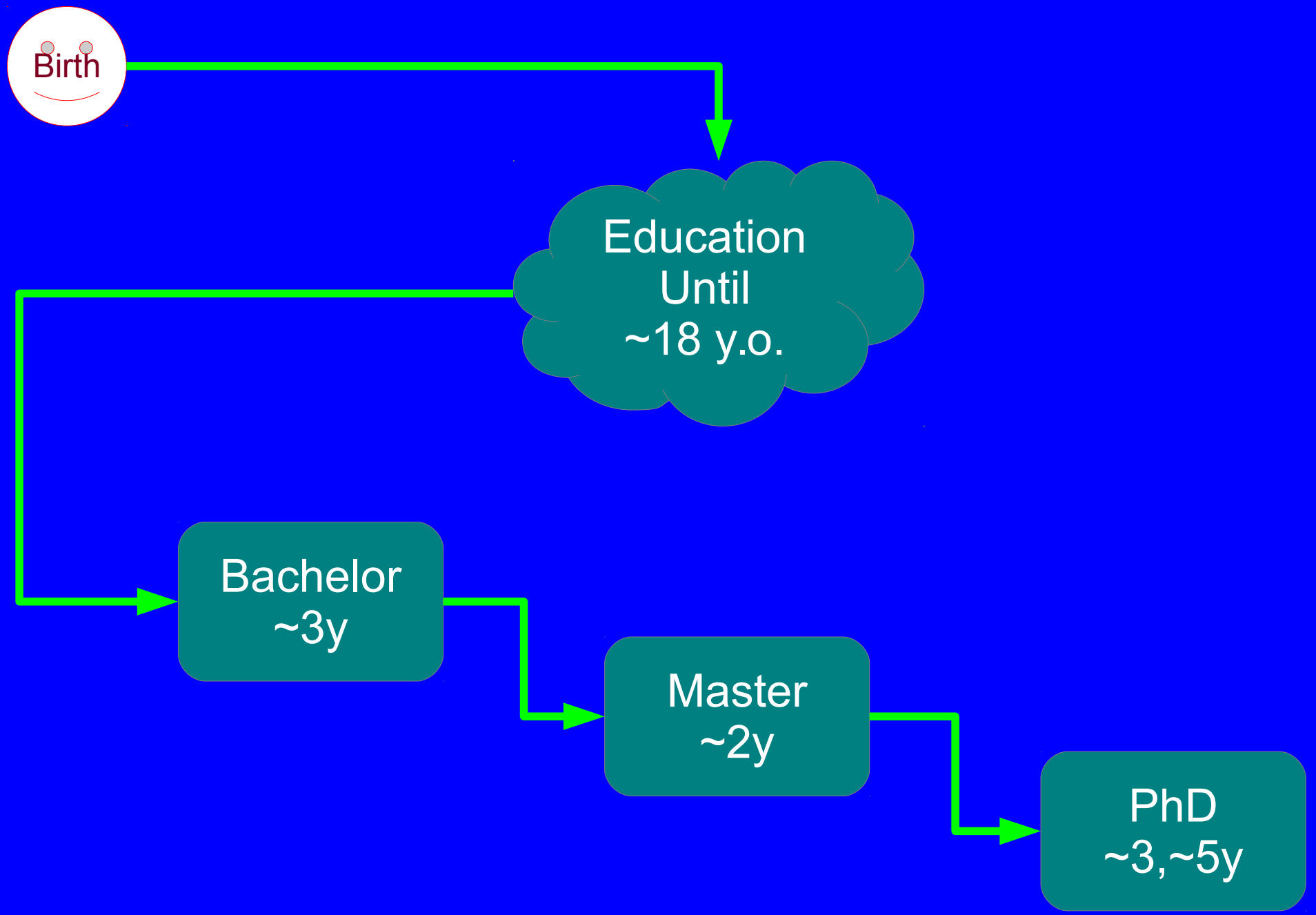
DIT950 Guest Lecture

Shallow introduction to formal methods


Birth
Education
Until
~18 y.o.
Bachelor
~3y
Master
~2y
PhD
~3,~5y

Why study?

Why study?

- get jobs
  - better paid
  - less repetitive
  - more free time (…?)
- learn new things
- be more aware
- enjoy my time

What to study?

What to study?

