SYSC 4101 / SYSC5105

Context

Validation?
Verification?
Testing?

Why do we test?

What should we do during testing?

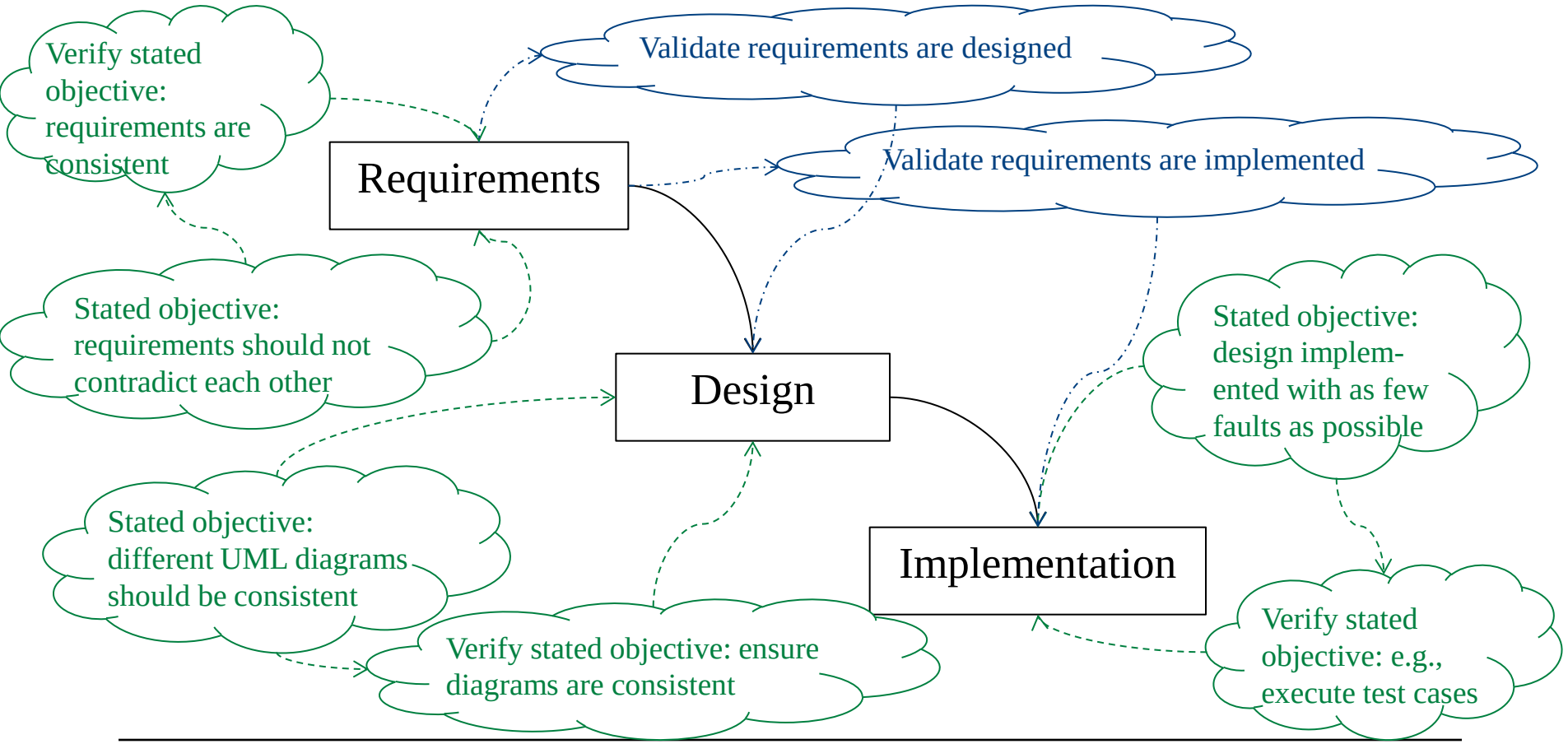
Definitions *(Verification vs. Validation)*

- ***Software Verification****: IEEE definition (Std 610.12.1990)*
  - ***The process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase.***
  - Checking whether the system adheres to properties termed as *verification properties*
  - Constructing the system well—do not use this definition

- ***Software Validation****: IEEE definition (Std 610.12.1990)*
  - ***The process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements.***
  - Relationship with other software engineering activities (e.g., Requirements elicitation, Analysis)
  - Constructing the right system—do not use this definition

Definitions (Verification vs. Validation)

Consider the waterfall software development model and its three first phases.



