

# Testing techniques

---

Tools and Techniques for Software Testing - Barbara Russo  
SwSE - Software and Systems Engineering group

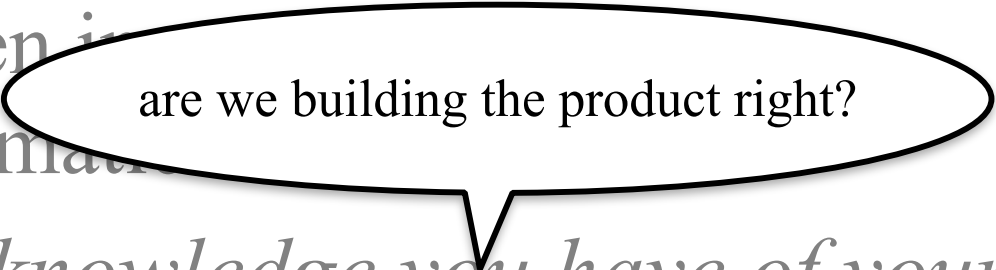
---

# Testing

- **Software testing** is a verification process that **detects** differences between **existing** and **required** conditions and to **evaluate** the features of the software item

# What is testing

- Understanding your **system's behaviour** under different types of **stimuli (input)**
- In some cases the output (behaviour) is clear/certain in other it might not
  - For example you might not know what will be the output for a given input if the system is not deterministic mathematical
- *Testing increases the knowledge you have of your system (in terms of input and behaviour)!*



are we building the product right?

# Why testing is hard

- We want
  - to know when software is good enough to release
  - to deliver software with few known bugs
- but
  - it is very hard to ensure quality in software
  - residual defect rate after shipping can be high

# Residual defect rate

- 1 - 10 defects/kloc (typical)
- 0.1 - 1 defects/kloc (high quality: Java libraries)
- 0.01 - 0.1 defects/kloc (very best: NASA)
- You may think not so bad, but

*1Mloc with 1 defect/kloc means  
1000 bugs are missed!*

# Exhaustive testing is infeasible

- Input space is generally too big to be covered exhaustively
- Imagine exhaustively testing a 32-bit multiply operation,  $a*b$ : there are  $2^{64}$  test cases!

# Statistical testing doesn't work for software

- Other engineering disciplines can test small random samples (e.g. 1% of hard drives manufactured) and infer defect rate for whole lot
- Many tricks to speed up time (e.g. opening a refrigerator 1000 times in 24 hours instead of 10 years)

# Failure rates are not uniform for software

- Overflow bugs (like Ariane 5) happen abruptly
- Pentium FDVI division bug affected approximately 1 in 9 billion divisions (Byte magazine)
- Ariane 5 [https://www.youtube.com/watch?v=gp\\_D8r-2hwk](https://www.youtube.com/watch?v=gp_D8r-2hwk)
- Ian Sommeriville <https://www.youtube.com/watch?v=W3YJeoYgozw>

*The failure of the Ariane 501 was caused by the complete loss of guidance and altitude information 37 seconds after start of the main engine ignition sequence (30 seconds after lift-off). This loss of information was due to specification and design errors in the software of the **inertial reference system**.*

*The internal SRI\* software exception was caused during execution of a data conversion from 64-bit floating point to 16-bit signed integer value. The floating point number which was converted had a value greater than what could be represented by a 16-bit signed integer.*



# Questions

- What was the failure for Ariane 5?
- What was the bug in Ariane 5 software?
- Which characteristic had the backup software?
- Which facility was missing in Ariane 5?
- What was the engineering principle used for the development of Ariane 5?
- What was the reason not to test Ariane 5?
- What was the problem related to development process?

# How to test

- Identify test specifications
- Define the testing process; for example:
  - Acceptance tests
  - Fitness tests
  - Unit tests
  - Regression tests and so on
- Design test cases

# Testing - example: An auction system

- Software Specification:
  - show the list of ongoing auctions by vocal command
- Testing:
  - Define testing specification:
    - at the vocal command “Show auctions’ list” a list of ongoing auctions is displayed on the screen
    - at the vocal command “Show” a question “what?” replayed

# Testing - example: An auction system

- Choose a testing stage
  - Unit testing
- Choose a testing harness
  - JUnit
- Choose a pass/fail criterion
  - assertion + oracle
- Execute the test

