Testing Concepts

Venkatesh-Prasad Ranganath
Kansas State University

Slides with * in title capture whiteboard content

Basics

- What is testing?
- What is the purpose of testing?
- How do you accomplish testing?

Basics*

- What is testing?
  - Ensuring a program runs correctly
    - Handle correct input by producing correct output
    - Handle invalid input by producing correct output
      - Correctness guided by Specification / Requirements
  - Checking for faults
  - Handling invalid inputs
  - Meeting the criteria of requirements
  - Checking for bugs
  - Making sure it fails properly

Basics*

- What is testing?
  - Checking for unexpected behaviors
  - Minimizing bad behaviors
  - Checking expected output for variety of inputs
  - Assurance from faulty code from not running how it is supposed to
  - Checking requirements have been met
  - Checking invalid inputs are handled appropriately

Basics*

- What is the purpose of testing?
  - Checking if requirements are met
  - Ensuring (checking?) quality of software
    - correctness, performance, reliability, …
  - Reducing maintenance cost of software
  - Way of guaranteeing the client software behaves as intended

Basics*

- What is the purpose of testing?
  - Companies are forced to :)
  - Meeting regulations :)
  - You want the product to be used
    - We use buggy products all the time :)
- Code is not faulty / buggy
- Ensuring user satisfaction (??)

Basics

- What is testing?
  - Checking if actual outcome is the expected outcome
- What is the purpose of testing?
  - Detect failures/errors/deviations *(this is fuzzy)*
  - Prove the presence of bugs (faults)
  - *What about proving the absence of bugs?*
- How do we accomplish testing?
  - Often by comparing two entities for equality

Fault, Error, and Failure*

1. Error: Exception thrown to handle certain behavior
2. Fault/Failure: Function does not work as expected
3. Error: Result of an external module not working as desired
4. Failure: An issue that causes unexpected stoppage
5. Error: State of program that results due to fault/failure
6. Fault: Mistake caused by programmer
7. Fault: Program operating sub-optimally

Fault, Error, and Failure

compute(s) = $(s + 1)^2 \bmod 3 + 12$

Fault, Error, and Failure

compute(s) = $(s + 1)^2 \bmod 3 + 12$

```
def compute(s):
    s = s + 1   #1
    s = s * 2   #2
    s = s % 3   #3
    s = s + 12 #4
    return s
```

- Software Testing: Concepts and Operations by Ali Mili and Fairouz Tchier (Section 6.1)

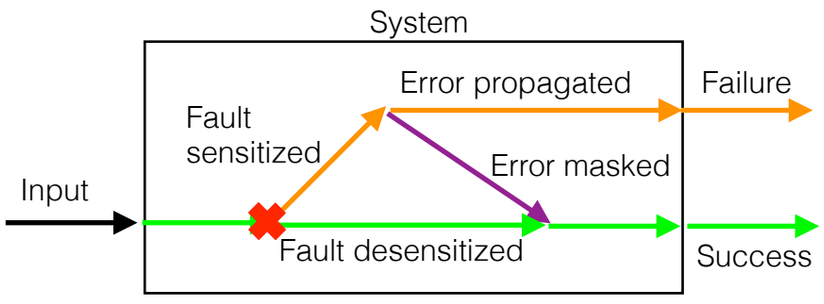
Fault, Error, and Failure

Steps	Correct / Incorrect	Correct / Incorrect	Correct / Incorrect
input	s ->1 / s -> 1	s -> 2 / s -> 2	s -> 3 / s -> 3
#1	s -> 2 / s -> 2	s -> 3 / s -> 3	s -> 4 / s -> 4
#2	s -> 4 / s -> 4	s -> 9 / s -> 6	s -> 16 / s -> 8
#3	s -> 1 / s -> 1	s -> 0 / s -> 0	s -> 1 / s -> 2
#4	s -> 13 / s -> 13	s -> 12 / s -> 12	s -> 13 / s -> 14
	• Fault desensitized	• Fault sensitized • Error masked	• Fault sensitized • Error propagated • Failure observed

Fault, Error, and Failure

compute(s) = $(s + 1)^2 \bmod 3 + 12$

```
def compute(s):
    s = s + 1   #1
    s = s * 2   #2
    s = s % 3   #3
    s = s + 12  #4
    return s
```



- *Fault* is the mistake (or cause of error).
- Executions may *sensitize* faults.
- *Error* is the impact of the fault on program states; *Error* is the difference between *correct and incorrect states*.
- Executions may *mask or propagate* errors.
- *Failure* is the event when the *program behaves incorrectly*, i.e., violates the *specification* (observed outcome differs from the expected outcome).

