Good coding practice in real life

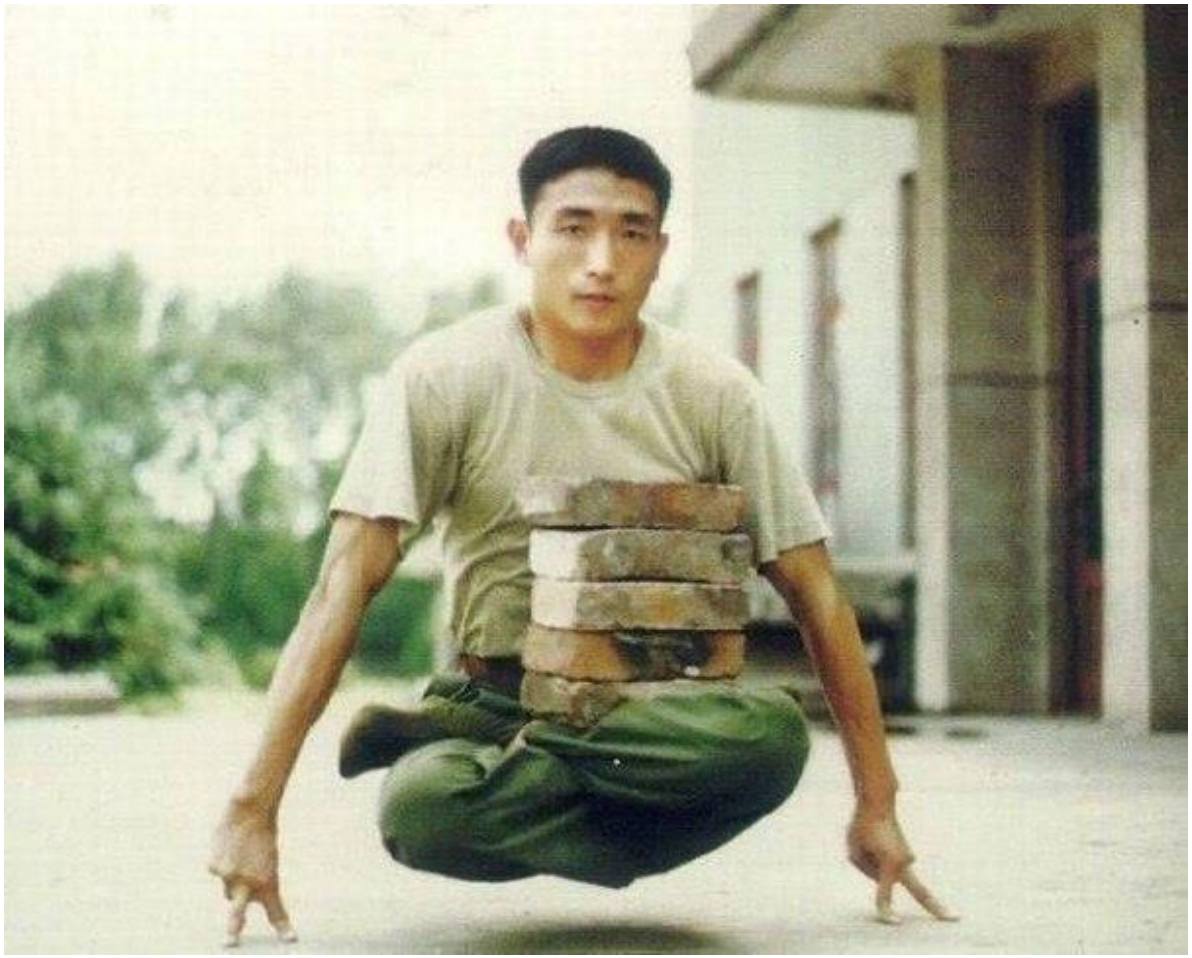
with a focus on

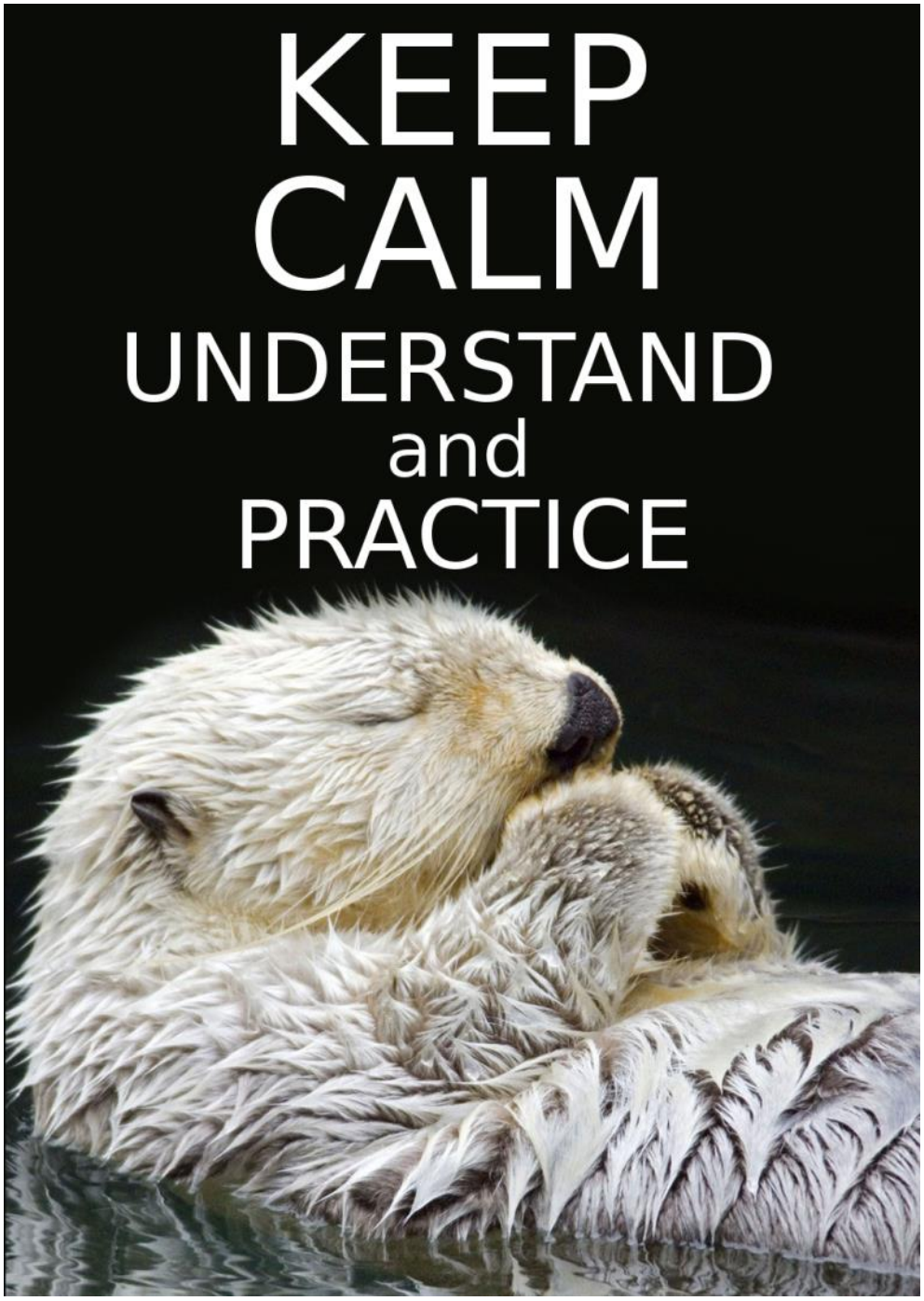
design by contract – part 1

Vladimir Alekseichenko

Good practices make life easier

Good practices are not easy




KEEP
CALM
UNDERSTAND
and
PRACTICE

Agenda

- Ariane 5.
- How to build reliable software?
- Several quote about simplicity.
