Software Testing and Verification **Fundamentals**

Sanjai Rayadurgam

SENG 5811 – Spring 2025 – Session 2

University of Minnesota
Driven to Discover®

A refresher

Discrete Mathematics

Recall *a* definition: The process consisting of **all life cycle activities**, both static and dynamic, concerned with planning, preparation, and evaluation of **software** products and **related work products** to determine that they **satisfy specified requirements**, to demonstrate that they are **fit for purpose** and to **detect defects**.

Testing in the most general sense is any **Verification** and **Validation** activity

Verification and Validation: Myers

Verification: An attempt to find errors by executing a program in a test or simulated environment

Validation: An attempt to find errors by executing a program in a real environment

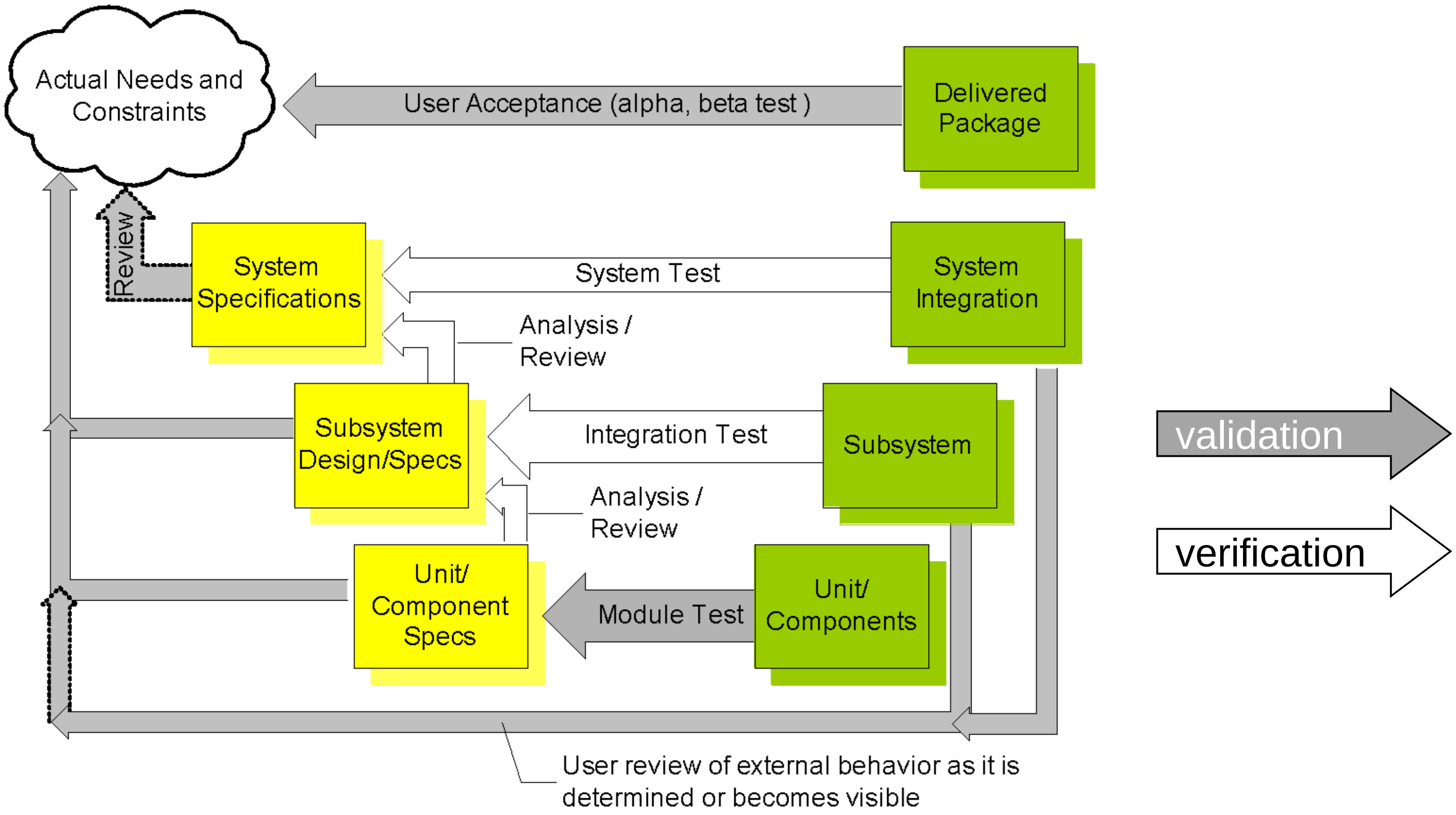
Verification and Validation

Verification: Checking whether the specifications are correctly implemented and whether the product meets its specification.