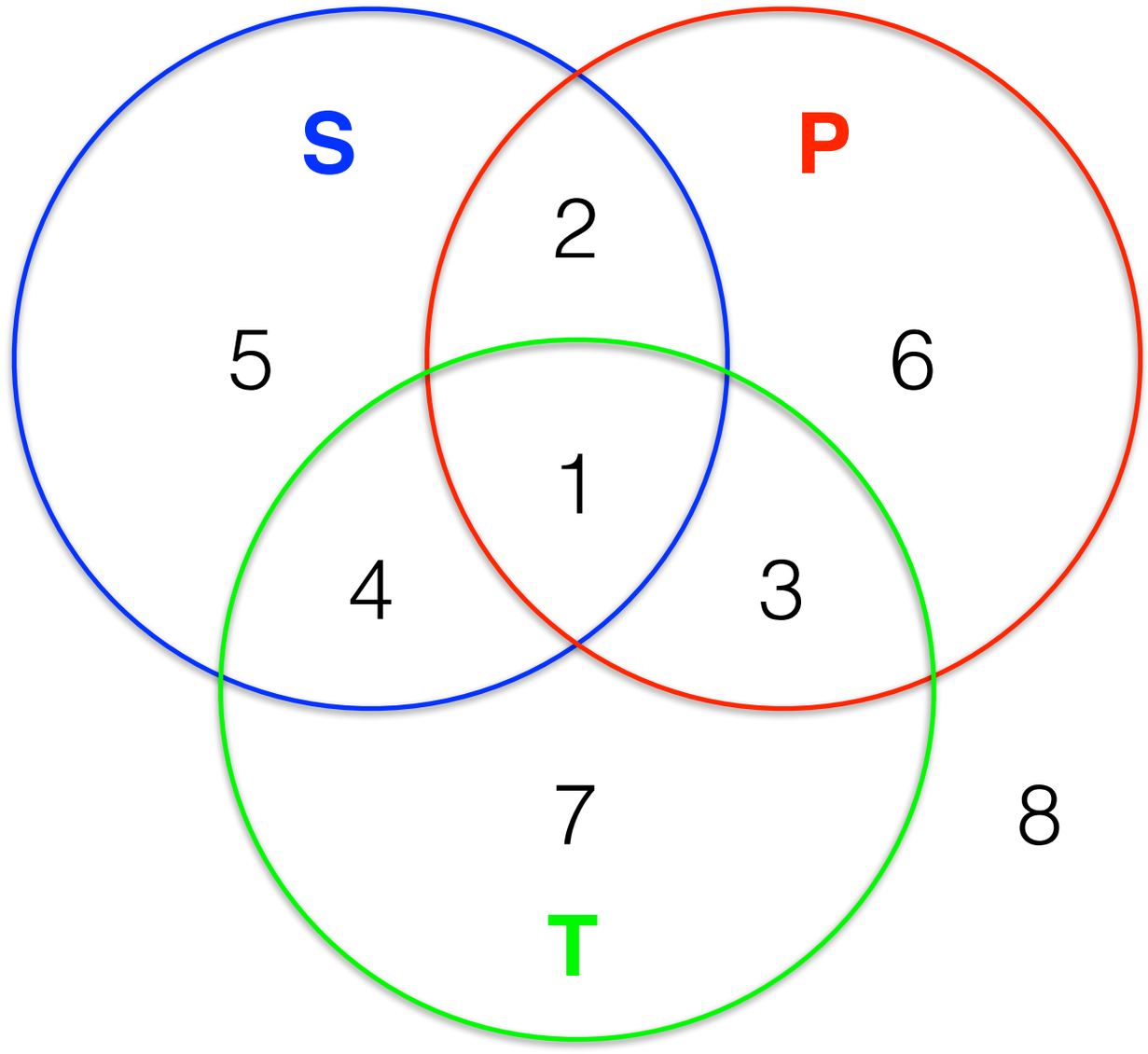
- Software Testing: Concepts and Operations by Ali Mili and Fairouz Tchier (Section 6.1)
- *Basic Concepts and Taxonomy of Dependable and Secure Computing* by Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. (Section 2.2)

Specification, Program, and Test

- *Specification (S)* describes what is to be accomplished.
  - sort an array of ints
  - check if a given number exists in an array of ints
- *Program (P)* describes/embodies how something is accomplished.
  - an implementation using quick sort
  - an implementation using binary search
- *Test (T)* checks parity between specification and implementation.