# Test Case

- **A test case is a choice of Inputs, Execution Conditions and Pass / Fail Criterion**

# Example - test case

- *Test specification:* at the vocal command “Show” a question “what?” replayed
- *I:* “Show”
- *EC:* unde vocal command
- *P/F C:*
  - when “Show -> What?” then PASS
  - when “Show -> a list of auctions is displayed?” then FAIL

# Straightforward testing does not work

- Input space may be big and complex!
  - **Arbitrary testing** is not convincing: “just try it and see if it works”
  - **Exhaustive testing** is clearly infeasible — even a simple `int -> int` function requires billions of runs to test all inputs
  - **Random testing** is less likely to discover bugs

# One convenient solution

---

Tools and Techniques for Software Testing - Barbara Russo  
SwSE - Software and Systems Engineering group

---



# Partitioning the **input space**

- We want to pick a set of tests that is small enough to run quickly, yet large enough to validate the program
- To do this, divide the **input space** into **subdomains**, each consisting of a set of inputs
- The subdomains completely cover the input space, so that every input lies in at least one subdomain

# Test selection with partition

- Then we choose one input set from each subdomain
- Partition the input space into sets of similar inputs
- Then use one representative of each set

# Test selection with partition

- This approach makes the **best use of limited testing resources** by choosing dissimilar test cases, and forcing the testing to explore parts of the input space that random testing might not reach

# When it is worth to apply

- Input space is very large, but program is relatively small or simple

# Examples - multiply

- multiply : BigInteger x BigInteger -> BigInteger
- partition BigInteger into:
  - BigNeg, SmallNeg, -1, 0, 1, SmallPos, BigPos
- Pick a value from each class
  - -265, -9 -1, 0, 1, 9, 265
- Test the  $7 \times 7 = 49$  combinations

# Example - max

- $\text{max} : \text{int } x, \text{int} \rightarrow \text{int}$
- Partition into:
  - $a < b, a = b, a > b$
- Pick value from each class
  - $(1, 2), (1, 1), (2, 1)$

# Exercise [menti.com](https://www.menti.com)

- Test the multiplication

- $\text{max} : \text{int} \times \text{int} \rightarrow \text{int}$

- $\text{abs}() : \text{int } x \rightarrow |x|$

$$g(x,y)=\text{abs}(x)*\text{max}(y,x)$$

- How many combinations?

# Example - intersect

- intersect : Set x Set  $\rightarrow$  Set
- Partition Set into:
  - $\emptyset$ , singleton, many
- Partition whole input space into:
  - this = that, this  $\subseteq$  that, this  $\supseteq$  that, this  $\cap$  that  $\neq \emptyset$ , this  $\cap$  that =  $\emptyset$
- Pick values that cover both partitions
  - $\{\}, \{\}$   $\{\}, \{2\}$   $\{\}, \{2,3,4\}$
  - $\{5\}, \{\}$   $\{5\}, \{2\}$   $\{4\}, \{2,3,4\}$
  - $\{2,3\}, \{\}$   $\{2,3\}, \{2\}$   $\{1,2\}, \{2,3\}$



# Boundary testing

- Include classes at boundaries of the input space
  - zero, min/max values, empty set, empty string, null
- Why? because bugs often occur at boundaries:
  - off-by-one bugs (for loops)
  - forget to handle empty containers
  - overflow errors in arithmetic

# Types of testing - depend on the access to information

---

Tools and Techniques for Software Testing - Barbara Russo  
SwSE - Software and Systems Engineering group

---

# Types of testing

- White box testing and black box testing

internal values

associations' values

public values

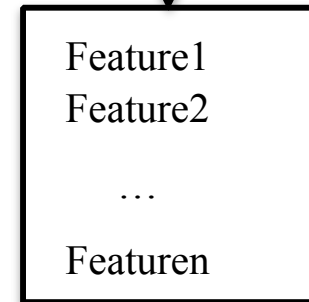
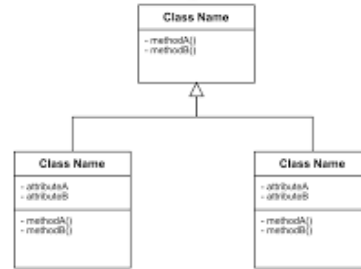
**Input**