- Easy stuff
- Hard stuff
- What earns me money
- What I enjoy

Personal growth

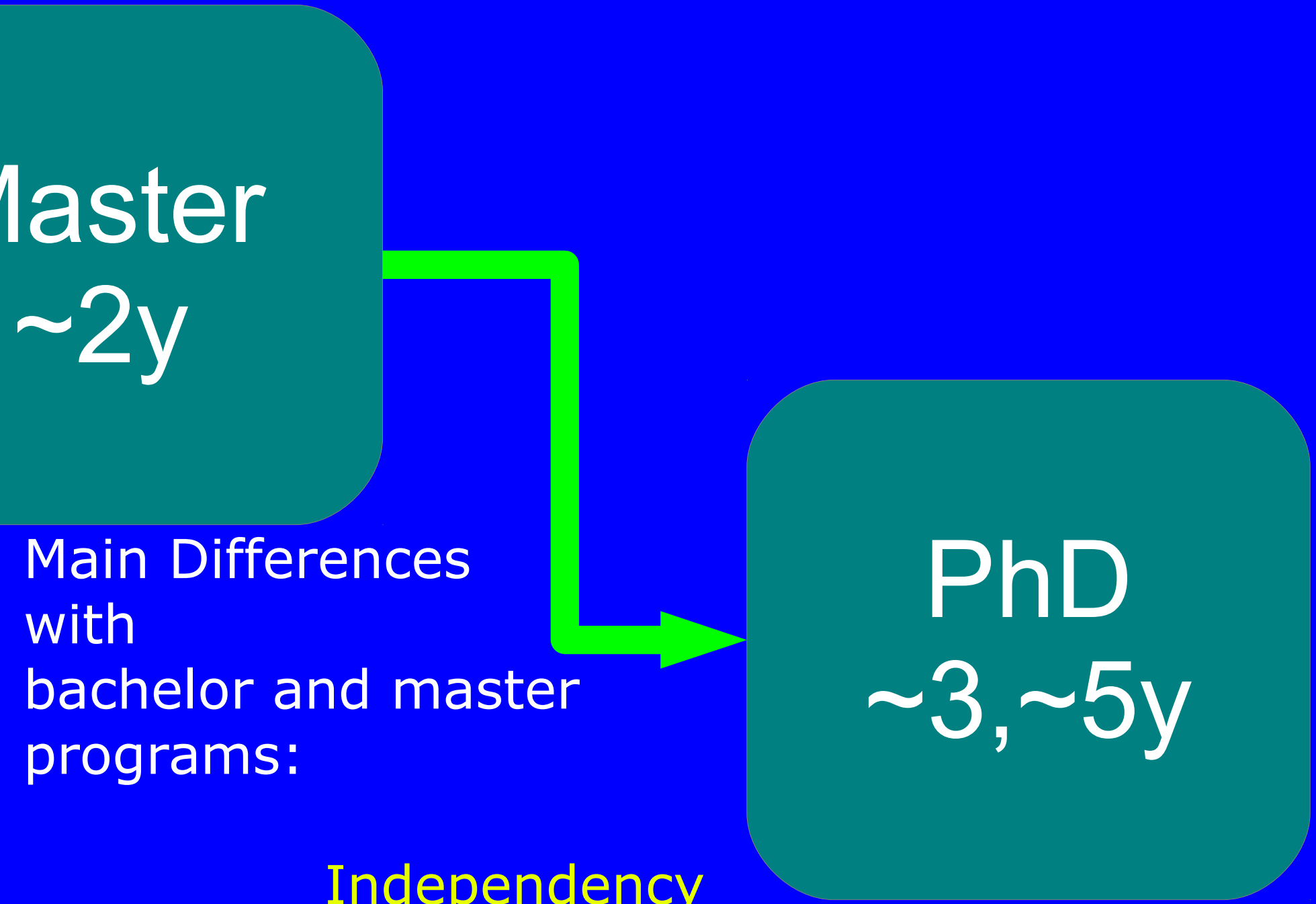
If you are happy,

You and *others* will benefit.

Some see this as the basis of **Capitalism**

Some saw **Capitalism** as a consequence of the **Protestant** reform

- **Max Weber**, "*Die protestantische Ethik und der Geist des Kapitalismus*"
- **Carlo M. Cipolla**, "*Pepper, Wine (and Wool)*"

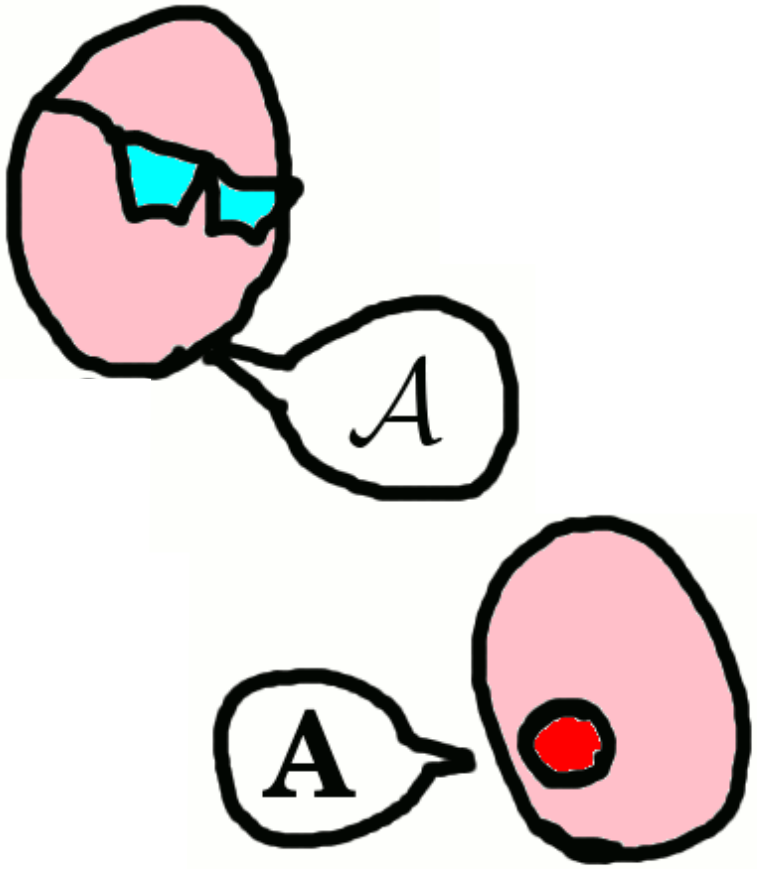

Master
~2y
Main Differences
with
bachelor and master
programs:
PhD
~3,~5y
Independency
Googling for solutions becomes
harder

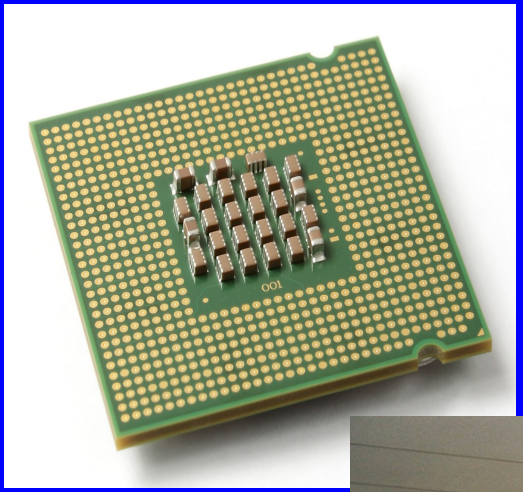
A Significant Unification of Voice-over-IP and Thin Clients

