

# 1. A Framework for Test and Analysis

Angelo Gargantini

Testing e verifica del software

AA 2015-16

# Goals of Software Testing and Verification

- to assess software **qualities**

## examples of sw qualities

- my program never crashes
- my program works
- my program is useful

- to make it possible to improve the software by finding **defects**

## example of sw defects

- the pointer is not null
- when the user presses OK button, the application ....

## Section 1

# Validation and Verification

## Validation

Does the software system meets the user's real needs? are we building the **right** software?

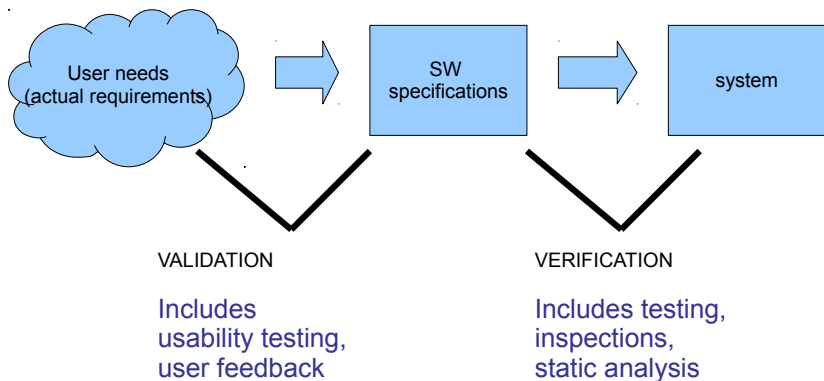
## Specification

A statement (document) about a particular proposed solution to a problem.

## Verification

Does the software system meets the requirements specifications? are we building the software **right**?

# Validation & Verification



# Validation & Verification - standard definitions

IEEE standard in its 4th edition defines the two terms as follows:

**Validation.** The assurance that a product, service, or system meets the needs of the customer and other identified stakeholders. It often involves acceptance and suitability with external customers. Contrast with verification.

**Verification.** The evaluation of whether or not a product, service, or system complies with a regulation, requirement, specification, or imposed condition. It is often an internal process. Contrast with validation.

ISO 9001 standard defines them this way :

**Verification** is the conformation that a product meets identified specifications.

**Validation** is the conformation that a product appropriately meets its design function or the intended use.

Capability Maturity Model (CMMI-SW v1.1):

**Software Verification:** The process of evaluating software to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase.

**Software Validation:** The process of evaluating software during or at the end of the development process to determine whether it satisfies specified requirements.

Boehm succinctly expressed the difference between:

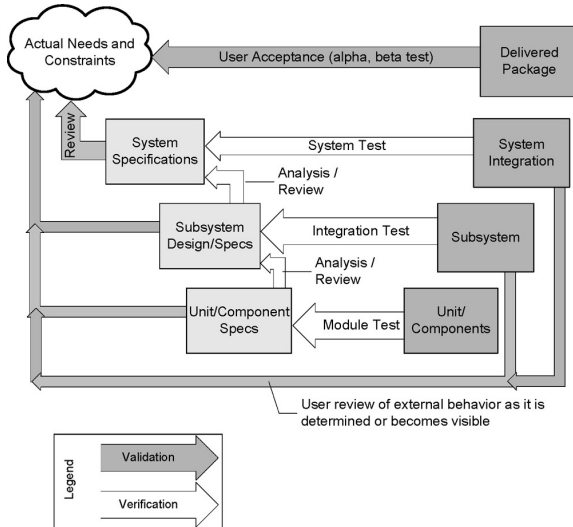
**Software Verification:** Are we building the product right?

**Software Validation:** Are we building the right product?

# Example



# Validation & Verification



- Verification generally compares two or more artifacts
- Verification can consist in checking for *self-consistency* and well-formedness **one** artifact.
  - For example, we can certainly determine that some programs are "incorrect" because they are ill-formed.
  - We may likewise determine that a specification itself is ill-formed because it is inconsistent (requires two properties that cannot both be true) or ambiguous (can be interpreted to require some property or not),
  - or because it does not satisfy some other well-formedness constraint that we impose, such as adherence to a standard imposed by a regulatory agency.