**Input**

**Input**

```

13
14 float foo (int a, int b, int c, int d, float e) {
15     if (a == 0) {
16         return 0;
17     }
18     int x = 0;
19     if ((a==b) || ((c == d) && bug(a) )) {
20         x=1;
21     }
22     e = 1/x;
23     return e;
24 }
25
26
27 public Boolean bug(int a) {
28     if(a==1){
29         boolean dummy = false;
30         return dummy;
31     }
32     else {return false;}
33 }
34
    
```



**Expected Behaviour (If Any) Expected Behaviour (If Any) Expected Behaviour (If Any)**

# Type of testing

- The type of testing (**White or Black**) that you can perform depends on the accessibility to the information about your system or software
- The more you know about the internal structure of software/system the greater amount of techniques you can apply

# Black-box (functional) testing

- **Goal:** Understanding the behaviour *without being biased by the specific implementation*
  - Testers do not see how software has been implemented
- Exercise different **public features or services** of software/system under different *environmental settings* (e.g., portability for OS) or *hardware/software configurations*

# Understanding the behaviour

- The solutions of an equation of second order

$$0 = ax^2 + bx + c$$

- are two  $x_{1,2}$ ; the function is

$$(a, b, c) \Rightarrow x_{1,2}$$

- How many combinations of  $a, b, c$  to check the number and type of solutions? [menti.com](https://www.menti.com)

# Solution

Is it always true?  $a=0$

$a \neq 0 \Rightarrow a=1$

How many outputs? Let's see the behaviours:  $b^2-4ac > 0$ ,

$b^2-4ac=0$ ,  $b^2-4ac < 0 \Rightarrow (1,1,0)$ ,  $\{(1,0,0), (1,4,1)\}$ ,  $(1,1,1)$

# White-box testing

- **Goal:** Understanding the behaviour *of a specific implementation*
- Exercise the internal structure



# With WBT

- *Exercise independent execution paths* within a module or unit
- *Exercise logical decisions* on both true and false
- *Execute loops at their boundaries* and
- *Exercise internal data structures* to ensure their validity