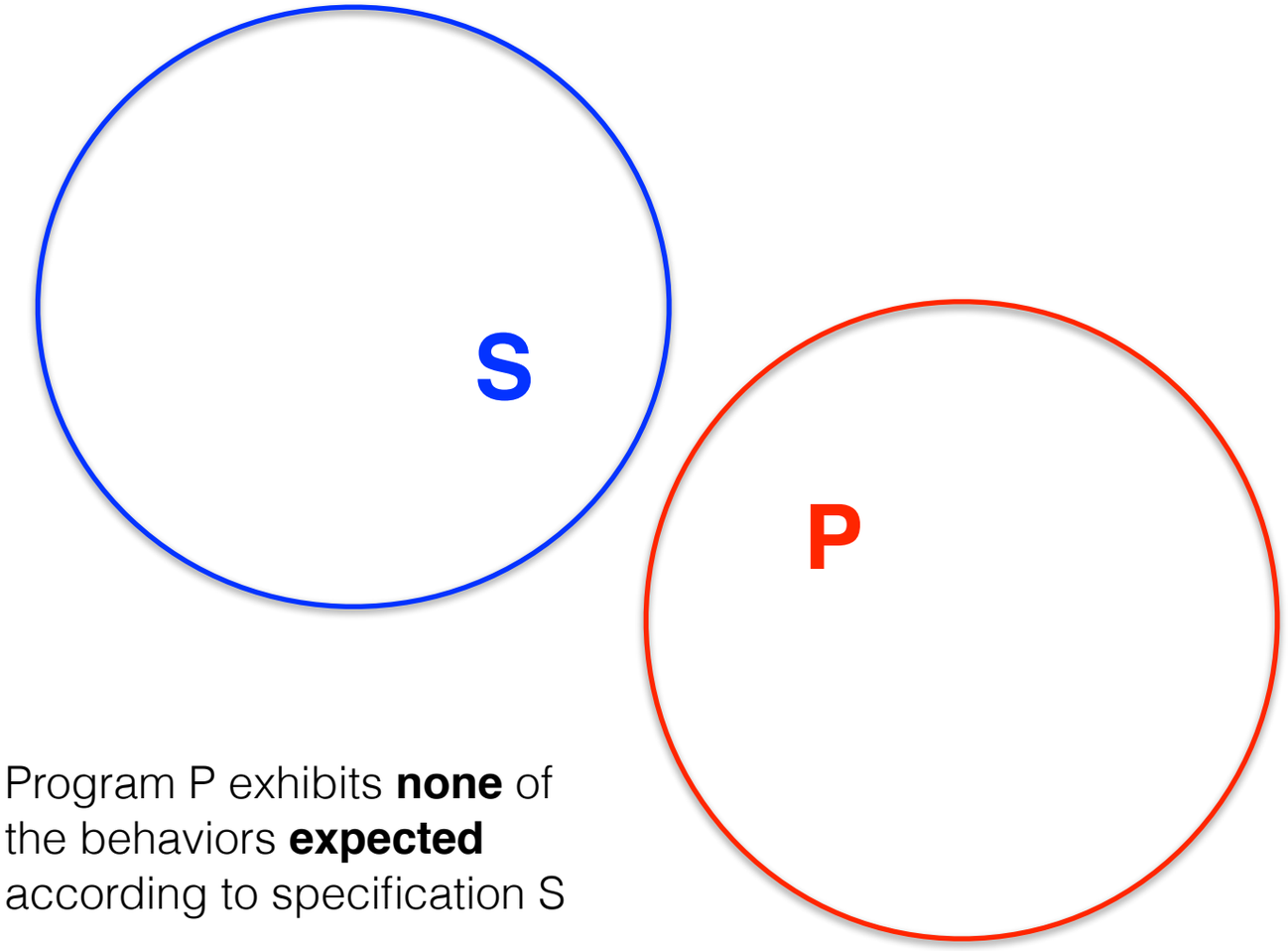
- Software Testing: A Craftsman's Approach, 4ed, by Paul C Jorgensen. (Chapter 1)

Specification, Program and Test: How are they related?



