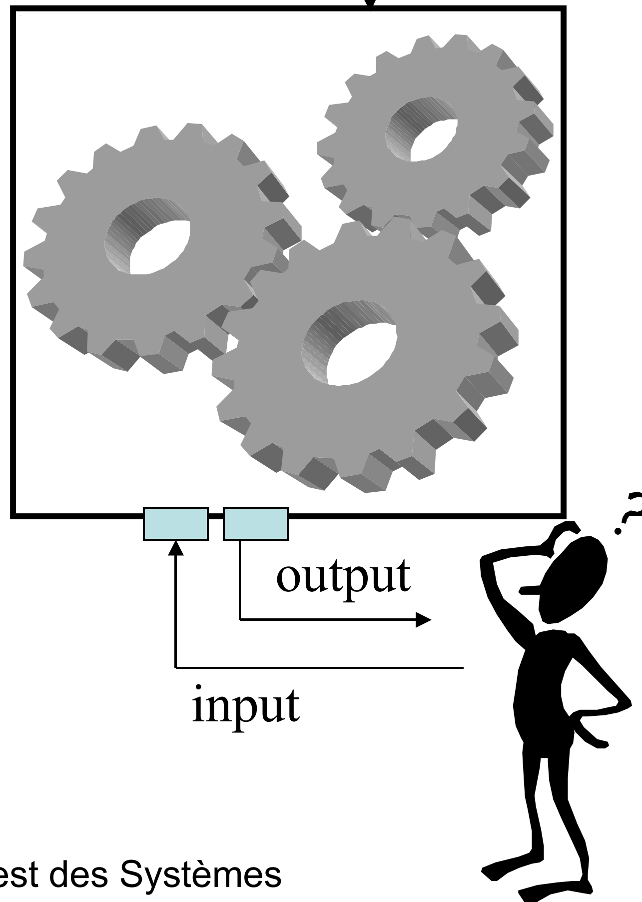
- In its initial state, this IOTS is idle until it receives ***go***
- In any state, it accepts ***go*** (« input-enabledness »); after ***go***, it may emit ***done***.

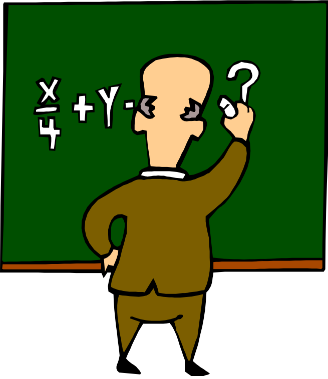


Testing Systems

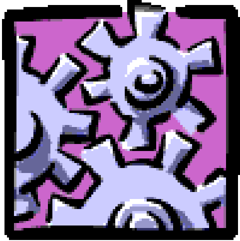


- A system is a dynamic entity, embedded in the physical world
- It is *observable* via some limited interface/procedure
- It is not always *controllable*
- Quite different from a piece of text (formula, program) or a diagram



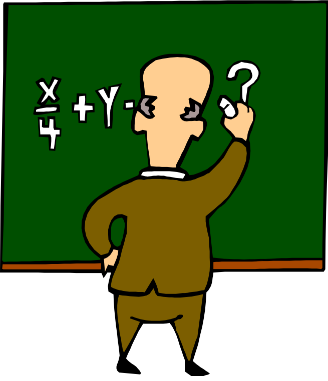


Back to basic concepts

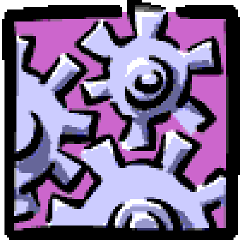


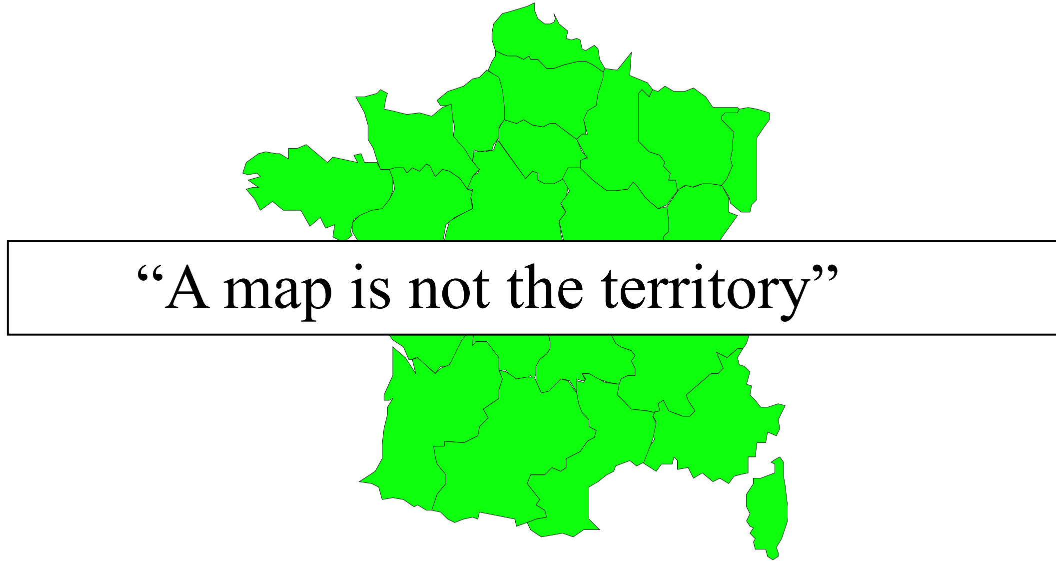
- *One proves formulas*
- *One checks models*
- *One tests systems*