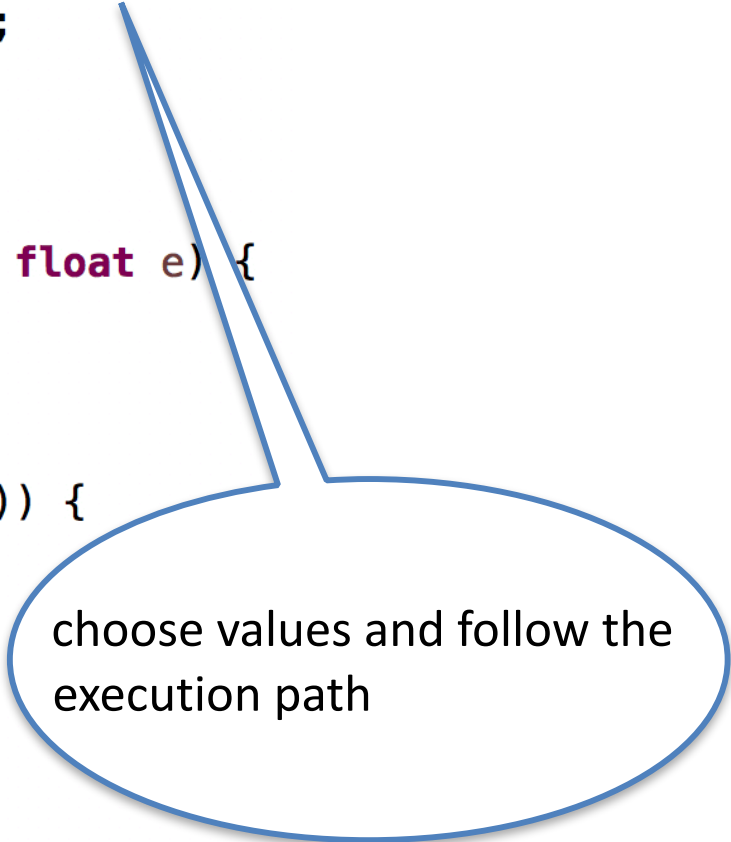
# Execution paths

```
13
14⊖ float foo (int a, int b, int c, int d, float e) {
15     if (a == 0) {
16         return 0;
17     }
18     int x = 0;
19     if ((a==b) || ((c == d) && bug(a) )) {
20         x=1;
21     }
22     e = 1/x;
23     return e;
24 }
25
26
27⊖ public Boolean bug(int a) {
28     if(a==1){
29         boolean dummy = false;
30         return dummy;
31     }
32     else {return false;}
33 }
34
```

```

6 public class FooClass {
7
8 public static void main(String[] args) {
9     FooClass fooClass = new FooClass();
10    fooClass.foo(1,2,3,4,5);
11
12 }
13
14 float foo (int a, int b, int c, int d, float e) {
15     if (a == 0) {
16         return 0;
17     }
18     int x = 0;
19     if ((a==b) || ((c == d) && bug(a) )) {
20         x=1;
21     }
22     e = 1/x;
23     return e;
24 }
25
26 public Boolean bug(int a) {
27     if(a==1){
28         return true;
29     }
30     else {return false;}
31 }
32
33 }

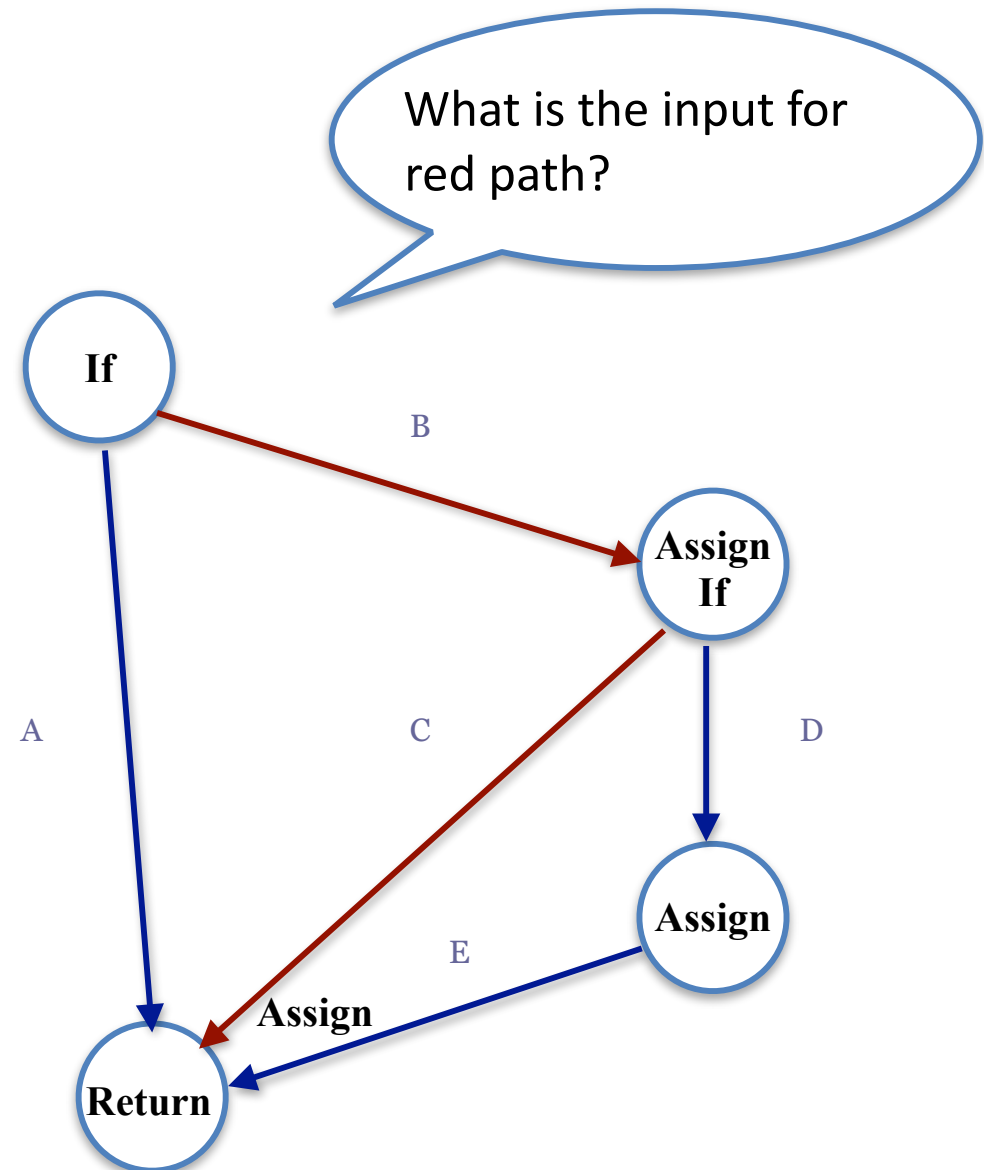
```