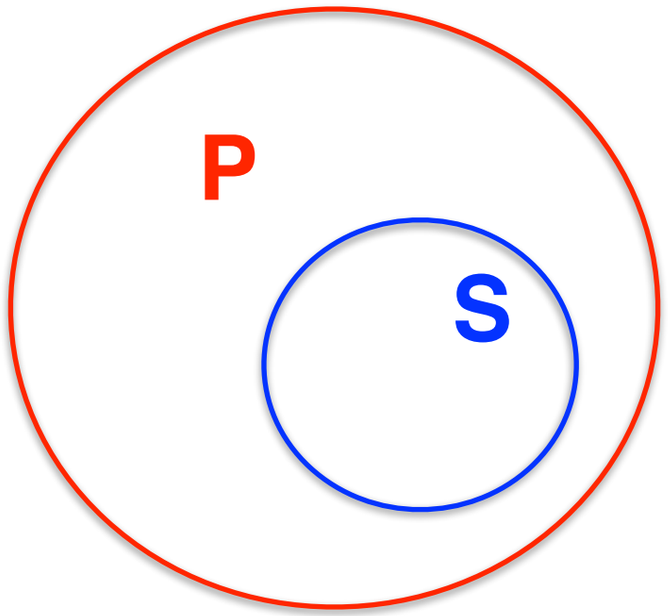
- Software Testing: A Craftsman's Approach, 4ed, by Paul C Jorgensen. (Chapter 1)

Specification and Program: How are they related?

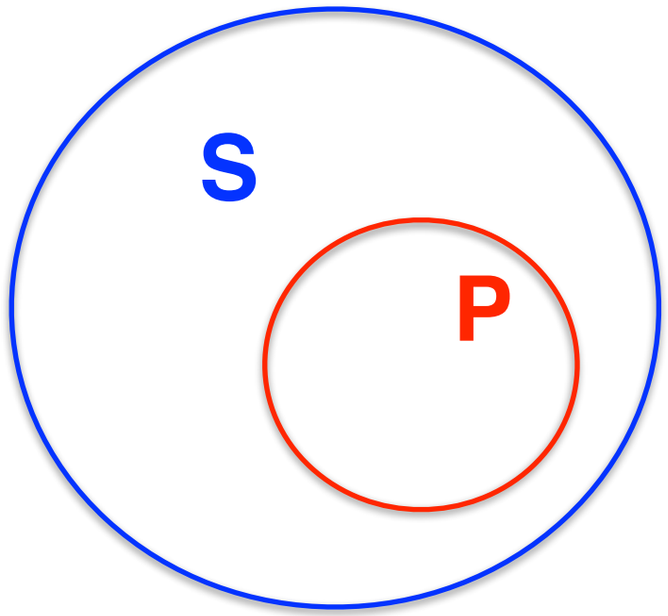


- Program P exhibits **none** of the behaviors **expected** according to specification S

Specification and Program: How are they related?

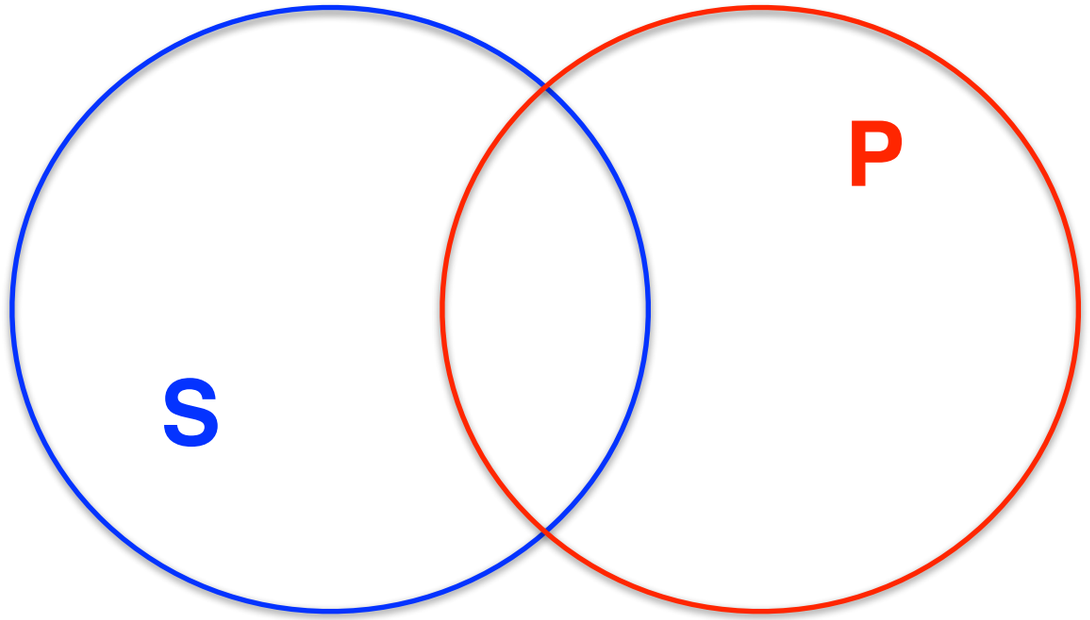


- Program P exhibits **all** of the behaviors **expected** according to specification S
- **Some** behaviors of program P are **not expected** behaviors according to specification S