Verification vs Validation

Aspect	Verification	Validation
Purpose	Ensure the product is built correctly	Ensure the right product is built
Focus	Internal consistency, correctness, compliance	Meeting user needs and intended use
Testing Type	Static Testing	Dynamic Testing
Methods	Reviews, inspections, walkthroughs, static analysis	Unit testing, integration testing, system testing, user acceptance testing (UAT)
Code Execution	Does not involve executing the code	Involves running the code
Goal	Ensure the system matches design specifications	Ensure the system performs as expected by users
Examples	Code reviews, requirement reviews, design analysis	Code reviews, requirement reviews, design analysis

Definitions (V&V Techniques)

- Dynamic Techniques: i.e., through the execution of the system
  - Verification Testing (or simply, Testing): Inputs supplied to the system are valued (values instead of symbols)
    - ➢ The most used V&V technique
  - Symbolic execution
- Static Techniques: i.e., no program execution
  - Program slicing
  - Model Checking
  - Abstract interpretation
  - Inspection

Focus of SYSC4101/SYSC5105

These are non-exhaustive lists.

Symbolic execution

- Inputs supplied to the system are symbolic (symbols, not values)
- Visualizing what is accomplished by a sequence of (assignment) statements
- Compute an input/output function: any value of variable x in a procedure is a function f(a,b,c) of procedure parameters a, b and c. Symbolic execution computes f().

A simple program

```
1.  read(x,y);
2.  z := x+y;
3.  x := x-y;
4.  z := x*z;
5.  write(z);
```

Execute it with values x=2 and y=4.

z = -12

Execute it with symbols x=α and y=β.

z = α*α-β*β

Symbolic Execution (cont.)

```
1. void foo(int x, int y) {
2.   int t;
3.   if(x>y)
4.     t=x+1;
5.   else
6.     t=y;
7.   if(t<=x)
8.     // do something
9. }
```

- Replace `x`, `y`, `t` with symbols α, β, γ, respectively
- After line 6:
  - $\alpha>\beta \Rightarrow \gamma=\alpha+1$
  - $\alpha\leq\beta \Rightarrow \gamma=\beta$
- Line 8 executes (path condition) if
  - ① $\alpha>\beta \Rightarrow \gamma=\alpha+1 \wedge \gamma\leq\alpha$

  or
  - ② $\alpha\leq\beta \Rightarrow \gamma=\beta \wedge \gamma\leq\alpha$
- Are expressions satisfiable?
  - ① $\alpha+1\leq\alpha$ is not satisfiable
  - ② $\alpha\leq\beta \wedge \beta\leq\alpha$ is satisfiable: $\alpha=\beta$
- Code can “do something” only when x=y

Tools exist that compute path conditions and solve them!

e.g., Java Path Finder (NASA)

Program slicing

- Given a variable and its location in the control flow of the program, build an executable subprogram from the original program by identifying and discarding the statements irrelevant to the computation of the value to be assumed by that variable at that point.
- Lot's of tool support! Typical compiler task.

```
1.  begin
2.    read(x,y);
3.    total := 0.0;
4.    sum := 0.0;
5.    if x<=1
6.      then sum := y;
7.      else begin
8.        read(z);
9.        total := x*y;
10.     end;
11.   write(total,sum);
12. end;
```

Slice on the value of z at statement 12.

```
1.  begin
2.    read(x,y);
5.    if x<=1
6.      then
7.      else begin
8.        read(z);
10.     end;
11. end;
```