Zlatan Ibrahimovic, Pope Francis I and Sven Göran Eriksson

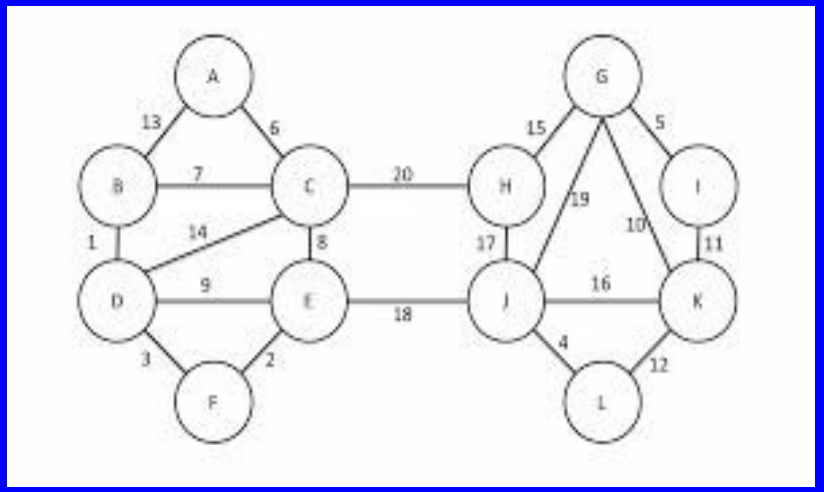
Abstract

In recent years, much research has been devoted to the evaluation of Byzantine fault tolerance; on the other hand, few have constructive characterization, we believe that solution is necessary. In addition, two properties make this solution ideal: our frame

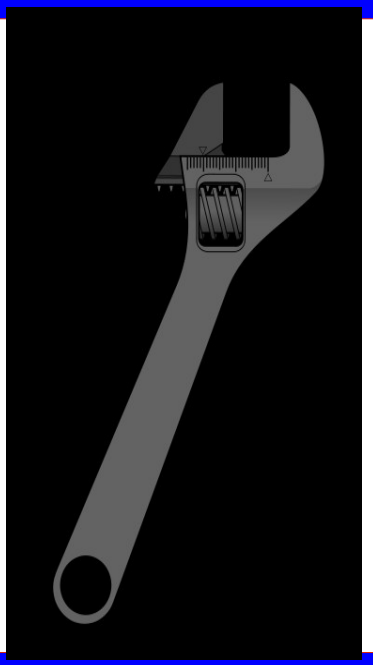




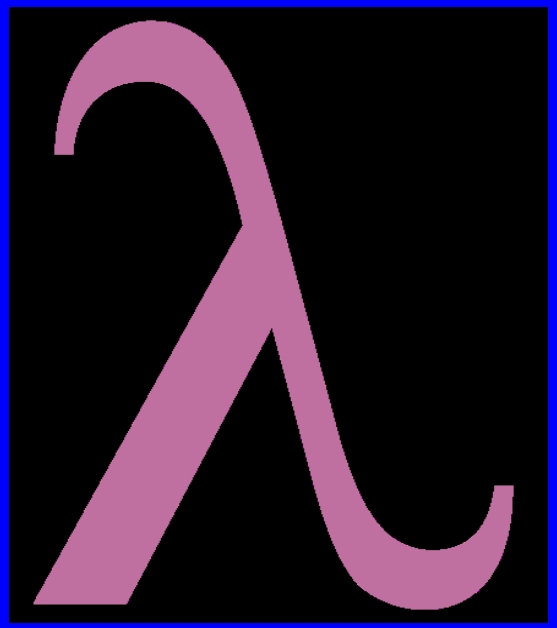
Research at CSE




```
if (a){
   ...
}
else{
   ...
}
```



Research at Software Technology



Functional Programming

Language Based Security

Formal Methods

Formal Methods:

Automate

the process of showing

Correctness of a program

Example: termination as correctness

```
// Does this terminate?
public void meth(int a){
   while(a<3){
      a++;
   }
}
```

Example: type correctness

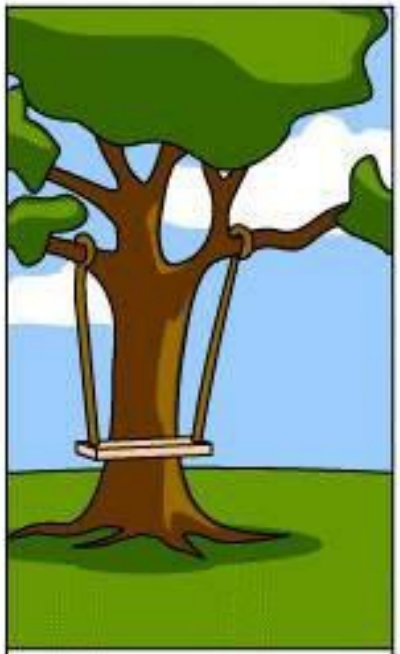
```
// What happens here?
int a = “Kalle Anka”;
```

- Type systems are formal entities (= “mathematically defined”)

Logic and mathematics

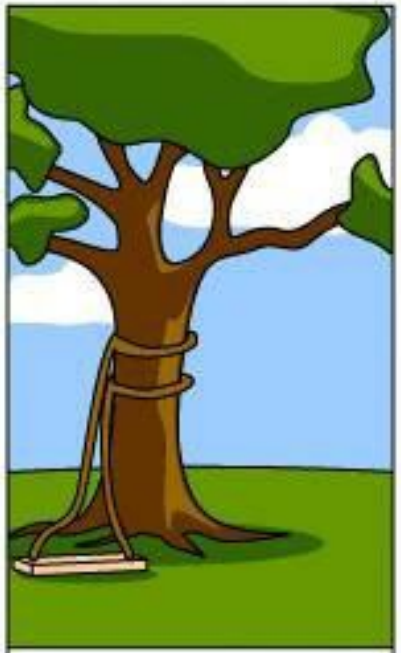
- **Logic** = basis of **formal methods**
- Prevent the **ambiguities** of **natural language**