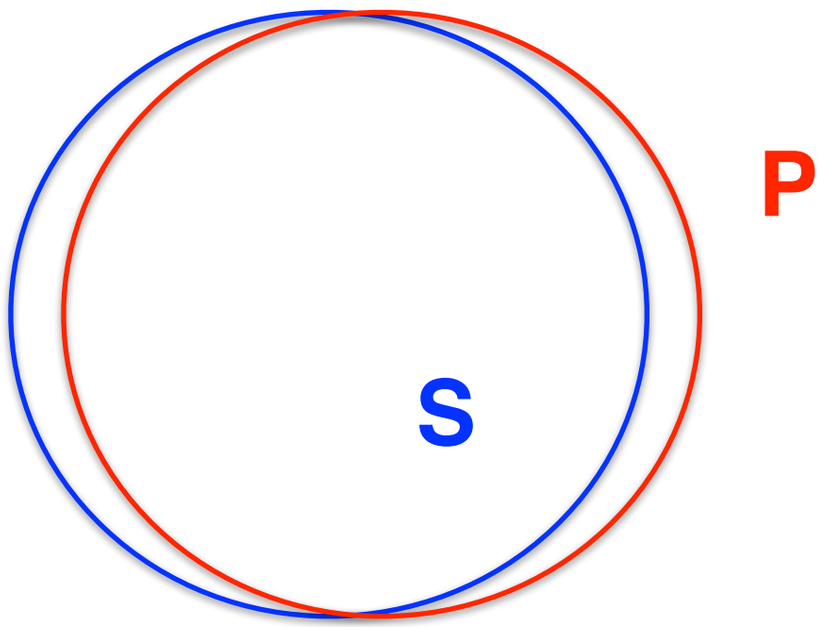


- Program P exhibits **some** of the behaviors **expected** according to specification S
- **All** behaviors of program P are **expected** behaviors according to specification S

Specification and Program: How are they related?

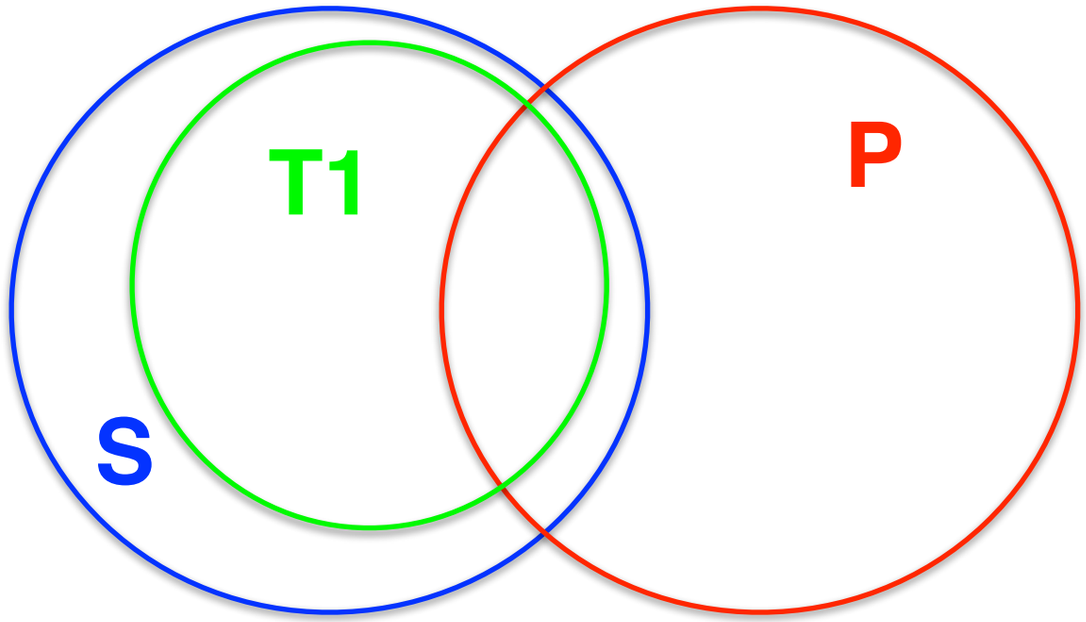


- **Some** behaviors of program P are **expected** according to specification S
- **Most** behaviors of program P are **not expected** behaviors according to specification S