- About software correctness.
- Hoare logic.
- Strong and weak conditions.
- Class invariants.
- Definition: class correctness.

A contract

Definition: class correctness

The goal for today

understand it.

A class is correct with respect to its assertions if and only if:

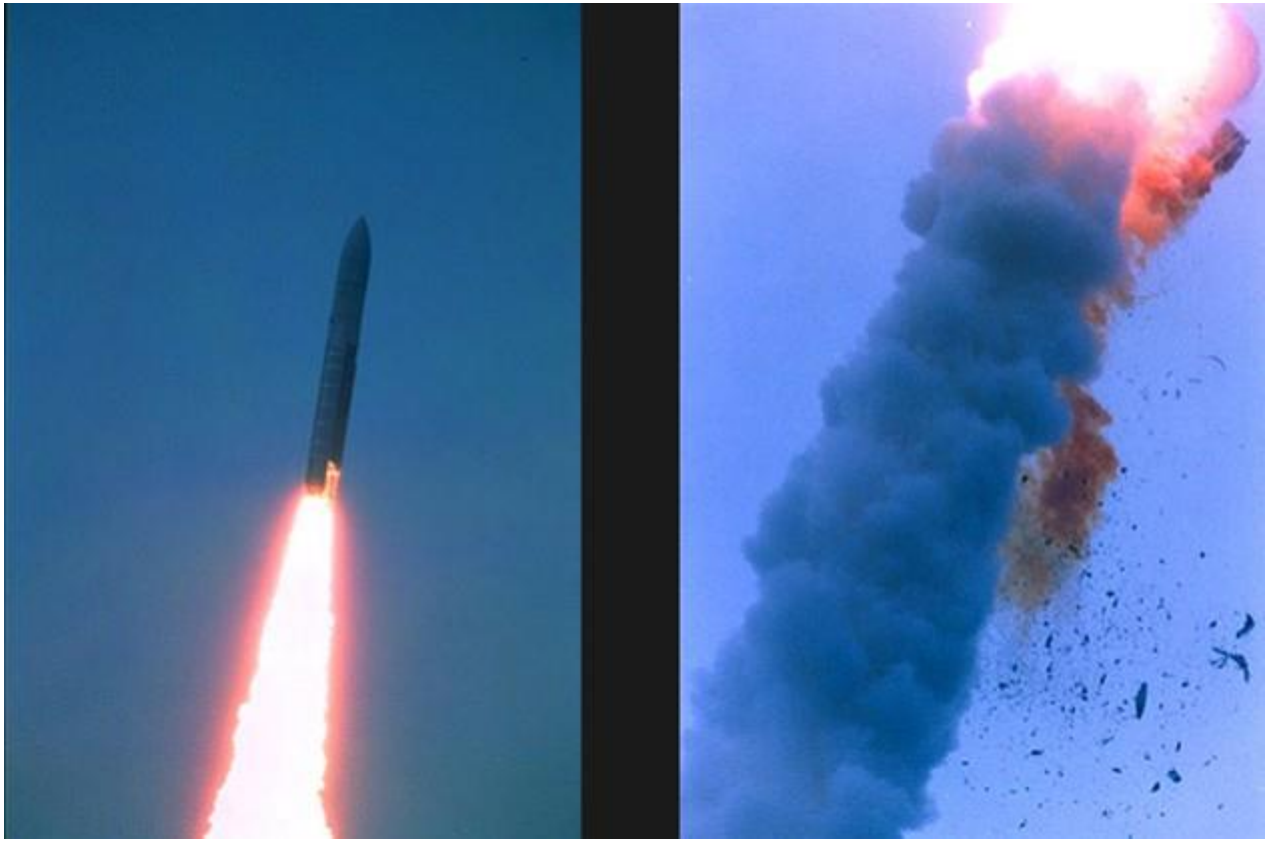
C1 • For any valid set of arguments x_p to a creation procedure p:

