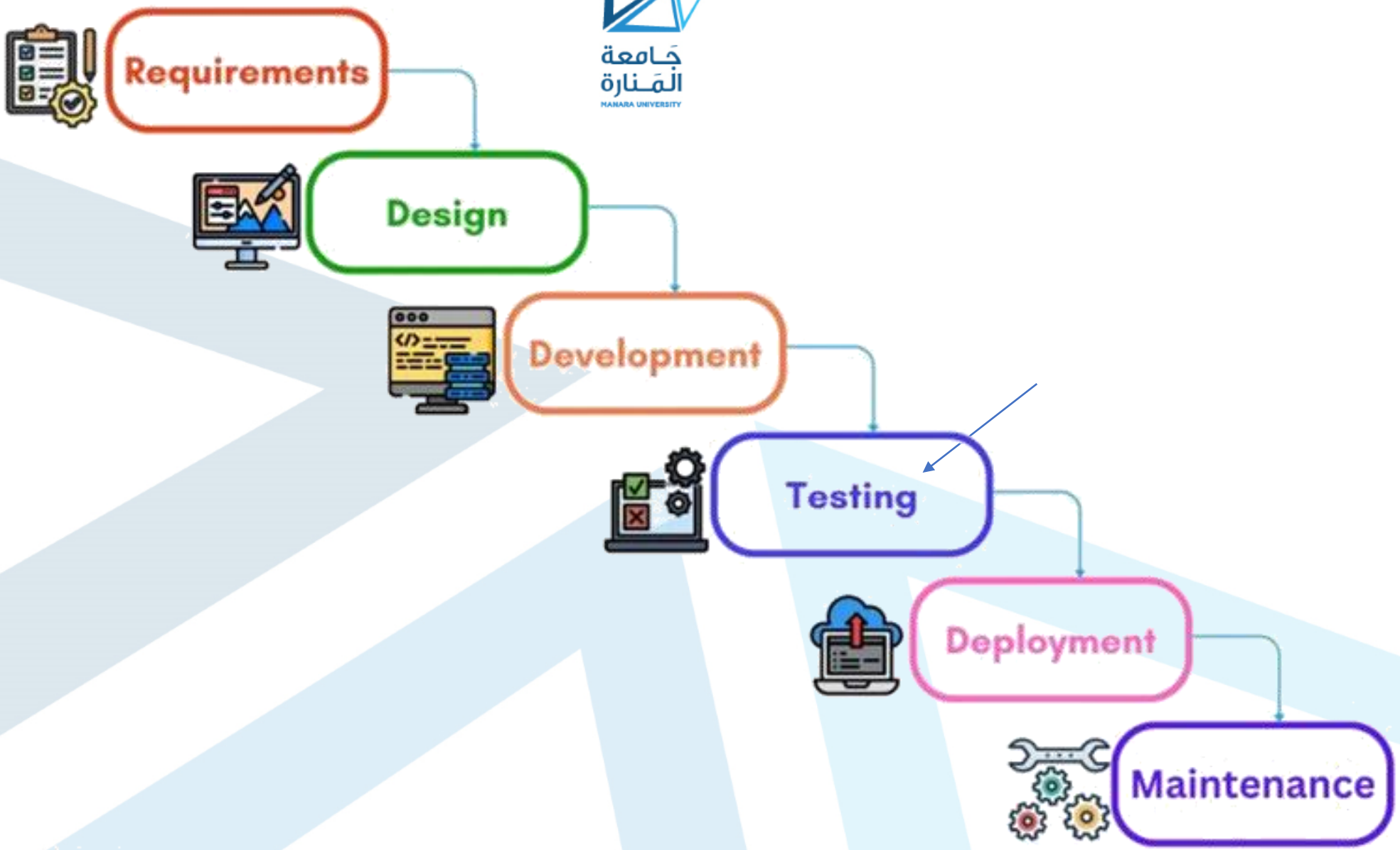
Software Engineering -2-

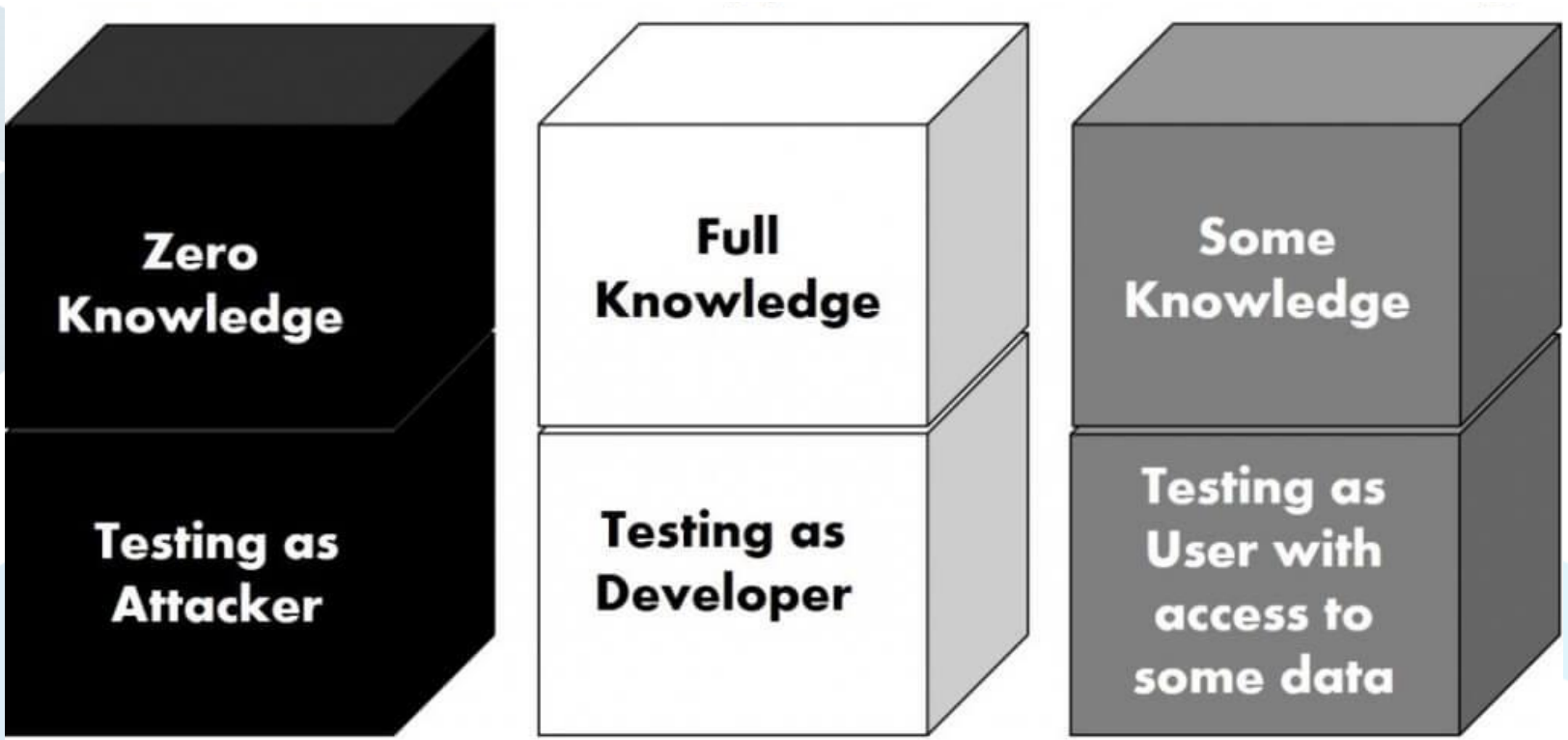
Lecture -7-

Dr. Inas Laila


Requirements
Design
Development
Testing
Deployment
Maintenance

Test Technique

- White Box
- Black Box
- Gray Box



الصندوق الأسود	الصندوق الأبيض	المعيار
غير مطلوبة	مطلوبة	المعرفة بالكود
الوظائف والمتطلبات	الهيكل الداخلي	التركيز
Equivalence Partitioning, Boundary Value, Decision Tables	Statement, Branch, Path, Condition, Loop	التقنيات