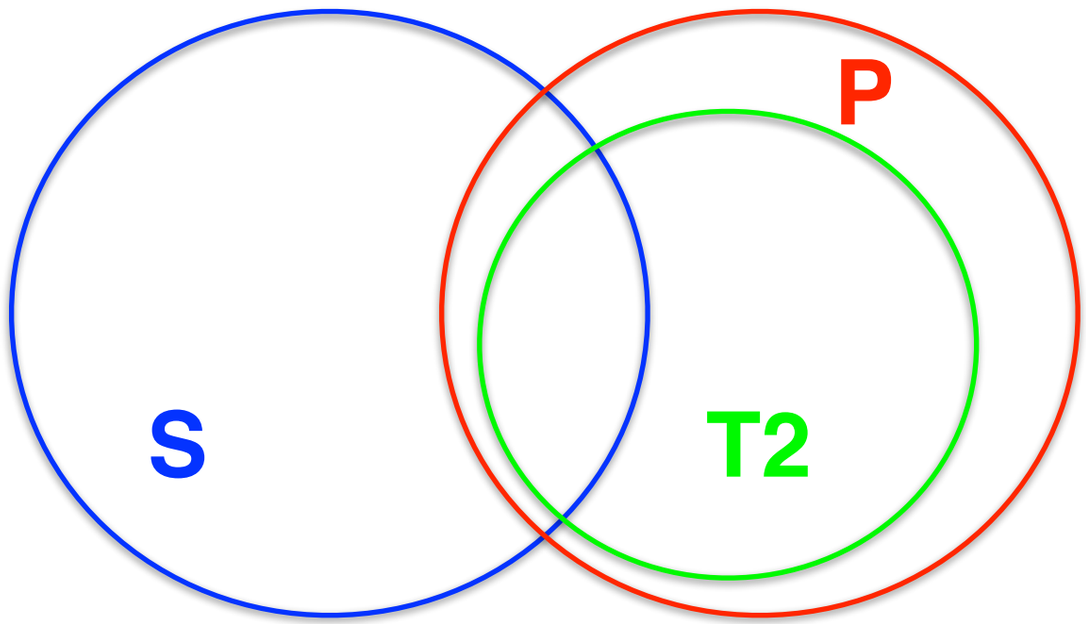
- **Most** behaviors of program P are **expected** according to specification S
- **Some** behaviors of program P are **not expected** behaviors according to specification S

Specification, Program, and Test: How are they related?



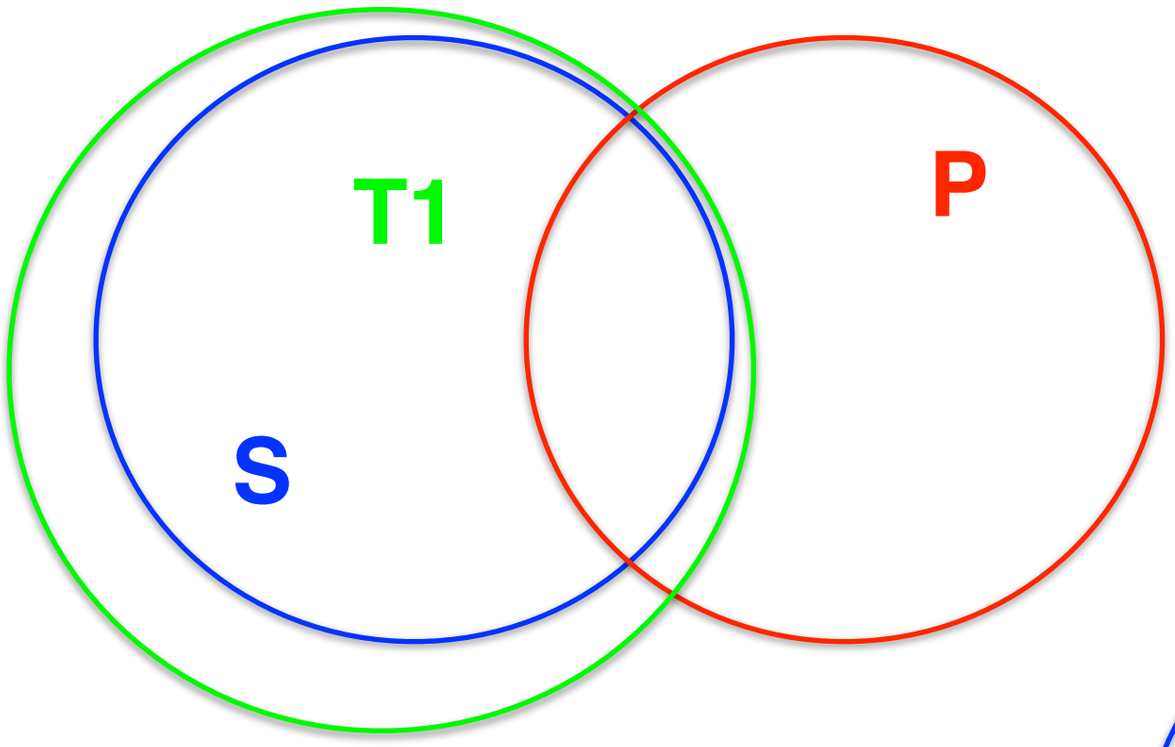
- Test T1 tests for **most** of the behaviors **expected** according to specification S
- Test T1 tests for **some** of the behaviors **exhibited** by program P