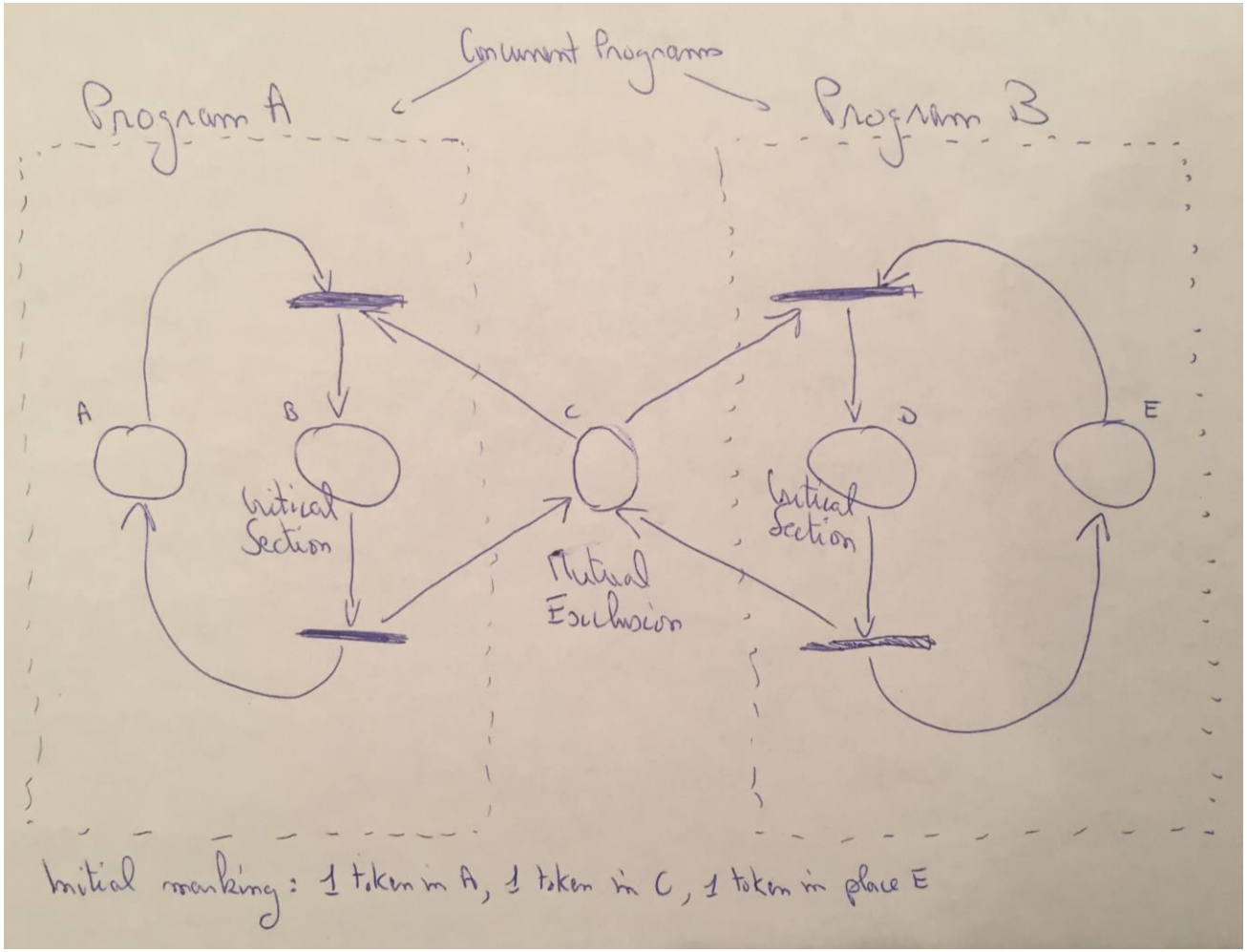
Slice on the value of x at statement 9.

```
1.  begin
2.    read(x,y);
12. end;
```