WHITE Box Testing (WBT)

اختبار الصندوق الأبيض هو نوع من **الاختبارات** التي تركز على **التحقق من الكود الداخلي للبرنامج**، وليس فقط النتائج النهائية. يعتمد اجراء هذا الاختبار على **معرفة الكود الداخلي** ويتطلب معرفة **الهيكل البرمجي** (الدوال، المتغيرات، الحلقات، الشروط والتفرعات).

الهدف من هذا الاختبار هو **اختبار كل جزء من بنية البرنامج** للتأكد من أن كل مسار يعمل بشكل صحيح. ويمكن تلخيص الأهداف بانها

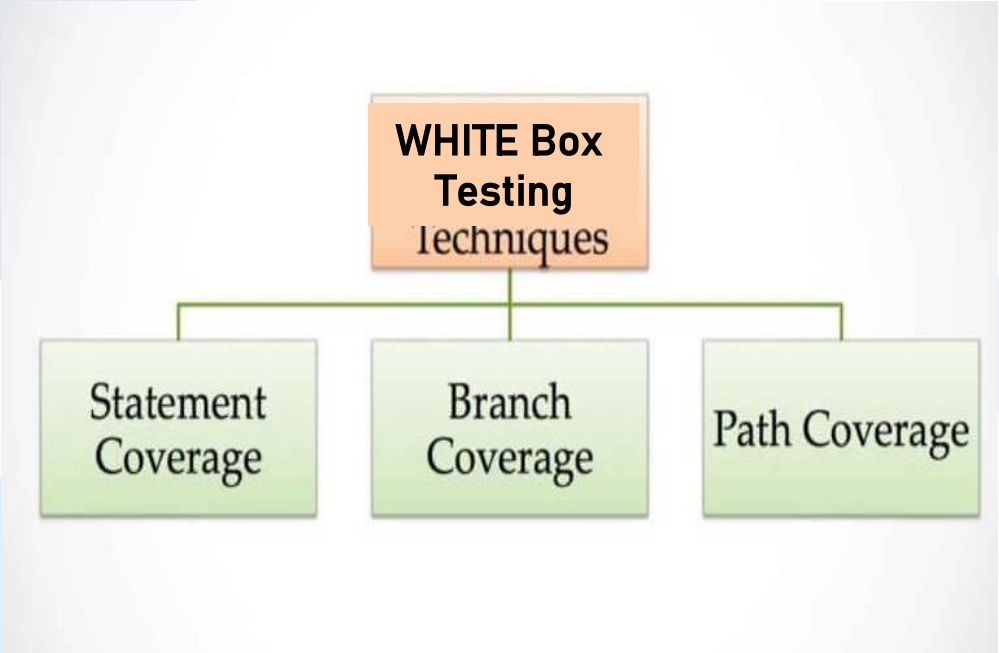
- التأكد من أن جميع المسارات في الكود تعمل كما يجب.
- كشف الأخطاء في المنطق، الحلقات، الشروط، والتفرعات.
- تغطية جميع التعليمات.

How do you perform White Box Testing?