“Building the thing right” : Does the product meet its spec?

Validation: Checking the program against the “real” requirements and/or user expectations

“Building the right thing”: Does the product fit its purpose?



Actual Needs and Constraints
User Acceptance (alpha, beta test)
Delivered Package
Review
System Specifications
System Test
System Integration
Analysis / Review
Subsystem Design/Specs
Integration Test
Subsystem
Analysis / Review
Unit/ Component Specs
Module Test
Unit/ Components
User review of external behavior as it is determined or becomes visible
validation
verification

Assume that we are building an a patient monitoring system in a hospital setting to alert care provider when a cardiac patient needs immediate attention.

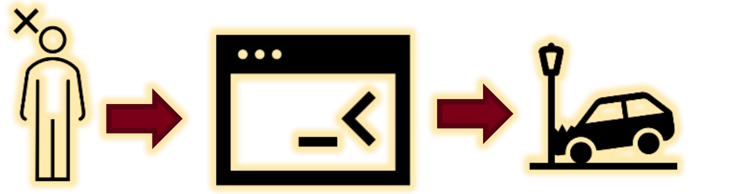
Need versus Spec

Notify the nurse right away in case of a patient emergency

The patient monitoring system should activate audio and visual alarms at the nurse station no later than three seconds after the onset of an "arrhythmia".

```
// pre-condition: input array x is a non-null array of integers
// post-condition: count of positive integers in x is returned

int countPositive (int[] x) {
  int count = 0;
  for (int i = 0; i < x.length; i++)
    if (x[i] >= 0) ++count;
  return count;
}
```



Is there anything wrong with the code?

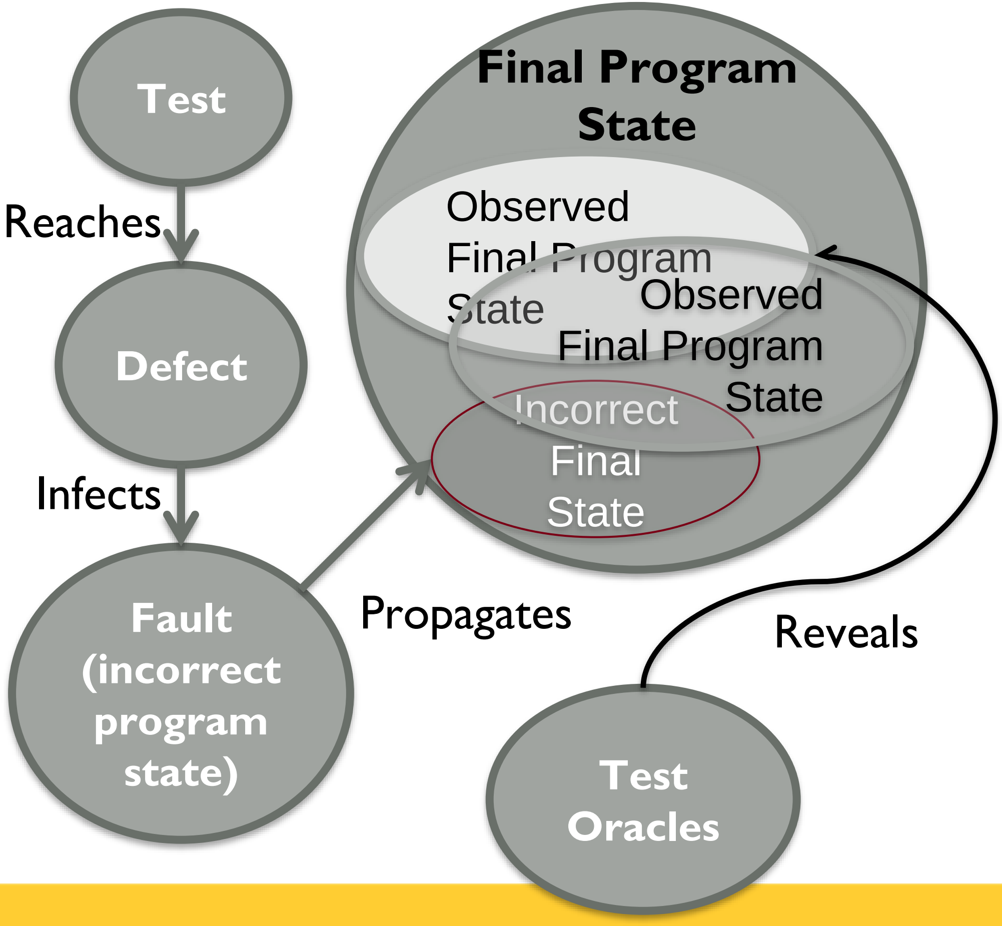
1. Identify a possible error made and the defect in the program
2. Give input data that will cause a failure in this program
3. Give input data that will not cause a failure
4. How should the defect be fixed and (how) can the error be caught early?

