A Framework for Testing and Analysis

資科系
林偉川

Learning objectives

- Introduce dimensions and tradeoff between test and analysis activities
- No perfect test or analysis techniques, nor a single test for all circumstances
- Distinguish validation from verification activities
- Understand limitations and possibilities of test and analysis

Verification and validation

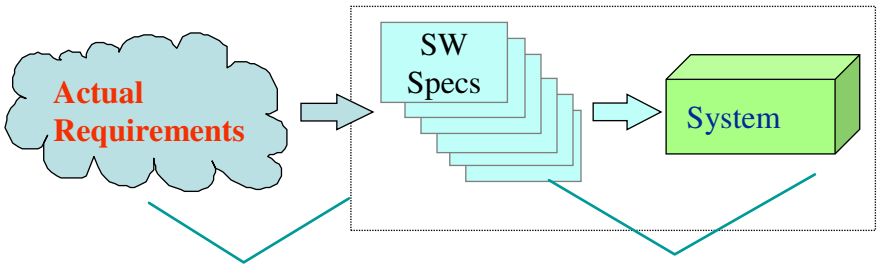
- Verification:
  does the software system meets the requirements specifications?

  are we building the software right?
- Validation:
  does the software system meets the user's real needs?

  are we building the right software?

3

Validation and Verification



Validation

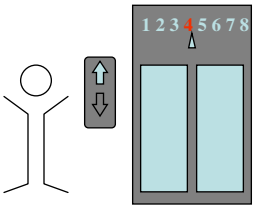
Includes usability testing, user feedback

Verification

Includes testing, inspections, static analysis

4

Verification or validation depends on the specification



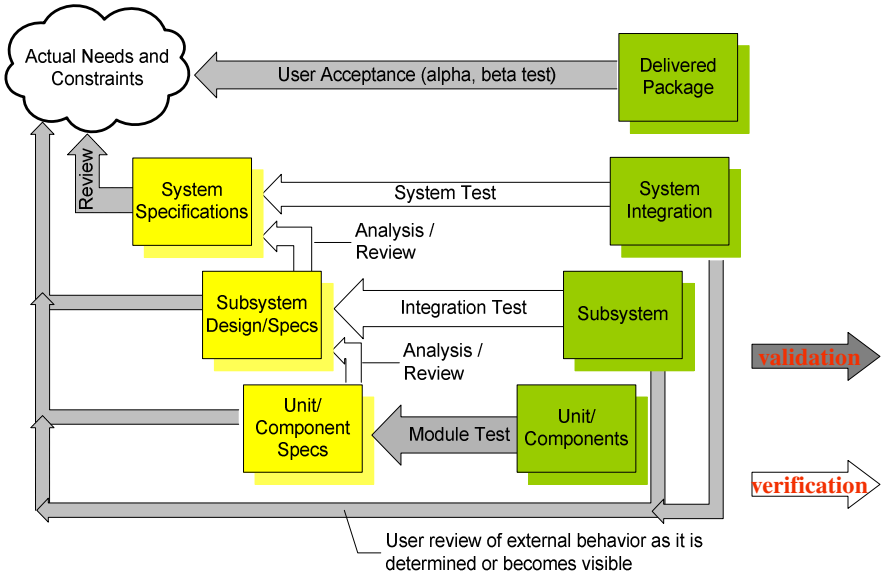
Example: elevator response

Unverifiable (but validatable) spec: ... if a user presses a request button at floor i, an available elevator must arrive at floor i **soon...**

Verifiable spec: ... if a user presses a request button at floor i, an available elevator must arrive at floor i **within 30 seconds...**

5

Validation and Verification Activities



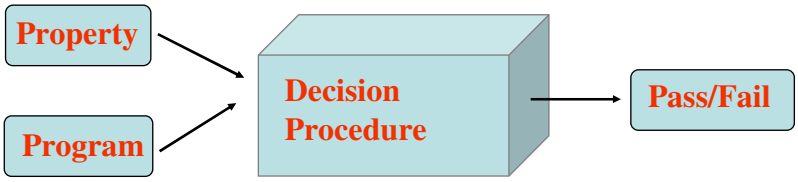
6

V & V

- Verification is checking the consistency of an implementation with a specification
- An overall design could play the role of "specification", a more detailed design could play the role of "implementation". Checking whether the detailed design is consistent with the overall design would be verification of the detailed design.
- In contrast to validation which compares a description (whether a requirements specification, a design, or a running system) against actual needs.