STEP 1) UNDERSTAND THE SOURCE CODE

Step 2) CREATE TEST CASES AND EXECUTE

يوجد العديد من تقنيات اختبار الصندوق الأبيض وهي تستخدم لتصميم حالات الاختبار بحيث تحقق تغطية مقدارها ١٠٠ بالمئة



تقنيات اختبار الصندوق الأبيض

١- تقنية تغطية التعليمات (Statement Coverage)

الهدف: التأكد من تنفيذ **كل سطر من الكود مرة واحدة على الأقل.**

- مثال:

```
def check_positive(n):
    if n > 0:
        return "Positive"
    else:
        return "Non-positive"
```

للمرور بكل التعليمات، يجب استخدام حالتي اختبار:

n > 0

n <= 0

يمكننا حساب النسبة المئوية من التعليمات المغطاة (المنفذة) عند تطبيق حالة اختبار معينة وفق القانون

$$Statement\ Coverage = \frac{Number\ of\ executed\ \text{statements}}{Total\ number\ of\ statments} \times 100$$

بفرض لدينا الكود التالي:

```
1-  public static void main(){
2-  Scanner sc=new Scanner (System.in);
3-  int x=sc. nextInt() ;
4-  int y=sc. nextInt() ;
5- if(x>y)
6- System.out.print("x bigger than y");
7- else
8- System.out.print("y bigger than x");
9- }
```

أوجد Statement Coverage في كل من الحالات التالية:

Test Case 1

- Y=2 , X=3

ال Statement المنفذة هي

```
1-  public static void main(){
2-  Scanner sc=new Scanner (System.in);
3-  int x=sc. nextInt() ;
4-  int y=sc. nextInt() ;
5- if(x>y)
6- System.out.print("x bigger than y");
7- else
8- System.out.print("y bigger than x");
9- }
```

ومنه يكون

Number of executed statements = 7

Total number of statements =9

$$\text{Statement Coverage} = \frac{7}{9} \times 100 = 78\%$$

Test Case 2

- Y=5 , X=4

Number of executed statements = 8

Total number of statements =9

$$\text{Statement Coverage} = \frac{8}{9} \times 100 = 89\%$$

نلاحظ أننا بحاجة إلى اختبارين من أجل تغطية كافة Statements بنسبة 100%

٢- تقنية تغطية الفروع (Branch Coverage)

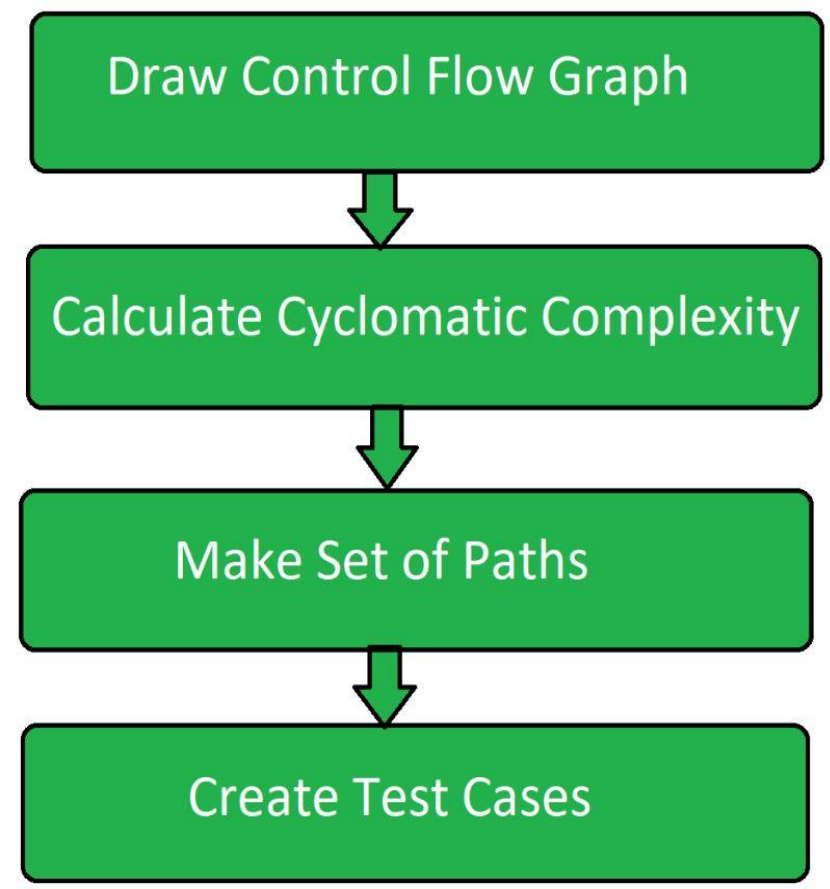
الهدف: التأكد من تنفيذ **كل فرع من الفروع الشرطية** على الأقل مرة واحدة.

```
if x > 0:
    print("Positive")
else:
    print("Non-positive")
```