$\{Default_C$ **and** $pre_p\ (x_p)\}\ \ Body_p\ \ \{post_p\ (x_p)$ **and** $INV\}$

C2 • For every exported routine r and any set of valid arguments x_r:

$\{pre_r\ (x_r)$ **and** $INV\}\ \ Body_r\ \ \{post_r\ (x_r)$ **and** $INV\}$

Ariane 5



On 4 June 1996, THE MAIDEN FLIGHT OF the Ariane 5 launcher exploded about 37 seconds after liftoff.

Photo: **Photographs by ESA/CNES; ESA**

Ariane 5

What the programmers said:

„The disaster is clearly the result of a programming error".

Ariane 5

convert (horizontal_bias: DOUBLE): INTEGER

… data conversion from **64-bit** floating point to **16-bit** integer.

This procedure work correctly in the rocket **Ariane 4** and has been reused in the **Ariane 5**.

Ariane 5

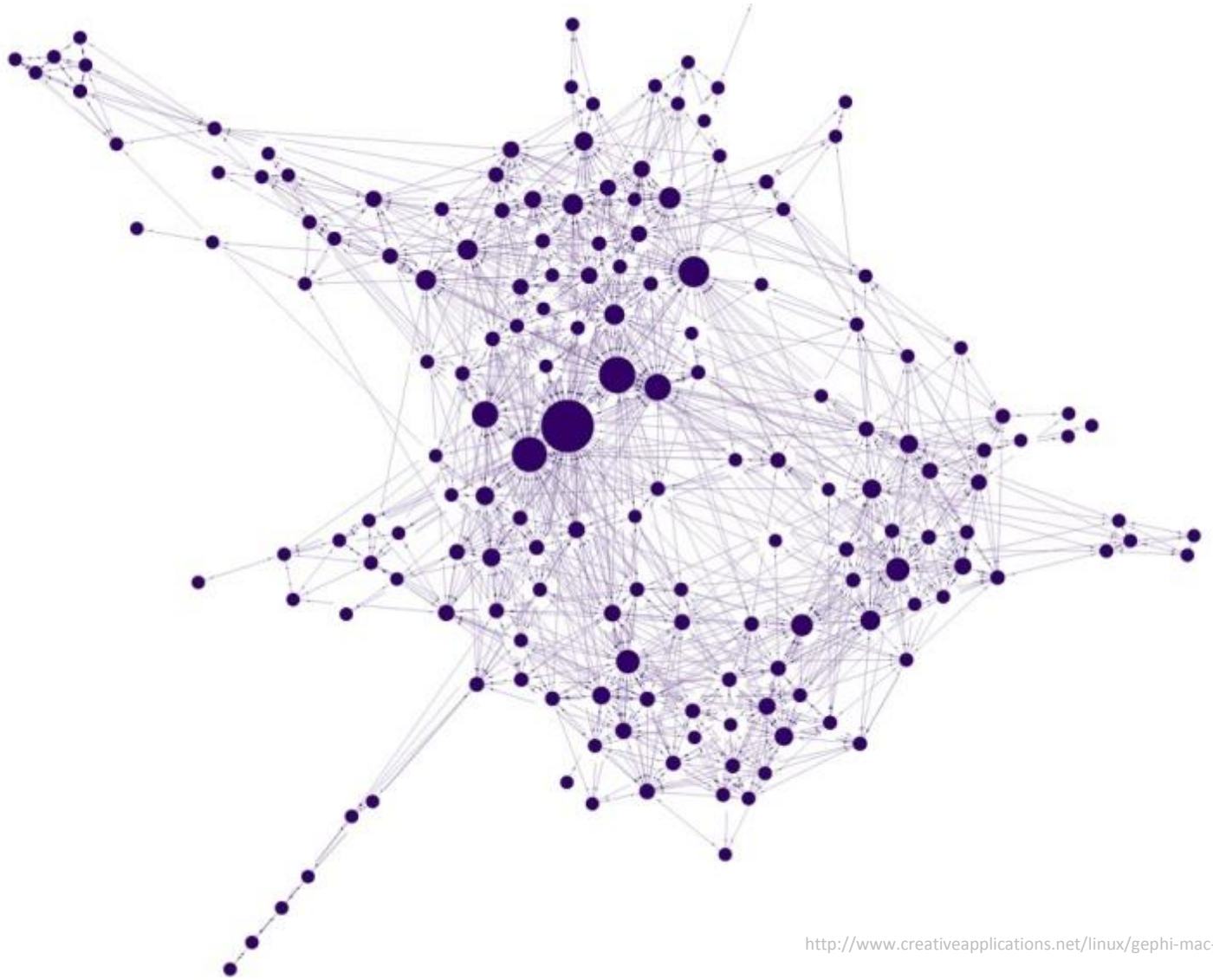
The cause of the failure was the use of **Ariane 5** procedure of unknown semantic **correctness**.