- *A system is not*
  - A formula
  - A formal specification
  - A model
  - A program (considered as a formula when proving programs)



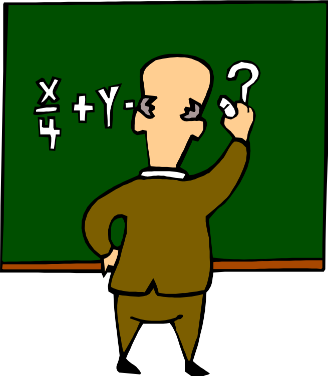
A philosophical interlude



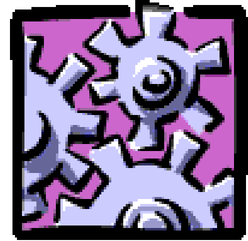


"A map is not the territory"

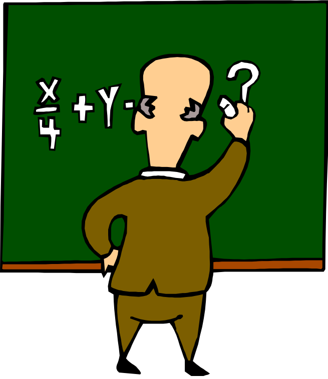
A program text, or a specification text, or a model, is not the system



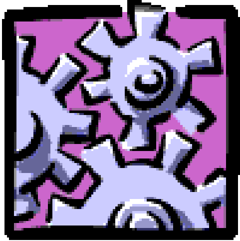
Black-Box Testing



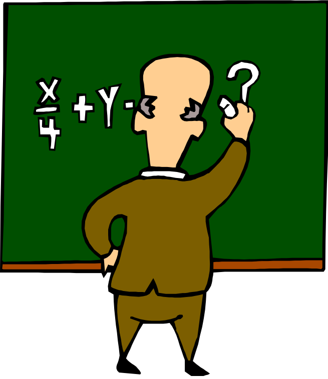
- **Black-Box Testing:**
  - the structure of the SUT (System Under Test) is not known
- **However,**
  - necessity of making explicit the class of "testable implementations" => notion of some **Testability Hypothesis** on the SUT



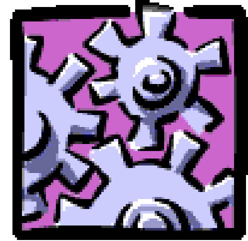
Testability?



- If the SUT can be *any diabolic system*, there is no sensible way of testing it ☹
- Fortunately, *some basic assumptions are feasible* (example: correct implementation of booleans, determinism, …)
- Some others can be *verified in another way*: static checks on the program, preliminary tests, …

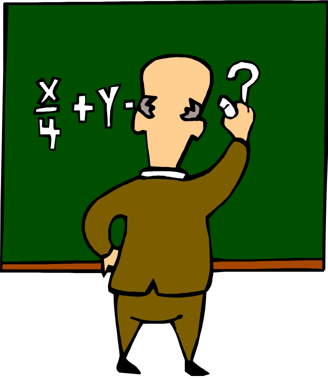


BBT + FS

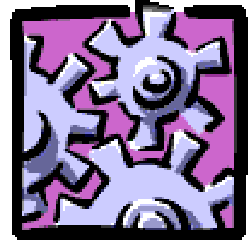


- Importance of an adequate and formal definition of the **Satisfaction Relation** of an implementation w.r.t. a specification

- Careful definition of the class of considered implementations

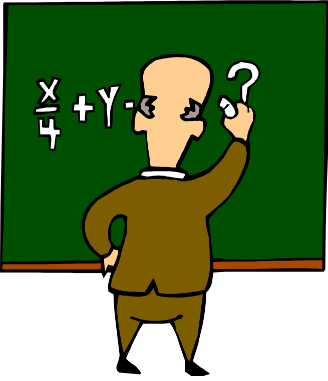


Wanted: a satisfaction relation

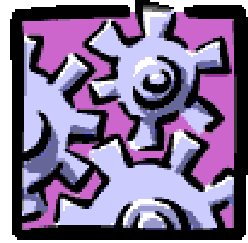


?

- Given some “testable” SUT, what does it mean that it satisfies SP?
- What is the correctness reference? Is there an “exhaustive” (or “complete”) set of tests?
- SP is some sort of model or formula; SUT is some sort of system; how to define *“SUT sat SP”* or *“SUT conf SP”* in such an heterogeneous context?



Content of the course



- The case of Datatype Specifications (HOL)
- The case of Finite State Machines (HOL)
- The case of Input-Output Transition Systems (HOL)
- NB: other cases in the literature