How the customer explained it



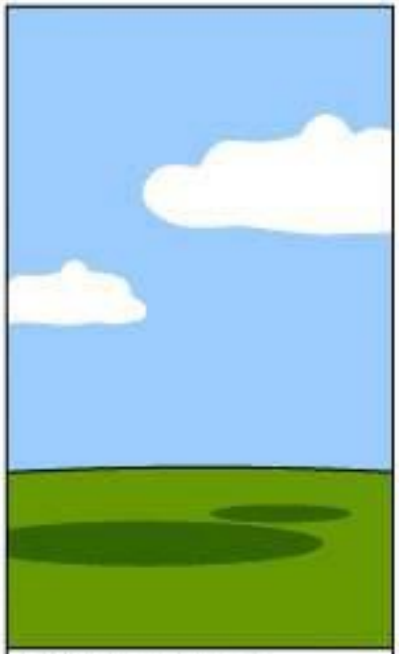
How the Project Leader understood it

How the Analyst designed it

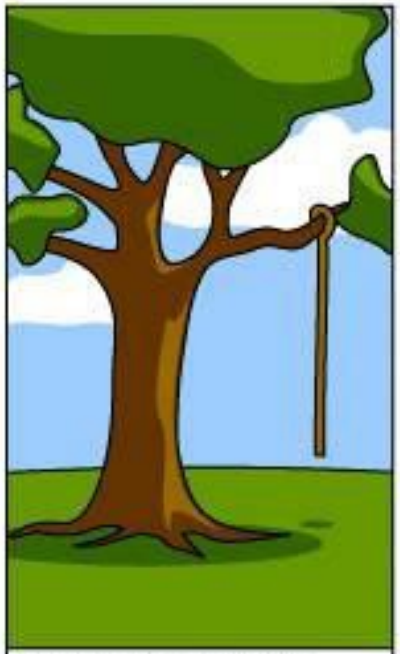


How the Programmer wrote it

How the Business Consultant described it

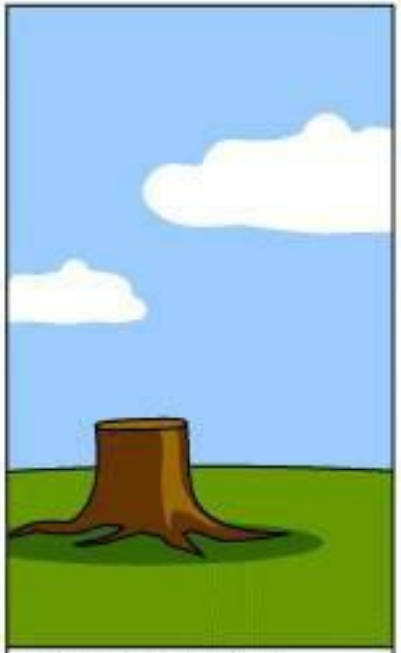


How the project was documented



What operations installed

How the customer was billed



How it was supported

What the customer really needed

Languages

Running dog

Syntax

Semantics: Usually subjective

How to make semantics objective

- Invent a terminology
  - (e.g. Medical science, engineering)
- Use mathematics and logic

Either way: refer to **basic concepts** that we can **assume** everyone understands in the same way

Use of mathematics to describe **syntax** and **semantics** of a language

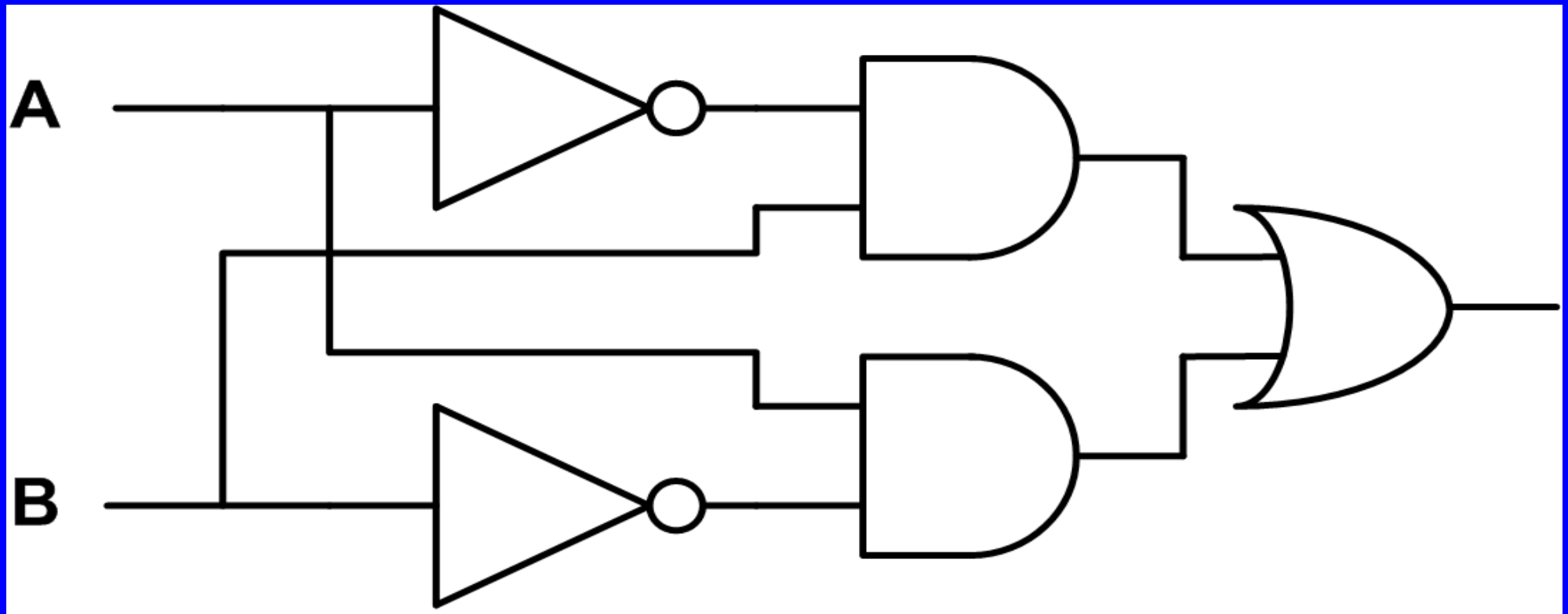
Usually means: computers can process it

Formal Methods theoretical tools

- Formal logics:
  - Propositional
  - N-th order
  - Temporal
- Formal languages

Propositional logic

- Formalizes events/facts:



First (N-th) order logic

- Formalizes
  - events+**objects**
  - their **properties**
  - The relations among **objects**
  - **First order logic contains Propositional logic**

FOL

- Red(ferrari) Expensive(ferrari)
  Red(bugatti) Expensive(bugatti)
  Red(lamborghini) Expensive(lamborghini)
- **Constants**: bugatti, ferrari, lamborghini
- **Predicates**: Red, Expensive

- In FOL: 3 constants+2 predicates
- In PL : 3*2 symbols: must express everything as a fact
  - e.g. “ferrari is red”, “lamborghini is expensive”
- If a porsche is added, in PL need to define 2 more symbols

FOL vs PL

- FOL allows to express properties of items in sets

  In a compact way, without having to know the items in advance

$$\forall x \in \mathbb{N}.x < x + 1$$

More on this:

- DAT060, Logic in Computer Science

Formal Methods tools

- Theorem proving
- Static analysis
- Model checking
- Specification Languages

Theorem Prover

- Theorem:

  logical formula that is always true *(valid)*.
  - And there exist a proof of such validity.

  e.g. Sum of triangle's angles is 180°
- A theorem prover can prove a logical formula (semi) automatically.