Let's look at it as a Big Picture.

Software architecture

http://www.theophilemorin.com

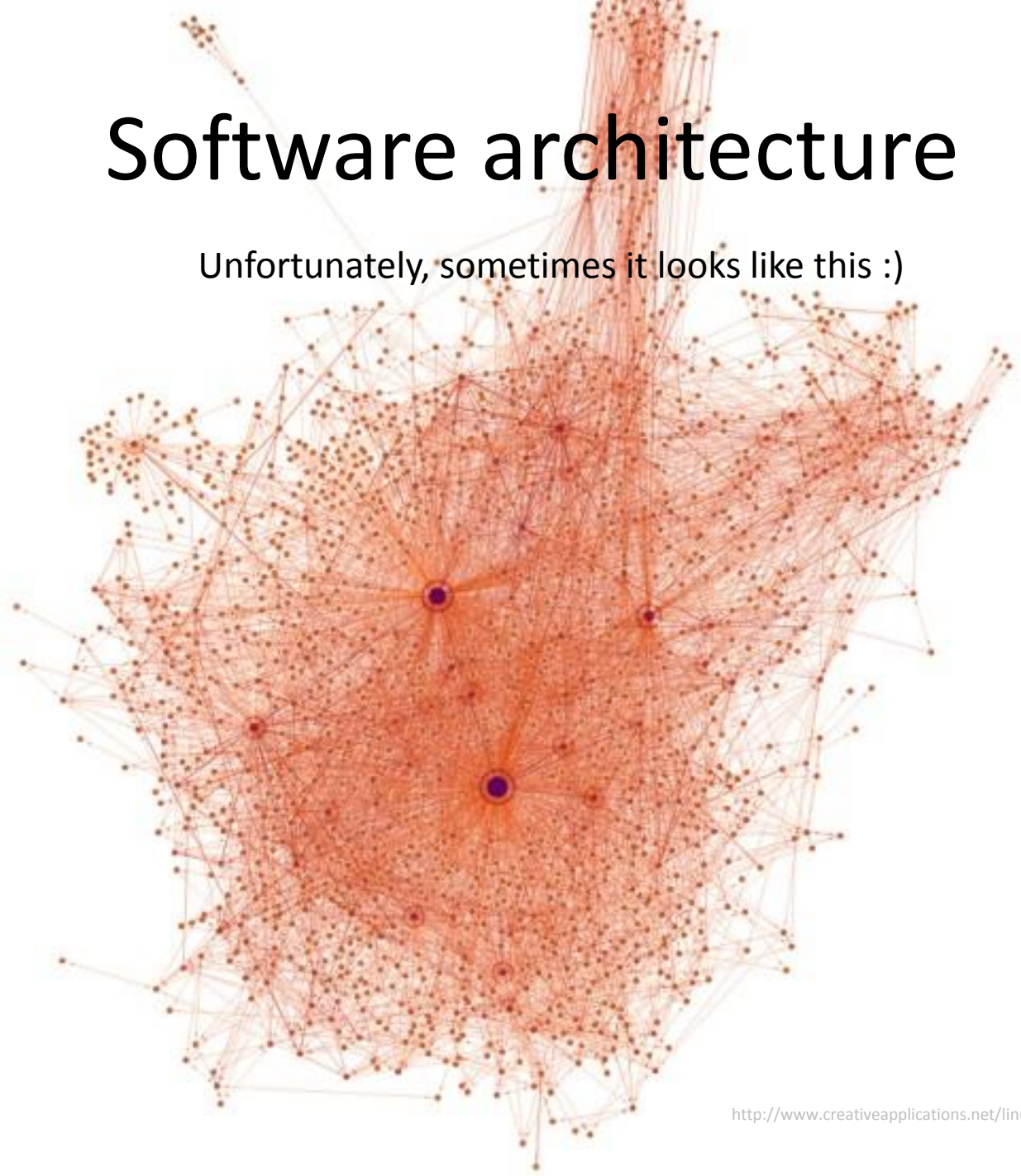
Software architecture



http://www.creativeapplications.net/linux/gephi-mac-windows-linux

Software architecture

Unfortunately, sometimes it looks like this :)



http://www.creativeapplications.net/linux/gephi-mac-windows-linux


How to build reliable software?