Exercise

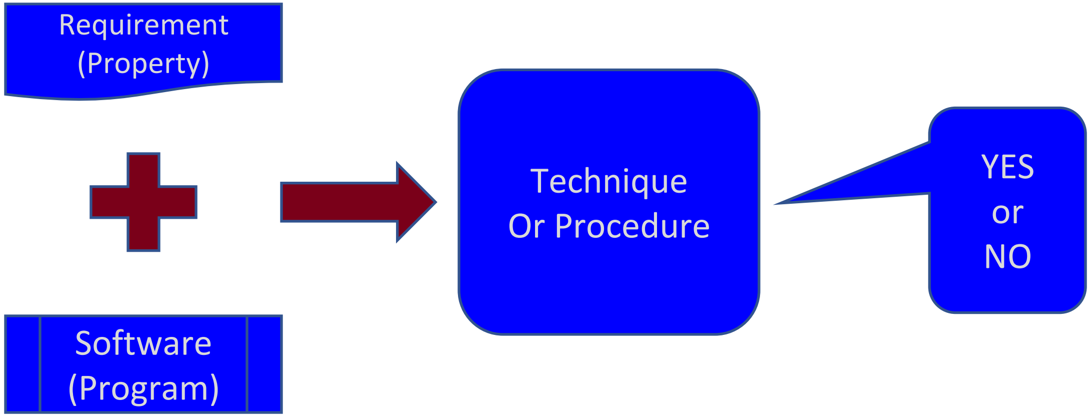
From Amman & Offutt: Introduction to Software Testing, Edition 2 (Ch 2)

