

# COMP1531

## 3.4 - SDLC Testing - Verification & Validation

# Verification

Verification in a system life cycle context is a set of activities that compares a product of the system life cycle against the required characteristics for that product. This may include, but is not limited to, specified requirements, design description and the system itself.

# Validation

Validation in a system life cycle context is a set of activities ensuring and gaining confidence that a system is able to accomplish its intended use, goals and objectives.

\*ISO/IEC/IEEE 29148:2018

# Verification

System has been built right

# Validation

The right system has been  
built

\*by intuition

# Formal Verification

- Proving (via Mathematics) that a piece of software has certain desirable properties
- Treats the software, or the algorithms implemented in the software, as a mathematical object that can be reasoned about.
- Typically involves tools like proof assistants, model checkers or automatic theorem provers.
- **Not something we cover in this course**

# Formal Verification

- Tends to have a high cost in terms of effort
- E.g. to **verify a microkernel**
  - it took ~20 person years
  - and ~480,000 lines of proof script
  - for ~10,000 of C

What is testing  
anyway?

“Testing shows the  
presence, not the  
absence of  
bugs” – *Edsger W.  
Dijkstra*

# Unit testing

ISTQB definition:

The testing of individual software components

Method:

White-box

Who:

Software Engineers



# Integration Testing

ISTQB definition:

Testing performed to expose defects in the interfaces and in the interactions between integrated components or systems.

Method:

White-box or Black-box

Who:

Software Engineers or independent testers

# System Testing

ISTQB definition:

The process of testing an integrated system to verify that it meets specified requirements.

Method:

Black-box

Who:

Normally, independent testers

# Acceptance Testing

ISTQB definition:

Formal testing with respect to user needs, requirements, and business processes conducted to determine whether or not a system satisfies the acceptance criteria and to enable the user, customers or other authorized entity to determine whether or not to accept the system.

Method:

Black-box

Who:

User or Customer

How do we know if  
our tests are good?

# Coverage

- Test Coverage: a measure of how much of the feature set is covered with tests
- Code coverage: a measure of how much code is executed during testing

# Example: Leap years

```
1 def is_leap_year(year):  
2     if year % 4 != 0:  
3         return False  
4     elif year % 100 != 0:  
5         return True  
6     elif year % 400 != 0:  
7         return False  
8     else:  
9         return True
```

# Coverage.py

- Measure code coverage as a percentage of statements (lines) executed
- Can give us a good indication how much of our code is executed by the tests
- ... and most importantly highlight what has **not** been executed.

# Example: Year from day

```
1 def day_to_year(days):
2     '''
3     Given a number of days from January 1st 1970, return the year.
4     '''
5     year = 1970
6
7     while days > 365:
8         if is_leap_year(year):
9             if days > 366:
10                 days -= 366
11                 year += 1
12            else:
13                days -= 365
14                year += 1
15
16     return year
```



# Checking code coverage

- Run Coverage.py for your pytest:

```
python3-coverage run --source=. -m pytest
```

- View the coverage report:

```
python3-coverage report
```

- Generate HTML to see a breakdown (puts report in htmlcov/)

```
python3-coverage html
```

# Case study: Zune Bug

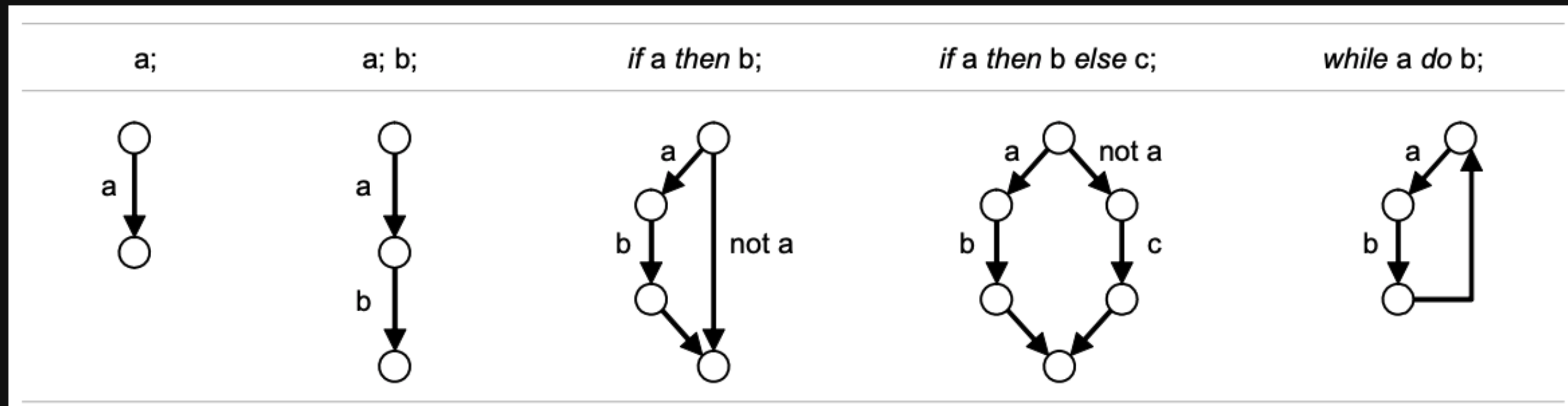
- On December 31st 2008, Microsoft Zunes stopped working for the whole day.
- The bug in the above code caused them to go into an infinite loop
- Hardly catastrophic, but embarrassing for Microsoft



Can we find the Zune bug  
with testing?

# Branch coverage checking

- For lines that can potentially jump to more than one other line (e.g. if statements), check how many of the possible branches were taken during execution
- Can be done with the --branch option in Coverage.py
- Sometimes referred to as edge coverage



Does code coverage imply  
test coverage?

What is the right  
level of code  
coverage?

# Summary

- Code coverage is useful
- It's more important to look at what's not covered than the coverage percentage
- Branch coverage is a more accurate measurement so you should use it instead of statement coverage
- Like all measurements, it's important to understand what meaning to attach to it