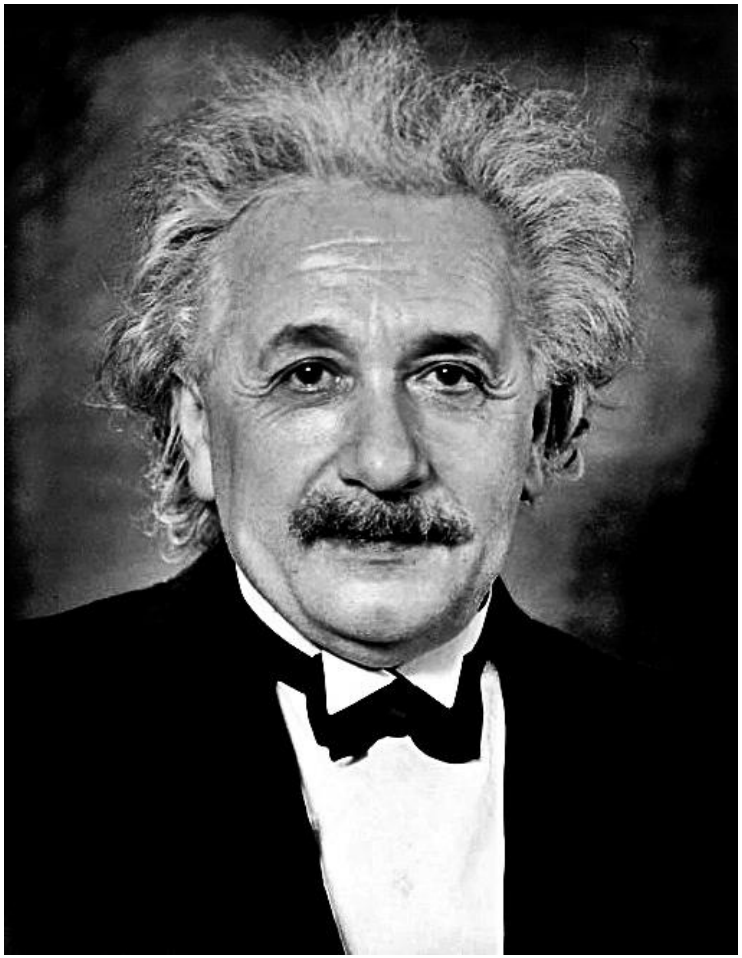
Leonardo da Vinci

http://pl.wikipedia.org/wiki/Leonardo_da_Vinci

Leonardo da Vinci

Simplicity is the ultimate sophistication.

Albert Einstein



http://www.creativeapplications.net/linux/gephi-mac-windows-linux

Albert Einstein

Any **intelligent fool** can make things **bigger**, more **complex**, and more **violent**. It takes a touch of genius - and a lot of courage - to move in the opposite direction.

The detection problem

The **biggest enemy** of **reliability**

(and perhaps of software quality in general)

is **complexity**.

Simple structures

Keeping our **structures** as **simple** as possible is **not enough** to ensure reliability, but it is a necessary condition.

Reliability

It is here defined as the combination of correctness and robustness

or more prosaically, as the absence of bugs.

Harlan Mills

http://www.computerhope.com/people/harlan_mills.htm

What is a correct program?

Writing correct programs does **not** mean that you can write programs **once** and for all.

What is a correct program?

The problem of writing **correct** program **logic** is **more difficult** than that of writing correct **syntax**.

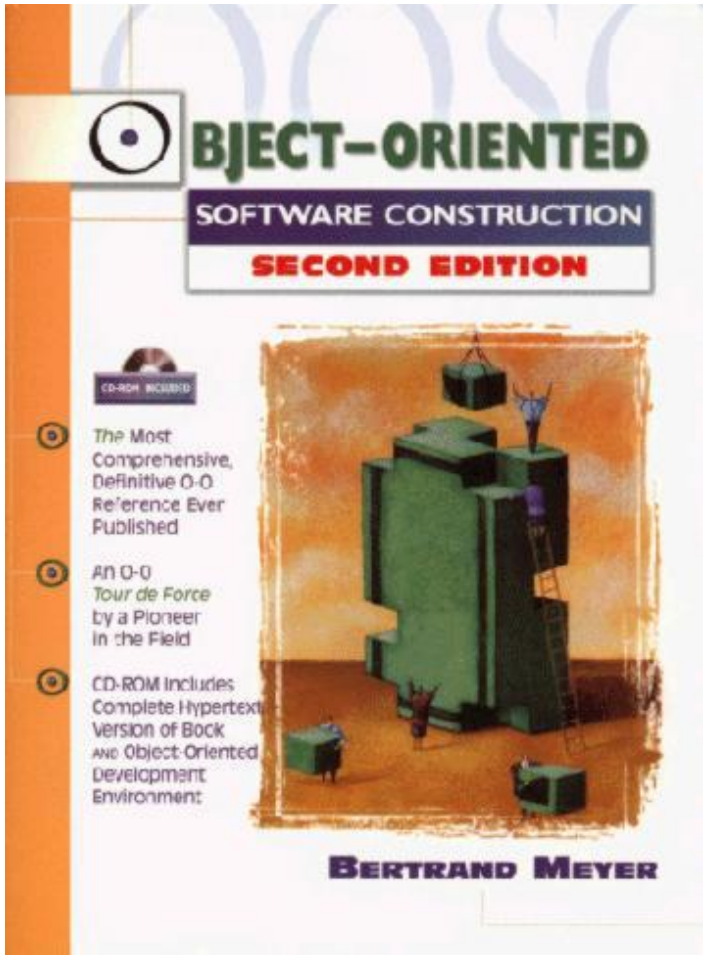
This code is correct or not?

```
public int DoSomething(int x, int y)
{
    int result = x * x / y;
    return result;
}
```

Bertrand Meyer

http://en.wikipedia.org/wiki/Bertrand_Meyer

Object-Oriented Software Construction



http://en.wikipedia.org/wiki/Object-oriented_Software_Construction

About software correctness

To consider the question meaningful, you would need to get not only the program but also a precise description of what it is supposed to do — **a specification**.