choose values and follow the execution path

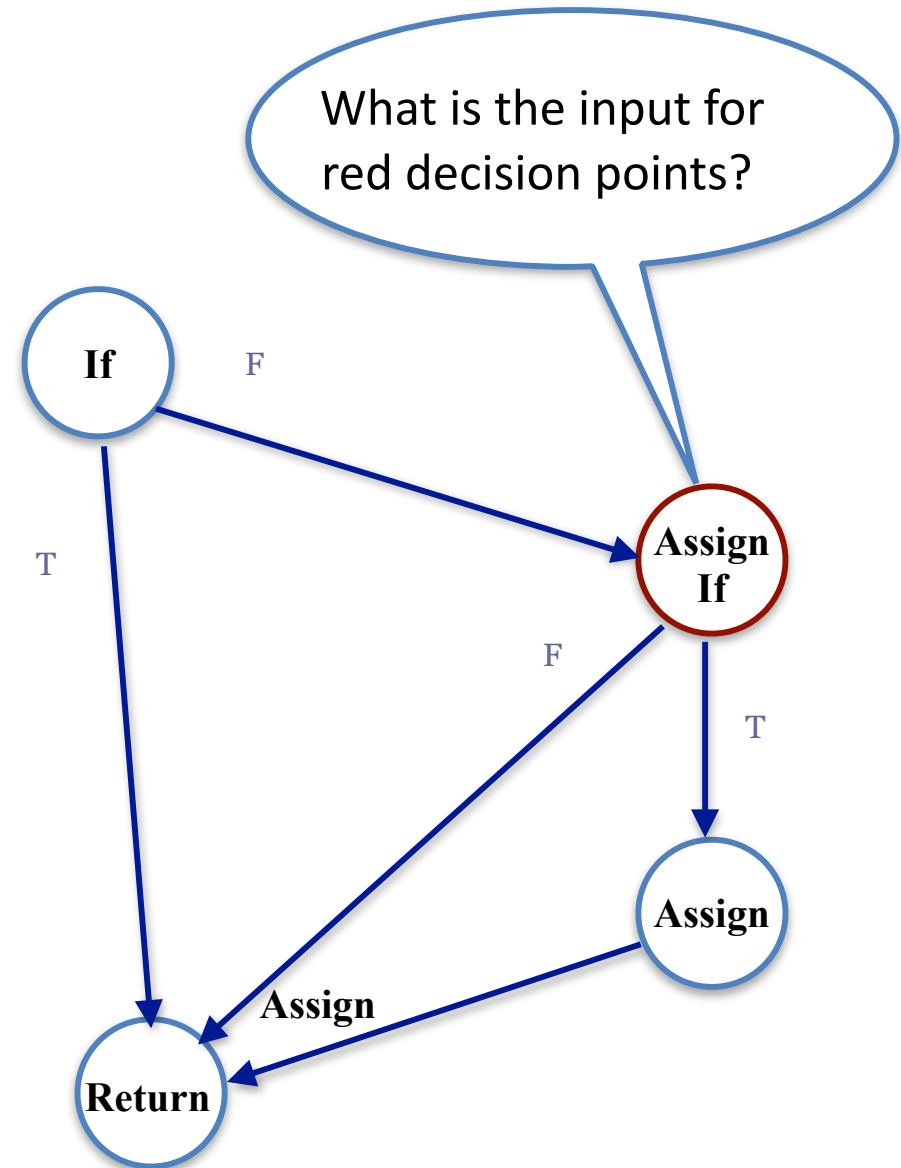
# Exercise independent execution paths

```
13
14⓪ float foo (int a, int b, int c, int d, float e) {
15     if (a == 0) {
16         return 0;
17     }
18     int x = 0;
19     if ((a==b) || ((c == d) && bug(a) )) {
20         x=1;
21     }
22     e = 1/x;
23     return e;
24 }
25
26
27⓪ public Boolean bug(int a) {
28     if(a==1){
29         boolean dummy = false;
30         return dummy;
31     }
32     else {return false;}
33 }
34
```