- Validation against actual requirements necessarily involves human judgment
- Verification can be automatized

## Section 2

# Degrees of Freedom

- 1 Can we arrive at some logically sound argument or proof that a program satisfies the specified properties?
- 2 Alan Turing proved that some problems cannot be solved by any computer program.
- 3 **an undecidable problem** is a decision problem for which it is known to be impossible to construct a single algorithm that always leads to a correct yes-or-no answer.
- 4 for instance the **halting problem**
- 5 every interesting property regarding the behavior of computer programs can be shown to "embed" the halting problem,

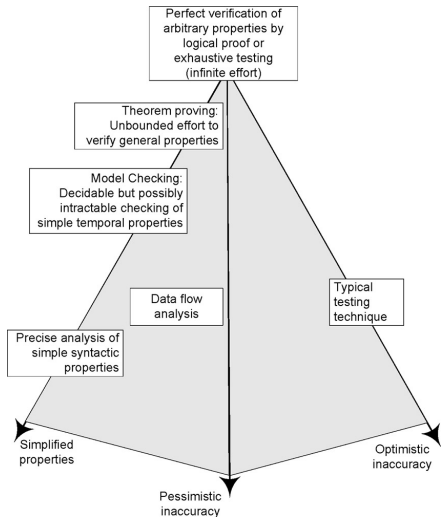
## HALTING PROBLEM

Given the description of an arbitrary program and a finite input, decide whether the program finishes running or will run forever.

```
static int sum(int a, int b) {return a+b;}
```

- 1 Exhaustive testing, that is, executing and checking every possible behavior of a program, would be a "proof by cases," which is a correct way to construct a logical proof. How long would this take?
- 2 there are only  $2^{32} \times 2^{32} = 2^{64} \approx 10^{21}$  different inputs on which the method `Trivial.sum()` need be tested to obtain a proof of its correctness. At one nanosecond ( $10^{-9}$  seconds) per test case, this will take approximately  $10^{12}$  seconds, or about 30,000 years.

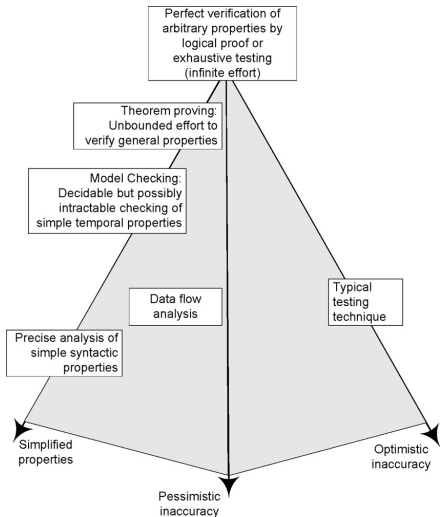
# Pessimistic and Optimistic inaccuracy



A (testing/analysis) technique can be approximate:

- 1 **pessimistic** : it is not guaranteed to accept a program even if the program does possess the property being analyzed
- 2 **optimistic** : if it may accept some programs that do not possess the property (i.e., it may not detect all violations)

# Simplification/abstraction



- 1 we want to verify a property  $S$ , but
  - 1 we cannot accept the optimistic inaccuracy of testing for  $S$
  - 2 precise analysis is too difficult
- 2 a simpler property  $S'$  is a sufficient, but not necessary, condition for  $S$
- 3 we check  $S'$  rather than  $S$
- 4 we require  $S'$  to be satisfied



```
int i, sum;
int first=1;
for (i=0; i<10; ++i) {
    if (first) {
        sum=0; first=0;
    }
    sum += i;
}
```

**Property:** each variable should be initialized with a value before its value is used in an expression

- 1 P vale??
- 2 C ???
- 3 Java ???

# Example of simplified property: Unmatched Semaphore Operations

Property: every semaphore it is eventually unlocked

```
if ( ... ) {  
    ... lock(S);  
}  
...  
if ( ... ) {  
    ... unlock(S);  
}
```

Static checking for match is necessarily inaccurate ...

Java solution: synchronized statements specify the object that provides the intrinsic lock

```
synchronized(S) {  
    ...  
}
```

It is guaranteed that the lock S is released.

# How to deal with undecidable problems

- 1 **optimistic inaccuracy**: we may accept some programs that do not possess the property (i.e., it may not detect all violations).
  - testing
- 2 **pessimistic inaccuracy**: it is not guaranteed to accept a program even if the program does possess the property being analyzed
  - automated program analysis techniques
- 3 **simplified properties**: reduce the degree of freedom for simplifying the property to check

# Some Terminology

**Safe (Sicuro):** A **safe** analysis has no optimistic inaccuracy, i.e., it accepts only correct programs.

- if a program is “wrong” it is rejected.

**Sound (Corretto):** An analysis of a program  $P$  with respect to a formula  $F$  is sound if the analysis returns true only when the program does satisfy the formula.

- if a program is accepted, it is correct
- no wrong program is accepted
- there may be correct programs that are not accepted (conservative)
- testing is not sound.

**Complete (completo):** An analysis of a program  $P$  with respect to a formula  $F$  is complete if the analysis always returns true when the program actually does satisfy the formula.

- every correct program is accepted

## Section 3

# Varieties of Software

## Section 4

### Further Reading

## Section 5

### Exercises