RIPR Model

- **R**eachability
- **I**nfection
- **P**ropagation
- **R**evealability



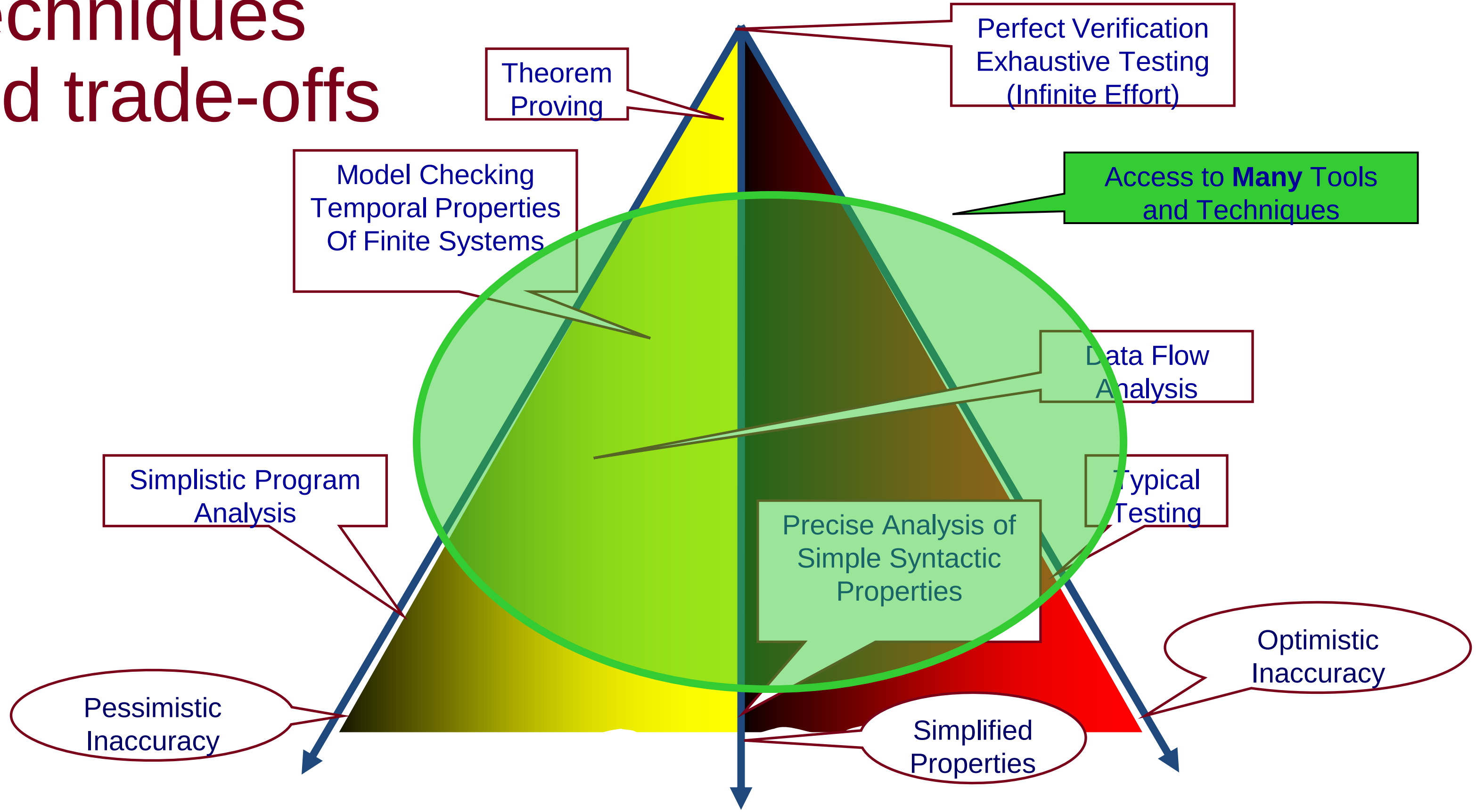
Ideally…



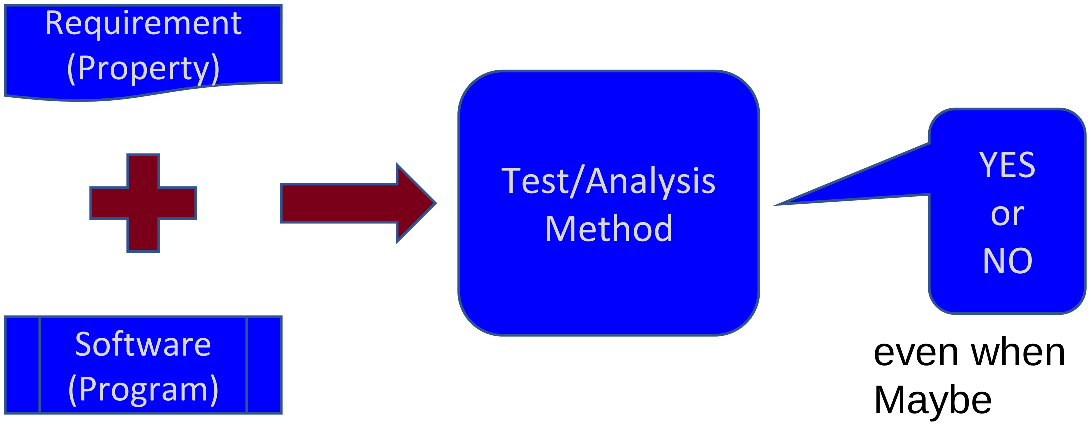
But…

- Correctness properties are undecidable, in general
- We must trade-off accuracy in different dimensions to make the problem tractable

Techniques and trade-offs



In practice…



even when
Maybe

A technique is called:

Sound: Yes means Yes
Optimistic: Yes means Maybe
Pessimistic: No means Maybe
Complete: No means No

Interlude

What is wrong with this code?