The instruction:

$$x := y + 1$$

is neither **correct** nor **incorrect**.

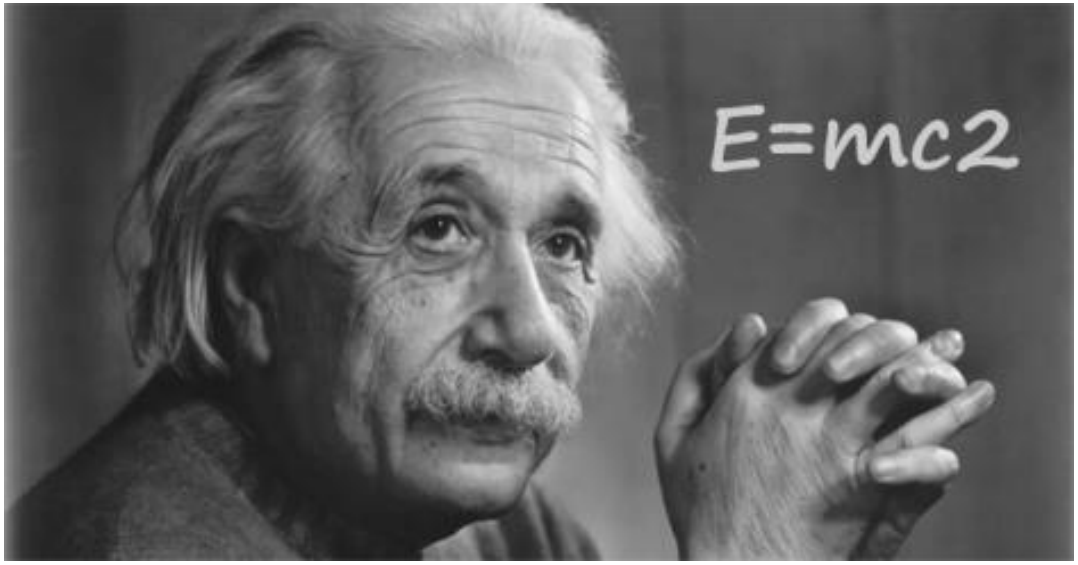
It’s correct for the specification

“Make sure that x and y have different values”

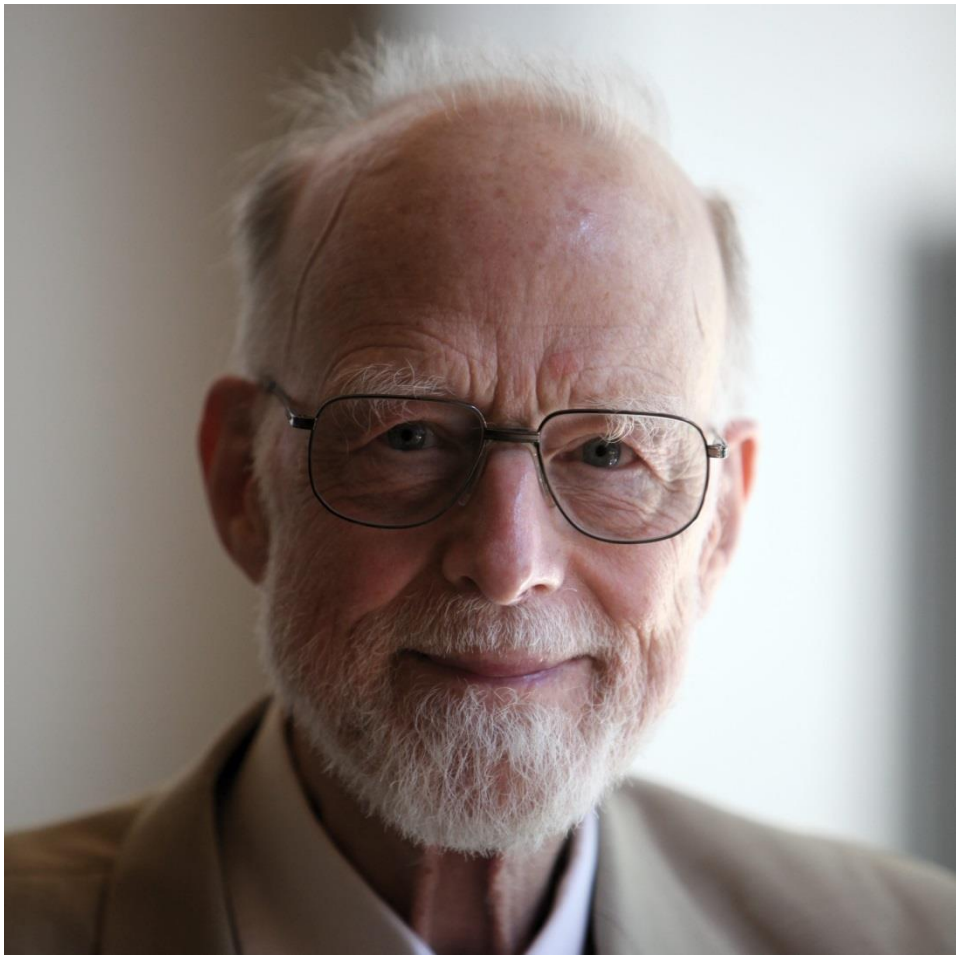
It’s incorrect for the specification

“Make sure that x has a negative value”

Correctness is a relative notion.