# Exercise logical decisions on both true and false

```
13
14 | float foo (int a, int b, int c, int d, float e) {
15 |     if (a == 0) {
16 |         return 0;
17 |     }
18 |     int x = 0;
19 |     if ((a==b) || ((c == d) && bug(a) )) {
20 |         x=1;
21 |     }
22 |     e = 1/x;
23 |     return e;
24 | }
25
26
27 | public Boolean bug(int a) {
28 |     if(a==1){
29 |         boolean dummy = false;
30 |         return dummy;
31 |     }
32 |     else {return false;}
33 | }
34
```



# Execute loops at their boundaries and within their operational bounds

```
3 public class LoopClassExample {
4     public static StringBuffer collapseSpaces(String argStr){
5         char last = argStr.charAt(0);
6         StringBuffer argBuf = new StringBuffer();
7         for(int cldx=0; cldx<argStr.length(); cldx++){
8             char ch = argStr.charAt(cldx);
9             if(ch!= ' ' || last!= ' '){
10                argBuf.append(ch);
11                last=ch;
12            }
13        }
14        return argBuf;
15    }
16 }
17
```

# Execu

```
3 public class LoopClassExample {
4     public static StringBuffer collapseSpaces(String argStr){
5         char last = argStr.charAt(0);
6         StringBuffer argBuf = new StringBuffer();
7         for(int cldx=0; cldx<argStr.length(); cldx++){
8             char ch = argStr.charAt(cldx);
9             if(ch!= ' ' || last!= ' '){
10                argBuf.append(ch);
11                last=ch;
12            }
13        }
14        return argBuf;
15    }
16 }
17 }
```

# daries

Input: argStr =Your

Output: argBuffer=Your

Input: argStr = ' 'You

Output: argBuffer=You

Input: argStr = I am

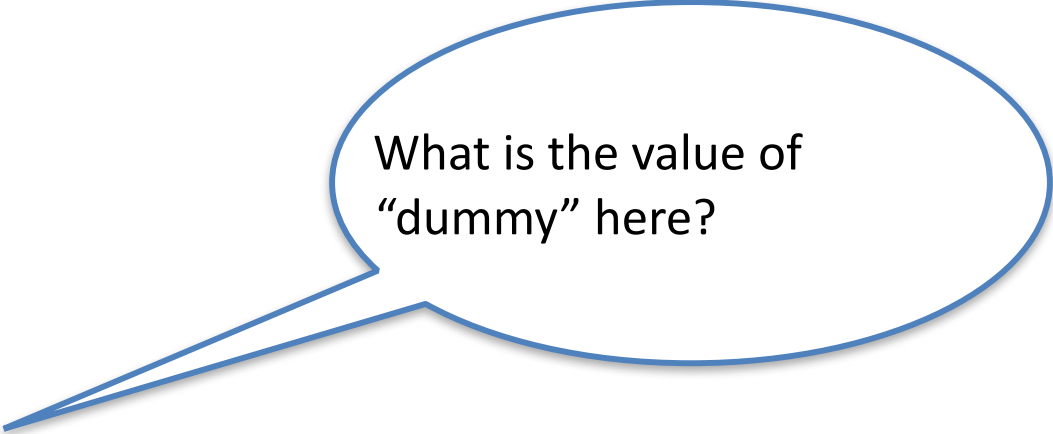
Output: argBuffer= I am

<i>last</i>	<i>ch</i>	<i>argBuf</i>	<i>cldx</i>
<i>Y</i>	<i>Y</i>	<i>Y</i>	
<i>Y</i>	<i>o</i>	<i>o</i>	
<i>o</i>	<i>u</i>	<i>u</i>	
<i>u</i>	<i>r</i>	<i>r</i>	
<i>r</i>			

case II and III: check boundaries and within operational bounds!

# Exercise internal data structures to ensure their validity

```
7⊖ /*
8  * A class that describes a simple company made of at least of one office.
9  */
10 public class DataStructureClass {
11     private Company myCompany;
12     private Office myOffice;
13
14⊖     public boolean isValid(Company company, Office office){
15         // No Company
16         if(company == null) {
17             return false;
18         }
19         // No Office
20         if(office == null) {
21             return false;
22         }
23         return true;
24     }
25
26     boolean dummy = isValid(myCompany, myOffice);
27
28 }
```



What is the value of  
“dummy” here?