7

You can't ~~always~~ *ever* get what you want



Correctness properties are un-decidable

the halting problem can be embedded in almost every property of interest

8

Undecidability

- Alan Turing proved that some problems cannot be solved by any computer program. The universality of computers – their ability to carry out any programmed algorithm, including simulations of other computers – includes logical paradoxes regarding programs for analyzing other programs
- Logical contradictions ensure from assuming that there is some problem P that can for some arbitrary program Q and input I, determine whether Q eventually halts.

9

Undecidability

- To avoid those logical contradiction, we must conclude that no such program for solving the "halting problem" can possibly exist
- Almost every interesting property regarding the behavior of computer programs can be shown to "embed" the halting problem. The existence of an infallible algorithmic check for the property of interest would imply the existence of a program that solves the halting problem, which we know to be impossible

10

Undecidability

- Undecidability of a property S merely implies that for each verification technique for checking S, there is at least one "pathological" program for which that technique cannot obtain a correct answer in finite time
- It does not imply that verification will always fail or even that it will usually fail, only that it will fail in at least one case
- In practice, failure is not only possible but common, and we are forced to accept a significant degree of inaccuracy

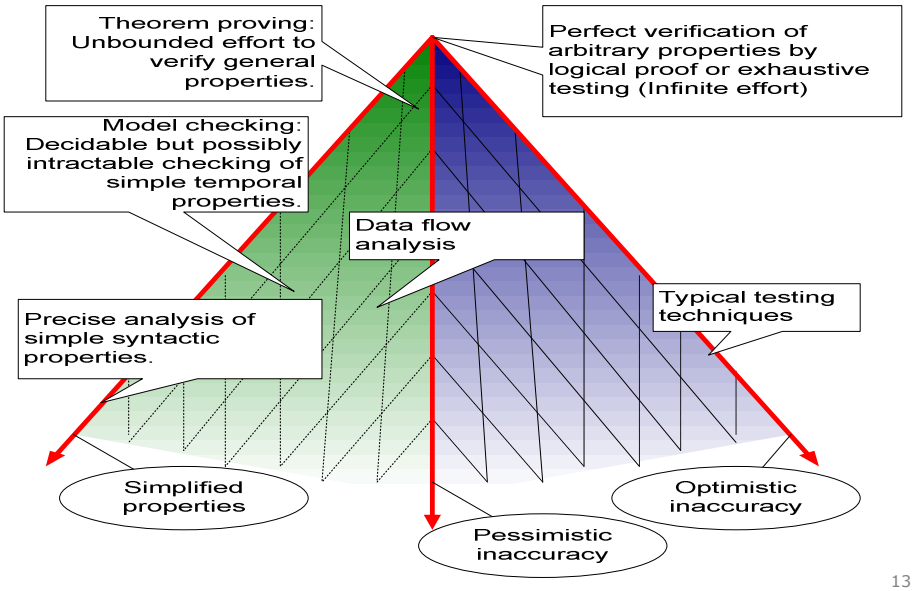
11

Degrees of freedom

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

12

Degrees of freedom



13

Example of simplified property: Unmatched Semaphore Operations

original problem

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

Static checking for match is necessarily inaccurate ...

simplified property

Java prescribes a more restrictive, but statically checkable construct.

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

14

Some Terminology

- **Safe**: A safe analysis has no optimistic inaccuracy, i.e., it accepts only correct programs. ➔ A safe analysis related to a program optimization is one that allows optimization only when the result of the optimization will be correct
- **Sound**: 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 satisfaction of a formula F is taken as an indication of correctness, then a sound analysis is the same as a safe or conservative analysis

15

Some Terminology

- **Complete**: 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. If satisfaction of a formula F is taken as an indication of correctness, then a complete analysis is one that admits only optimistic inaccuracy. An analysis that is sound but incomplete is a conservative analysis

16

Summary

- Most interesting properties are undecidable, thus in general we cannot count on tools that work without human intevention
- Assessing program qualities comprises two complementary sets of activities: validation (does the software do what it is supposed to do?) and verification (does the system behave as specificed?)
- There is no single technique for all purposes: test designers need to select a suitable combination of techniques

17