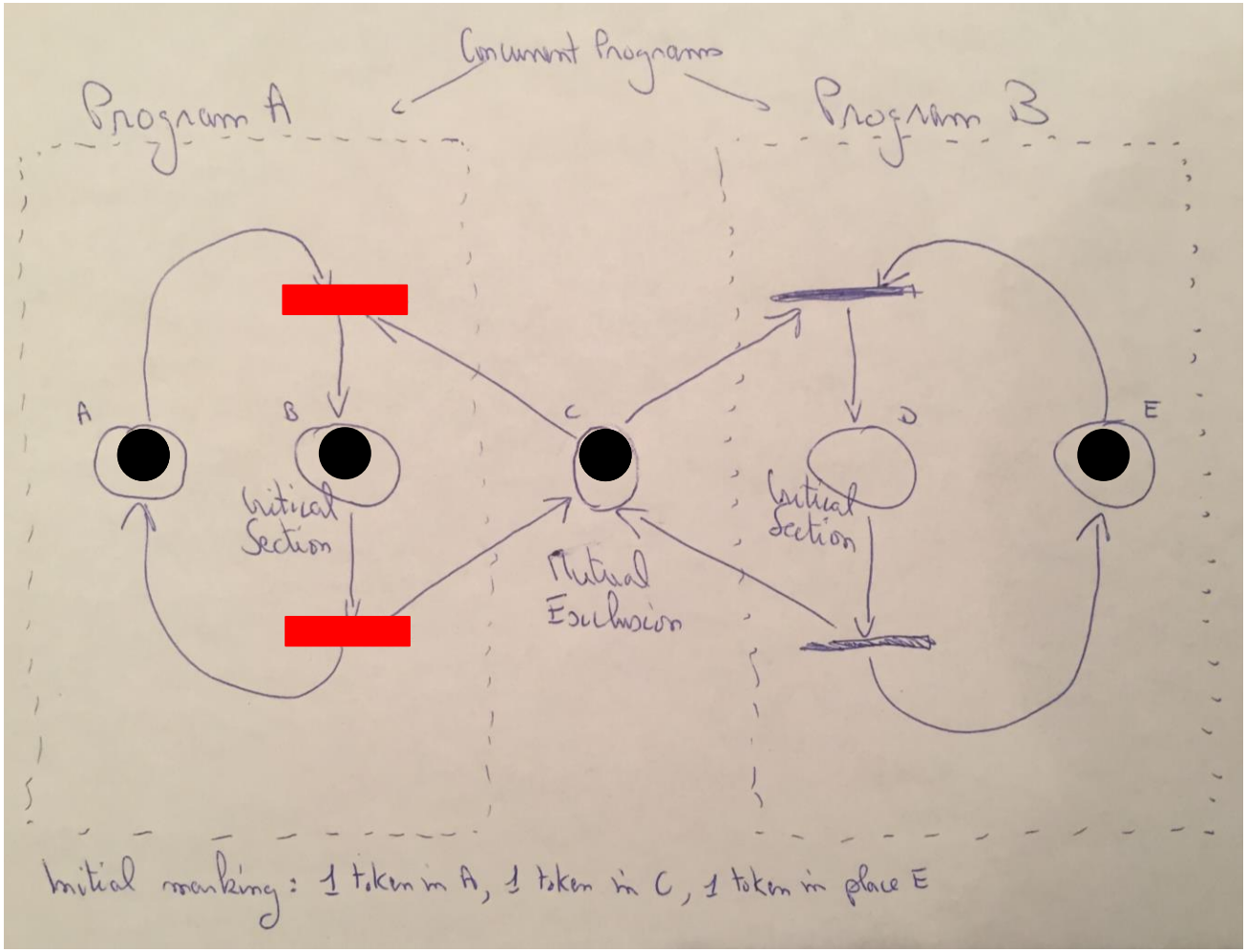
Model checking

- Verifying properties of the system using models (e.g., finite state machines, petri nets)
- Exhaustively and automatically check whether a model meets a given specification
- Examples:
  - Verify that a state is always reachable (liveness property) or, on the contrary, never reachable.
  - Verify there is no deadlock
  - Verify a program necessarily terminates

Aside note – Petri Nets

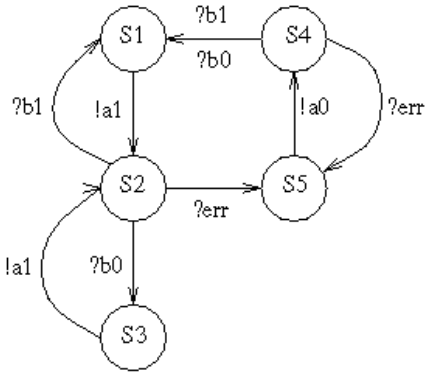


Aside note – Petri Nets

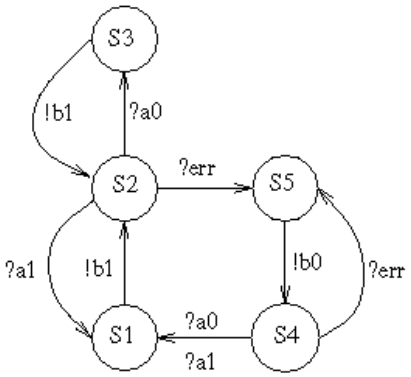


Model checking (cont.)