كم TCs نحتاج لتغطية الفروع؟

٣- تقنية تغطية المسارات (Path Coverage)

- الهدف: التأكد من اختبار **كل مسار ممكن في البرنامج**.
- تشمل كل التفرعات والتكرارات.
- مثال: في دوال بها شروط متداخلة (if-else if-else) وحلقات.



Path Testing is a method that is used to **design the test cases**.
In path testing method, the control flow graph(CFG) of a program is designed to find a set of linearly independent paths of execution.
In this method Cyclomatic Complexity is used to determine the number of linearly independent paths and then test cases are generated for each path.

Advantages of Path Testing:

1.Path testing method reduces the redundant tests.
2.Path testing focuses on the logic of the programs.
3.Path testing is used in test case design.

ما هو Control Flow Graph (CFG)

مخطط يوضح جميع المسارات الممكنة في البرنامج من البداية للنهاية.

يتم فيه تمثيل كل تعليمة قابلة للتنفيذ أو شرط بعقدة Node ونصل بين العقد عن طريق الحواف Edges تمثل تدفق التنفيذ بين العقد.

السهم من n1 إلى n2 يعني إمكانية تنفيذ العقدة n2 بعد تنفيذ العقدة n1

يستخدم كثيرًا في اختبارات الصندوق الأبيض (White Box Testing)

```
def example(x):
    if x > 0:
        print("Positive")
    else:
        print("Non-positive")
    print("Done")
```

CFG??

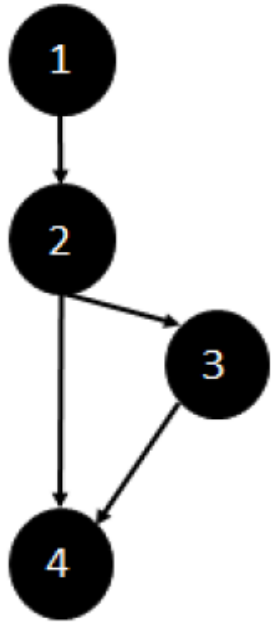
CFG Construction

Sequence of simple statement

1-Statement 1;
2-Statement 2;
3-Statement 3;
.....
..
...
n- Statement n;