- Test T2 tests for **some** of the behaviors **expected** according to specification S
- Test T2 tests for **most** of the behaviors **exhibited** by program P



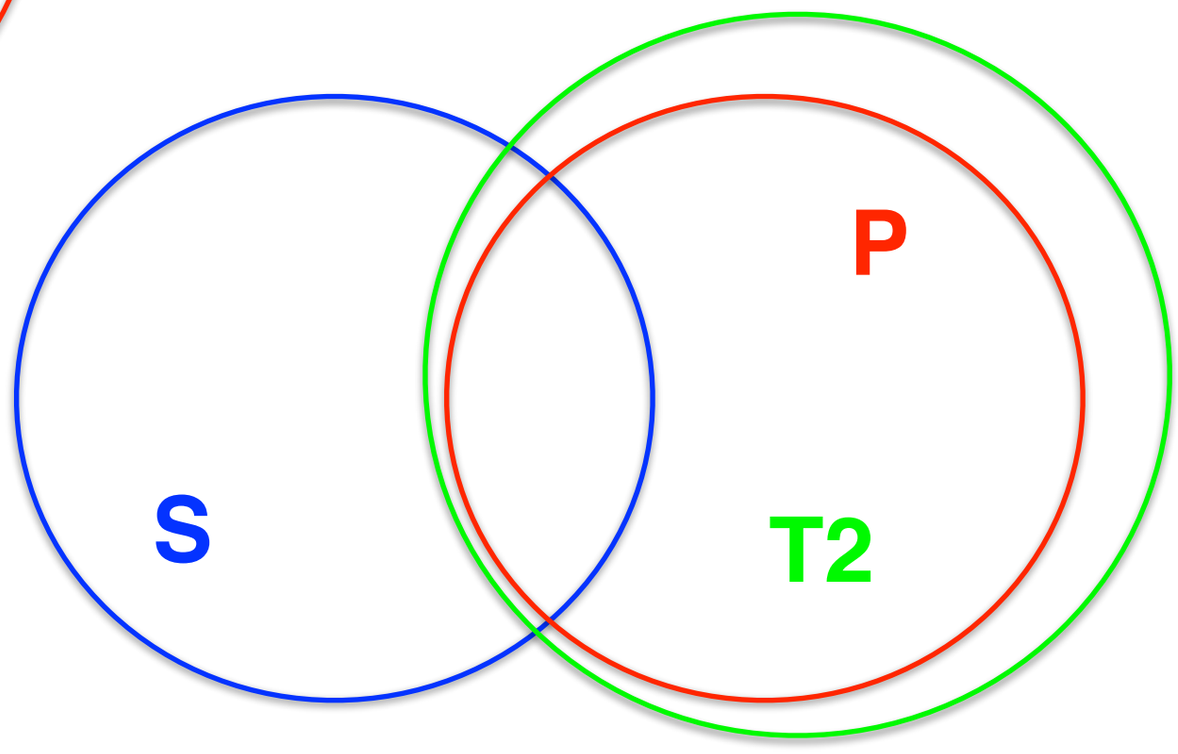
- Software Testing: A Craftsman's Approach, 4ed, by Paul C Jorgensen. (Chapter 1)

Specification, Program, and Test: How are they related?