- The *alternating-bit protocol*
  - Originally designed to make it possible to transmit information reliably over noisy telephone lines with low-speed (e.g., 300 baud) modems.
  - Model from 1969 paper
    - Bartlett, K.A., Scantlebury, R.A., and Wilkinson, P.T. `A note on reliable full-duplex transmission over half-duplex lines,' Comm. of the ACM, 1969, Vol. 12, No. 5, 260-265



Terminal A



Terminal B

Can you tell whether there is an unreachable state?

A model checker can!
e.g., SPIN with the Promela language (Bell Labs)

Abstract Interpretation

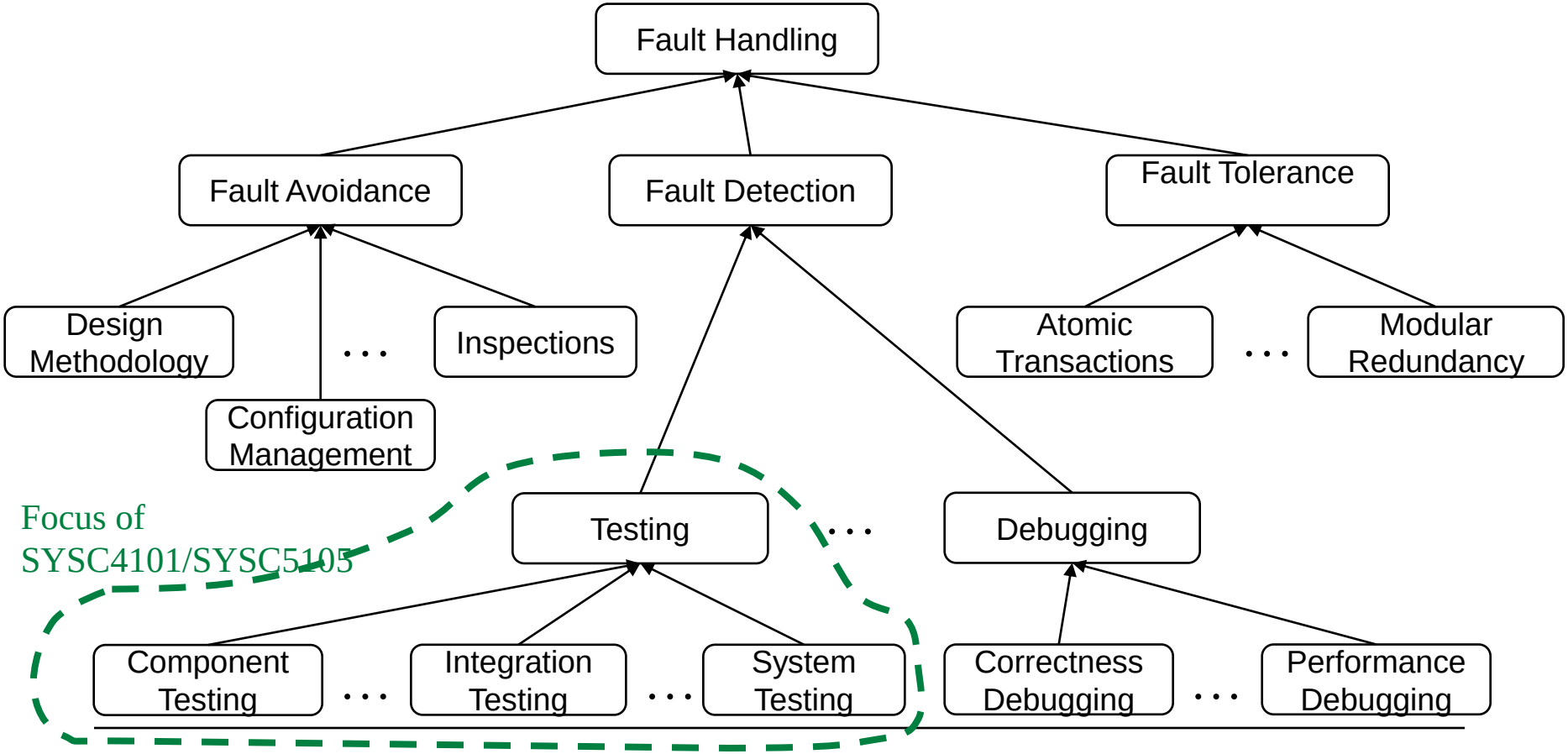
- With abstract interpretation, we make a sound approximation of the semantics of a program
  - Identify that all execution traces of the program are within bounds
    - All of them -> sound
    - Within bounds -> approximation
- If a property is true on the approximation then it is true for the program (all its executions)
  - Which properties?
    - Out of bounds counters (e.g., in loops), Division by zero, …
- Tool support scales up: 1,000,000 lines of C (Airbus) !
  - C++, Java, Ada support too.