Charles R. Hoare



http://en.wikipedia.org/wiki/C._A._R._Hoare

A correctness formula

$$\{P\}\ A\ \{Q\}$$

A correctness formula

{P} A {Q}

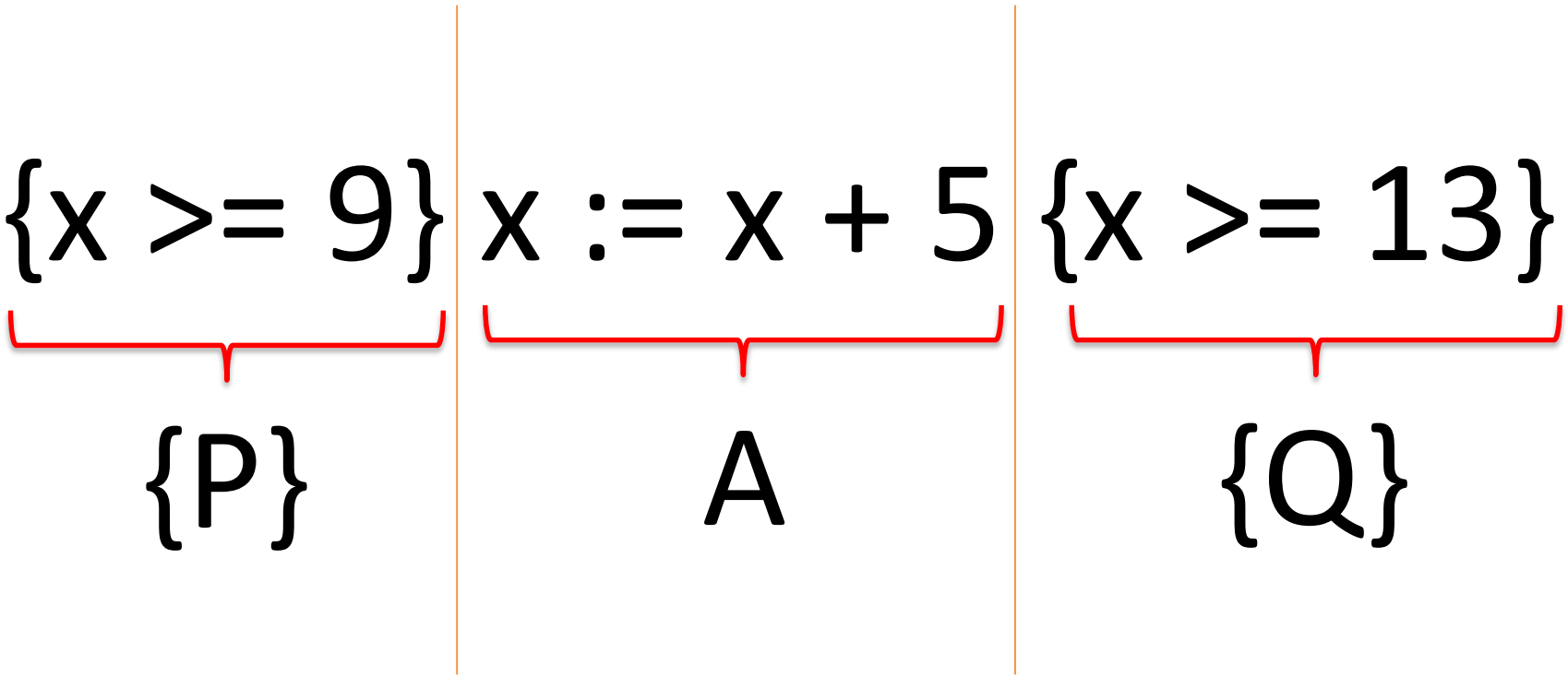
- Where P and Q are assertions and A is a command.
- **P** is named the **precondition** and **Q** the **postcondition**.

Hoare logic

{P} A {Q}

Any execution of A, starting in a state where P holds, will terminate in a state where Q holds.

A trivial correctness formula



Strong and weak conditions

Sinecure 1

{False} A {... }

As long as precondition is **False**
I can rest ☺

Sinecure 2

{... } A {True}


What do I do next?

Rights and obligations

If **you** promise to call **r** with **pre satisfied** then **I**, in return, promise to deliver a final state in which **post is satisfied**.

Assertion Violation rule (1)

A run-time assertion violation is the manifestation of a **bug** in the software.

Assertion violation rule (2)

- A **precondition** violation is the manifestation of a bug in the client.
- A **postcondition** violation is the manifestation of a bug in the supplier.

Precondition design

- The **tolerant** style
- The **demanding** style

The tolerant style

You appoint the supplier, in which case the condition will appear in a conditional instruction of the form **if condition then** ..., or an equivalent control structure, in the routine's body.


WE CAN DO IT
memegenerator.net

The demanding style

You assign the responsibility to clients, in which case the condition will appear as part of the routine's precondition.