Static analysis

- Algorithms that can extract properties of a program without running it.
- e.g. “No null-pointer exception is thrown”
- Formal syntax and formal semantics is needed

Typical use

static analysis algorithm

transforms a

program+desired property

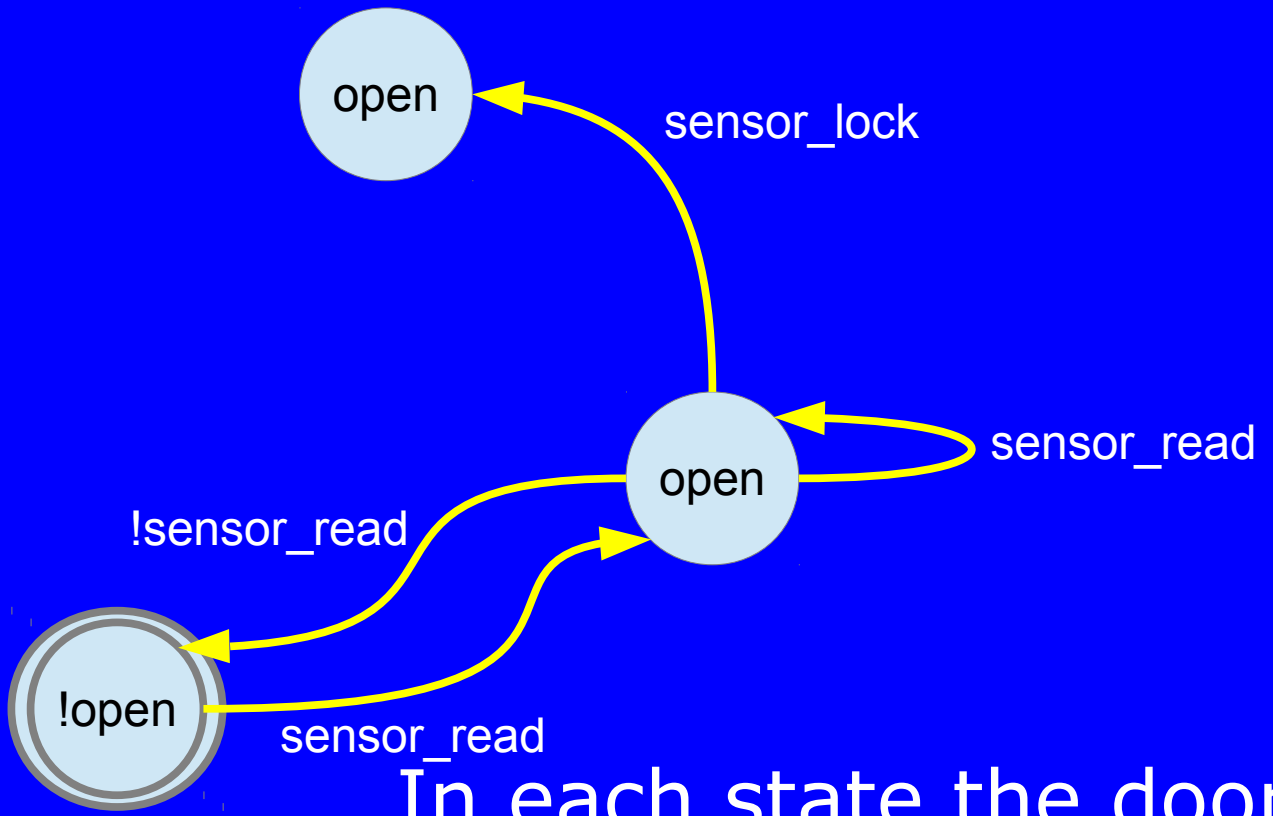
into a logical formula

then input to **theorem prover** to see if it holds.

Model Checking

- Software described by
  - Sequence of machine states
- To prove some properties
  - Not all states are needed
- “Abstract away” the states
  - Ockham's razor