- Test T1 tests for **all** of the behaviors **expected** according to specification S **and much more**
- Test T1 tests for **some** of the behaviors **exhibited** by program P

- Test T2 tests for **some** of the behaviors **expected** according to specification S
- Test T2 tests for **all** of the behaviors **exhibited** by program P **and much more**



Few Questions to Consider

- Can we prove the presence or absence of bugs with testing?
- What about fault/failure by commission and fault/ failure by omission?
- Can one fault lead to more than one error?
- Can one fault lead to more than one failure?
- Can the same error stem from different faults?
- Can the same failure stem from different faults?

Few Questions to Consider

- Can we prove the presence or absence of bugs with testing?
  - Testing can prove the presence of bugs — a failing test is evidence that the program can fail.
  - In general, testing cannot prove the absence of bugs — a passing test is not evidence that the program will not fail.
- What about fault/failure by commission and fault/failure by omission?
  - *Fault/Failure by commission* stems from an incorrect implementation, e.g., using `x*1` instead of `x+1`.
  - *Fault/Failure by omission* stems from the absence of correct implementation, e.g., absence of line 3 in compute().
- What about *different kind of faults*?

A detour into costs

Lifetime Cost	Software Engg %	Other Engg %
Design	>99	<1
Manufacturing	<1	>99

Lifetime Cost	Software Engg %	Other Engg %
Development	~50	>99
Testing	~50	<1

Maintenance Cost	Software Engg %	Other Engg %
Corrective	~20	>99
Adaptive	~80	<1

(Corrective) Mntn Cost	Software Engg %	Other Engg %
Design	~100	~1
Wear and Tear	~0	~99

- Software Testing: Concepts and Operations by Ali Mili and Fairouz Tchier (Section 6.1)

A detour into costs

- Possible reasons for the differences
  - Extent of reuse
  - Extent of repeatability
  - Extent of automation
  - Limited quality control
  - Digital vs Physical
  - Build one instance vs build multiple instances

- Software Testing: Concepts and Operations by Ali Mili and Fairouz Tchier (Section 6.1)

VOLUME
STORAGE
WHITE-BOX
BLACK-BOX
SAFETY
COMPATIBILITY
INSTALLATION
PERFORMANCE
DOCUMENTATION
USABILITY
RECOVERY
SERVICEABILITY
INTEGRATION
INTEROPERABILITY
SECURITY
RELIABILITY
ACCEPTANCE
REGRESSION
UNIT
PROCEDURE
FUNCTION
FACILITY
SYSTEM
STRESS

Testing Taxonomy

- Black-box Testing
- White-box Testing

What is the basis of Testing?

Testing Taxonomy

- Black-box Testing
  - Tests are based on specification

- White-box Testing
  - Tests are based on implementation

Testing Taxonomy

- Black-box Testing
  - Tests are based on specification
  - Independent of implementation
  - Cannot deal with all exhibited (observable) behaviors

- White-box Testing
  - Tests are based on implementation
  - Independent of specification
  - Cannot deal with unsupported behaviors
  - Leads to brittle tests

Testing Taxonomy

- Unit Testing
- Integration Testing
- System Testing
- Acceptance Testing

What is the granularity of Testing?

Testing Taxonomy

- Unit Testing (granularity)
  - Test a unit (e.g., function, class) of the system
    - A sorting function
- Integration Testing (granularity)
  - Test the modules can interact as intended
    - Mars Climate Rover mission in 1999: English units vs Metric units
- System Testing (granularity)
  - Test the system as a whole to find as many faults before releasing it to the user (development team-centric)
- Acceptance Testing (granularity)
  - Test the system as a whole to ensure it satisfies the requirements (user-centric)

- Software Testing: Concepts and Operations by Ali Mili and Fairouz Tchier (Section 3.3)

Testing Taxonomy

- Reliability Testing
- Safety Testing
- Security Testing
- Performance Testing
- Stress Testing
- Interoperability Testing
- Compatibility Testing
- Installation Testing
- Serviceability Testing
- Usability Testing
- ….

What aspect is being tested?

- Software Testing: Concepts and Operations by Ali Mili and Fairouz Tchier (Chapter 7)
- The Art of Software Testing by Glenford J Myers

Realizing Testing

What do we need to realize any kind of testing?

Realizing Testing

What do we need to realize any kind of testing?

- Expected outcome
- Ability to observe concerned actual outcome
- Ability to compare outcomes for deviations
    - *Compare expected outcome and observed outcome for deviations*

What else can make it better?

Realizing Testing

What do we need to realize any kind of testing?

- Expected outcome
- Ability to observe concerned actual outcome
- Ability to compare outcomes for deviations
  - *Compare expected outcome and observed outcome for deviations*

What else can make it better?

- Automatic test execution
- Automatic test data generation
- Automatic test case generation