It is supposed to find the year given the number of days since 1/1/1980 ?

```
year = 1980;

while (days > 365) {
  if (IsLeapYear(year)) {
    if (days > 366) {
      days -= 366;
      year += 1;
    }
  } else {
    days -= 365;
    year += 1;
  }
}
```

Good Engineering Principles

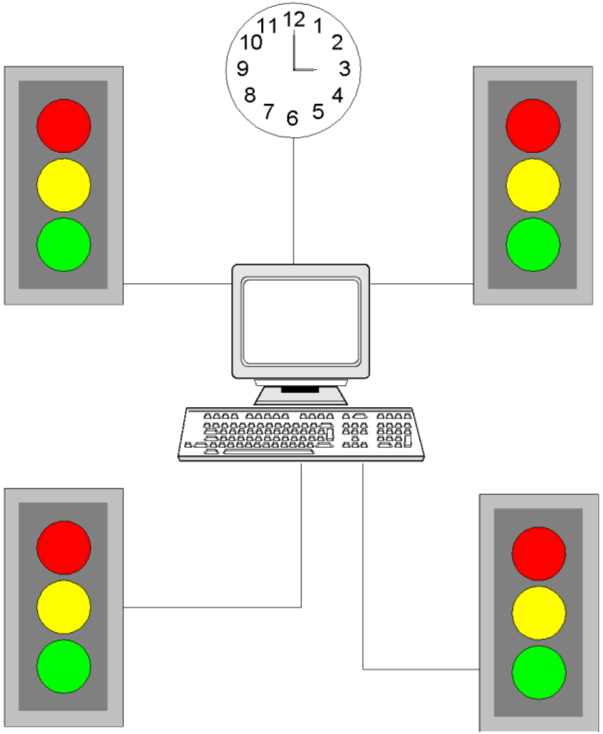
- For any engineering activity
  - **Partition**: divide and conquer
  - **Visibility**: making information accessible
  - **Feedback**: tuning the development process

- Specific to Analysis and Test
  - **Sensitivity**: better to fail every time than sometimes
  - **Redundancy**: making intentions explicit
  - **Restriction**: making the problem easier

Dependability Qualities

- Correctness
  - consistent with specification
    - Hard for non-trivial systems
- Reliability
  - likelihood of correct function for some ``unit'' of behavior
    - relative to a specification and usage profile
    - statistical approximation to correctness (100% reliable = correct)
- Safety:
  - preventing hazards
- Robustness
  - acceptable (degraded) behavior under extreme conditions

Example of Dependability Qualities



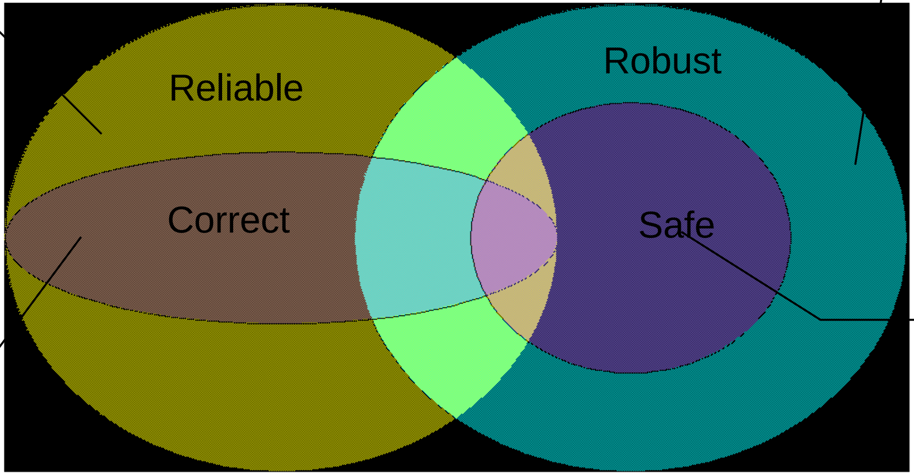
Correctness, reliability: let traffic pass according to correct pattern and central scheduling

Robustness, safety: Provide degraded function when possible; never signal conflicting greens.

Blinking red / blinking yellow is better than no lights; no lights is better than conflicting greens

How they are related

reliable but not correct: failures occur rarely



robust but not safe: catastrophic failures can occur

correct but not safe or robust: the specification is inadequate

safe but not correct: annoying failures can occur

University of Minnesota

Driven to Discover®

Crookston Duluth Morris Rochester Twin Cities