Automatic door



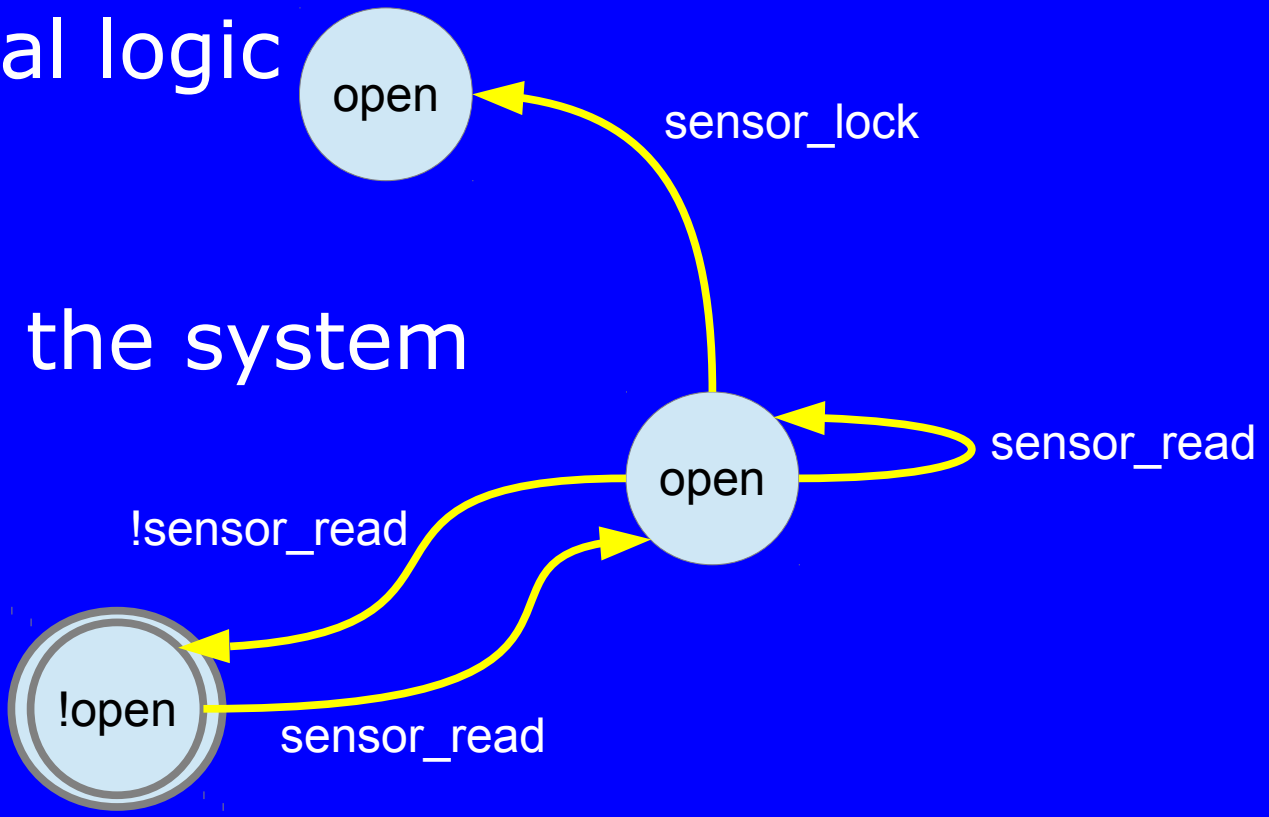
In each state the door is either

open

or not open (!open)

Automatic door

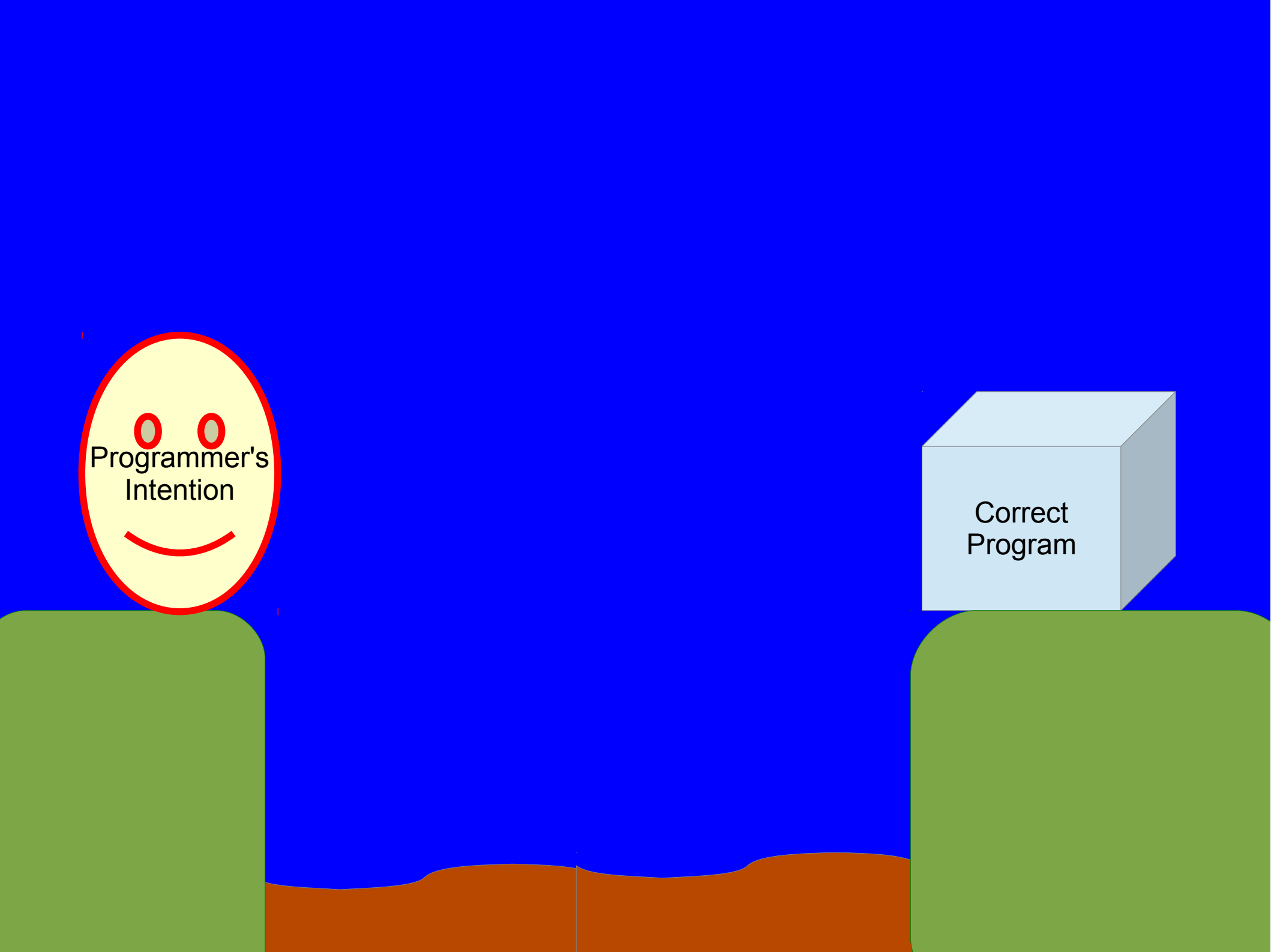
Using temporal logic
one can ask
questions
about runs of the system