We can do it, but ...

The demanding style

A word of caution: the **demanding** approach is only applicable if the **preconditions** remain **reasonable**.

Reasonable **precondition** principle

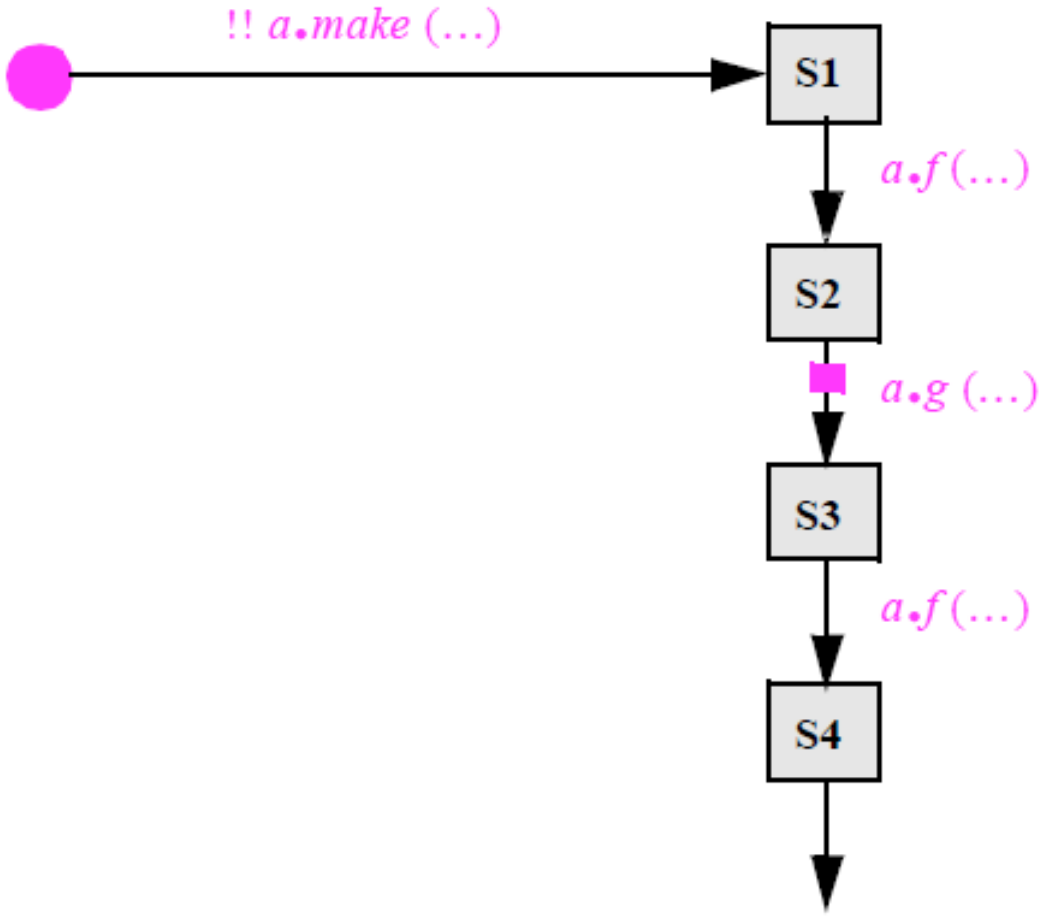
- The precondition appears in the official documentation distributed to authors of client modules.

- It is possible to justify the need for the precondition in terms of the specification only.

Class invariants

- **Preconditions** and **postconditions** describe the properties of individual routines.
- There is also a need for expressing global properties of the instances of a class, which must be preserved by all routines.

The life of an object



Invariants and contracting

{*INV* **and** *pre*} *body* {*INV* **and** *post*}

Any execution of body, started in any state in which ***INV*** *and* ***pre*** *both hold, will terminate in a state in which both* ***INV*** *and* ***post*** *hold.*

Definition: class correctness

A class is correct with respect to its assertions if and only if:

C1 • For any valid set of arguments x_p to a creation procedure p:

$\{Default_C$ **and** $pre_p\ (x_p)\}\ \ Body_p\ \ \{post_p\ (x_p)$ **and** $INV\}$

C2 • For every exported routine r and any set of valid arguments x_r:

$\{pre_r\ (x_r)$ **and** $INV\}\ \ Body_r\ \ \{post_r\ (x_r)$ **and** $INV\}$


Ok.. I almost understood it …

Withdrawing money from an ATM

Precondition:

- Cash > 0;
- Cash ≤ 10000;

Postcondition:

- Balance = old(Balance) – Cash;
- Amount = old(Amount) – Cash;

Invariant:

- Amount ≥ 0;
- Balance ≥ - 1000;

In summary

- Assertions are boolean expressions or predicates that evaluates to **True** or **False** in every state.
- Assertions are used in **preconditions**, **postconditions**, **invariants**.
- A **precondition** and a **postcondition** associated with a routine describe a contract between the class and its clients.
- The **invariant** of a class expresses the semantic constraints on instances of the class.

Resources

- **Object-Oriented Software Construction** by *Bertrand Meyer*.
- **How to write correct programs and know it** by *Harlan Mills* (http://dl.acm.org/citation.cfm?id=808459).
- **An Axiomatic Basis for Computer Programming by** *C.A.R. Hoare*.