Inspection

- Techniques aimed at *systematically* verifying software artifacts with the intent of finding as many defects as possible, as early as possible
- Performed by a group of team workers
- Faults often become more obvious to team members others than original author
- Works on source code but not only: any software artifact (e.g., requirements, diagrams, tests).

So … we are interested in Faults

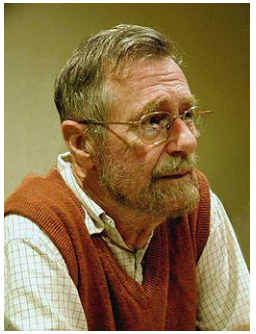
- We can be interested in faults in very different ways…



Goal of Testing

- Goal of Testing…

  "Program testing can be used to show the presence of bugs, but never to show their absence"
  [E.W. Dijkstra, Turing Award Lecture, 1972]



- No absolute certainty can be gained from testing
  - Because testing is necessarily finite.
- Testing should be integrated with other verification activities
  - See previous discussion on static and dynamic V&V techniques

- Main goal: demonstrate the software can be depended upon, i.e., *sufficient dependability*
  - *What is considered "sufficient" is context dependent !*
    - *E.g., phone app vs. aircraft auto pilot*

Remarks

- No matter how rigorous we are, software is going to be faulty
- Testing represents a substantial percentage of software development costs and time to market
- Impossible to test under all operating conditions – based on incomplete testing, we must gain confidence that the system has the desired behavior
- Testing large systems is complex – it requires strategy and technology- and is often done in practice

Cost of Testing

- You are going to spend about half of your development budget on testing, whether you want to or not.
- In real world usage, testing is the main post design activity
- Restricting early testing usually increases costs
- In some organizations there are more lines of test code than application code !!!
  - Rule of thumb: one line of application code => two lines of test code
  - Industry ratio Test-LOC / App-LOC can be 2, 3, 4
  - Open source (reverse) ratio App-LOC / Test-LOC can be 2, 3, … 7

Software Bugs - Cost

- "Impact of Inadequate Software Testing on US Economy"
  - National Institute of Standards and Technology (NIST), a US federal agency.
  - Studies in the manufacturing and transportation equipment sectors, to assess the cost to the U.S. economy of inadequate software testing infrastructure.
  - Results (annual cost):
    - Estimation: **$5.85 billion**
    - Projection to the entire U.S. economy: **$59.5 billion**
- Anecdotal evidence
  - Bug in telecom: 8-digit dollar cost in compensation
  - Debugging Fault: $100,000 for one!
- Consortium for Information and Software Quality
  - Poor software quality in the USA = $2 trillions in 2020

Software Bugs …

- Bug related to the year

  A 104 years old woman received an invitation to a kindergarten (1992).

- Interface misuse

  Underground train in London left a station without the driver (1990).

- Over budget project

  Failure in an automated luggage system in an airport (1995).

- NASA mission to Mars:

  Incorrect conversion from imperial to metric leads to loss of Mars satellite (1999)

- Ariane 5 Flight 501

  The space rocket was destroyed (1996).

- Therac-25

  Radiation therapy and X-ray machine killed several patients (1985-1987).