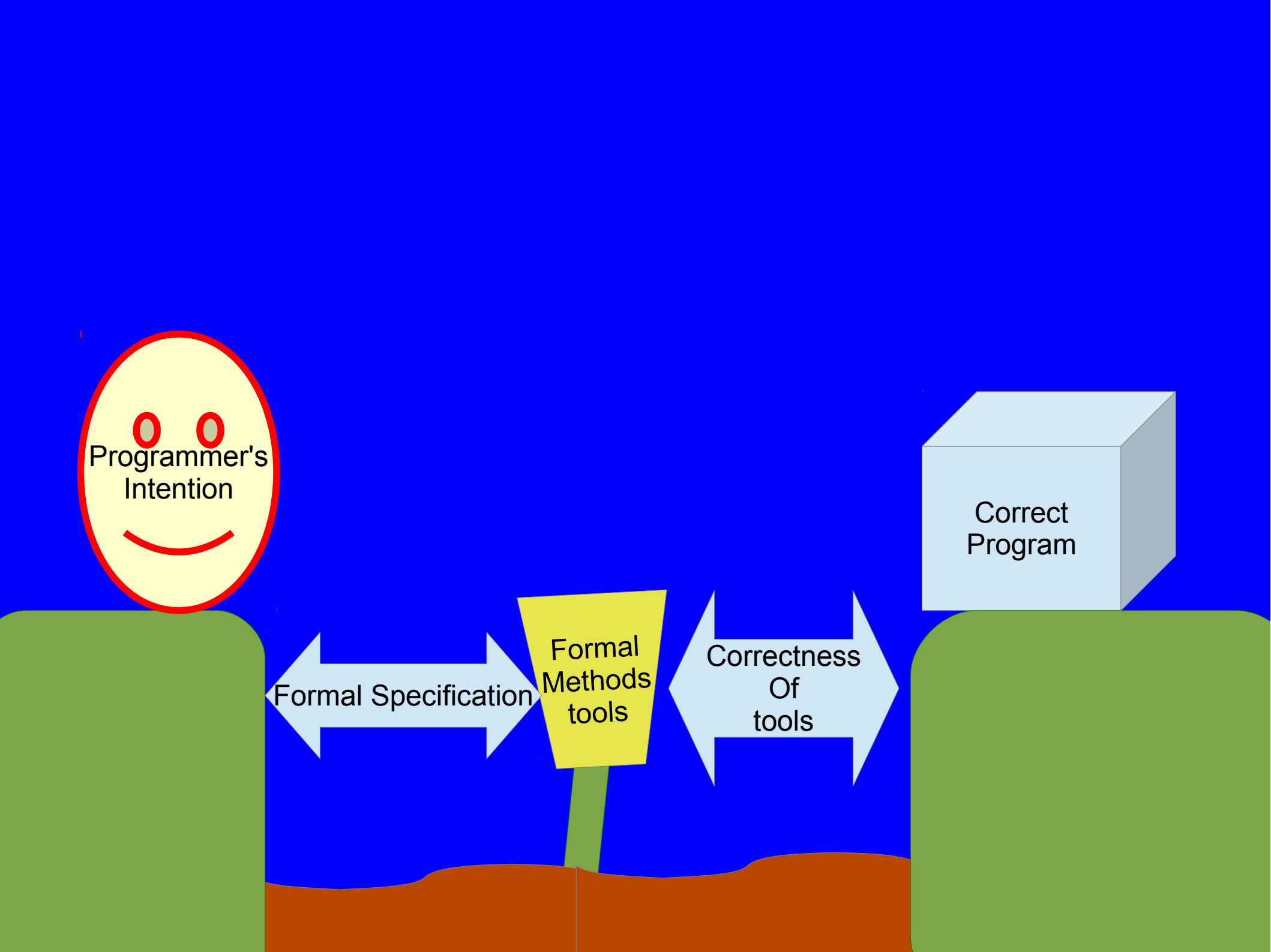


- ■ open (always open)
- ◆ open (eventually open)
- ◆■ open (eventually open)

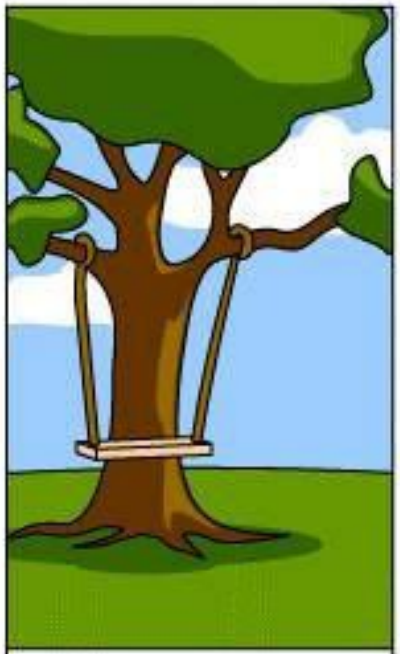
Formal Specification language

- Another computer language
- for telling what a program should do
  - Not for “computing things”
- Efficiency *should* not be important
- **Correctness becomes dependent from/defined by such specification**


Programmer's
Intention
Correct
Program

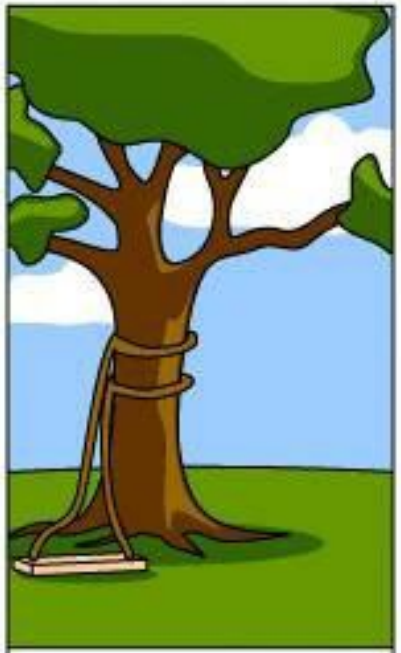

Programmer's Intention
Formal Specification
Formal Methods tools
Correctness Of tools
Correct Program

How the customer explained it



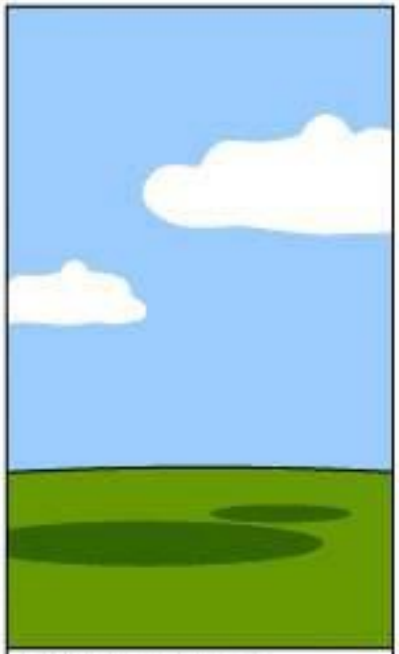
How the Project Leader understood it

How the Analyst designed it

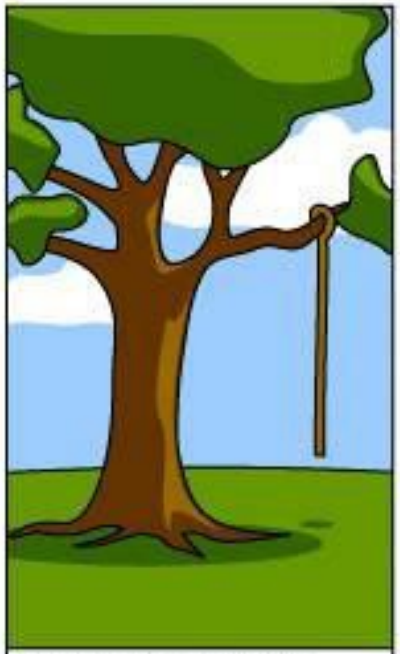


How the Programmer wrote it

How the Business Consultant described it

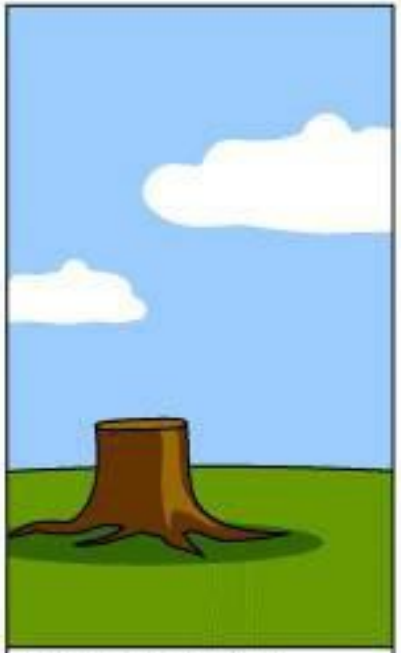


How the project was documented



What operations installed

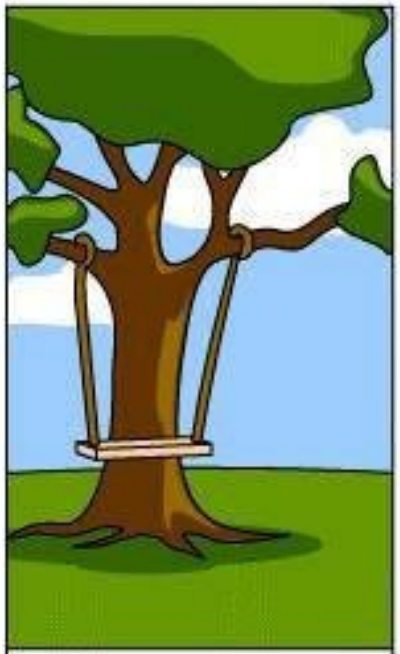
How the customer was billed



How it was supported

What the customer really needed

How the customer explained it

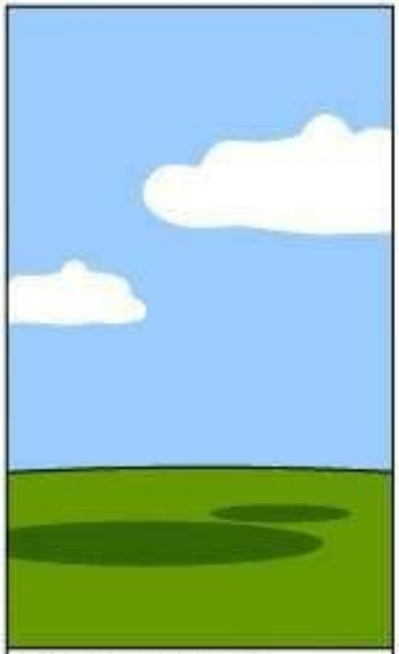


How the Project Leader understood it

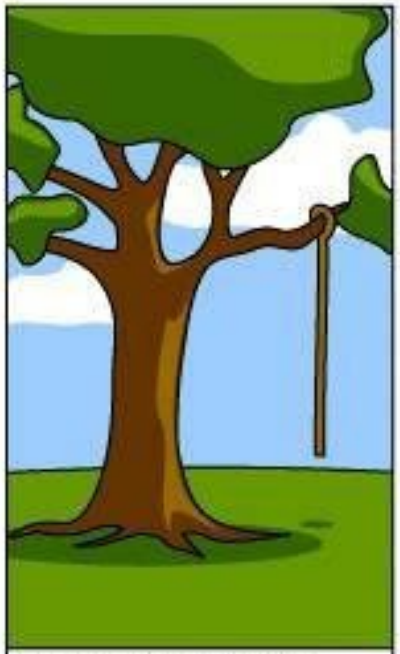
How the Analyst designed it

How the Programmer wrote it

How the Business Consultant described it

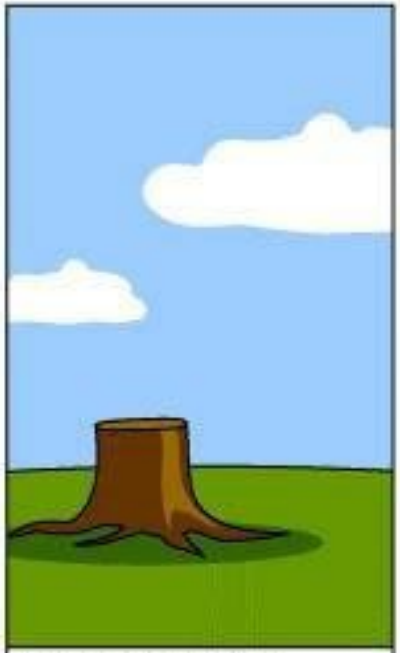


How the project was documented



What operations installed

How the customer was billed



How it was supported

What the customer really needed