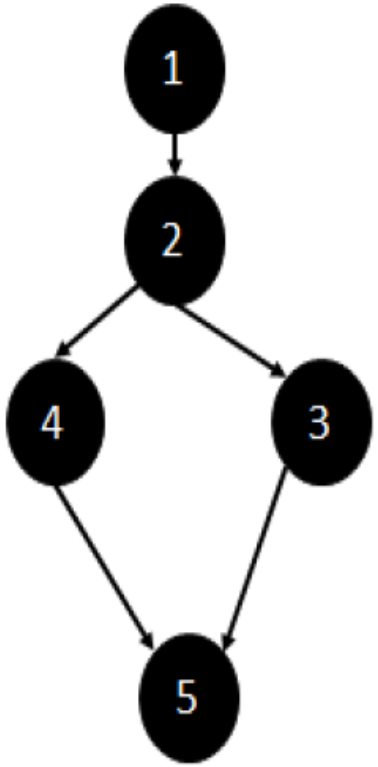
If-then statement

1.Statement
2.if(cond)
    3. Statement
4.Statement



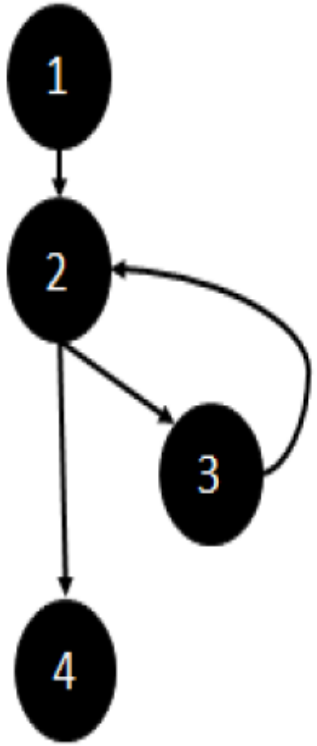
If-then-else statement

1.Statement
2.if(cond)
    3. Statement
4.Else Statement
5.Statement



While loop

1.Statement
2.while(cond)
    3. Statement
4. Statement



التعقيد الحلقي Cyclomatic Complexity

هو مقياس عددي يعكس عدد المسارات المستقلة **في برنامج ما** وهو يستخدم لتحديد **عدد حالات الاختبار المطلوبة لتغطية جميع المسارات المكنة** في البرنامج ويتم ذلك عن طريق يعطي عدد المسارات المستقلة في البيان.

يتم حسابه وفق التالي:

$$V(G) = e - n + 2P$$

E = the number of edges of the graph

N = the number of nodes of the graph

(p=1) حيث

مثال :كود max :

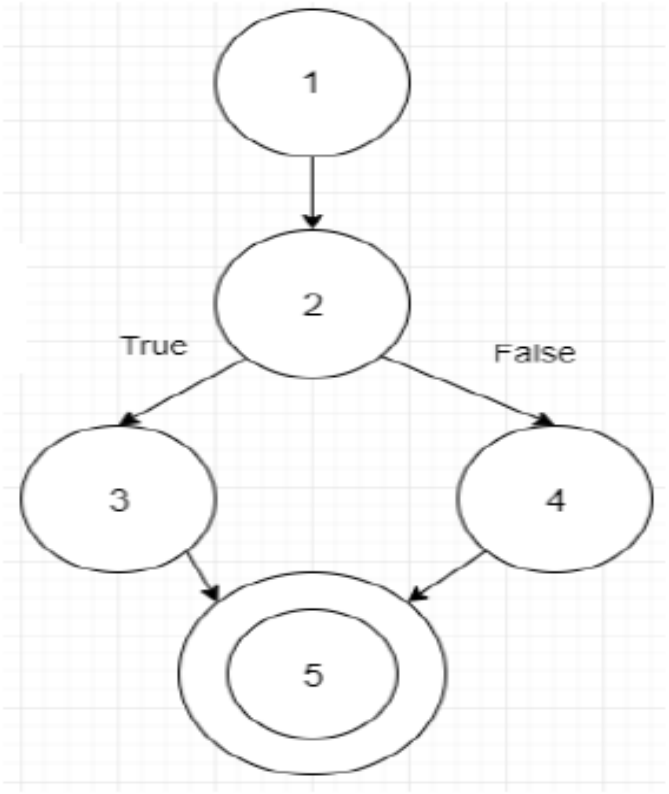
```
int max (int a,int b)
{
  1  int max
  2  if(a>b)
  3  max = a
  else
  4  max = b
  5 return max
}
```

1. رسم الCFG مع الأخذ بعين الاعتبار بداية واحدة ونهاية واحدة.

٢.احسب التعقيد الحلقي (عدد المسارات)

ماهي حالات الاختبار التي تقترحها هنا؟؟

CFG



2. تحديد المسارات ،

1,2,3,5

1,2,4,5

وعدد المسارات : هو عدد يمثل تعقيد الكود Cyclomatic Complexity.

مثال ٢ بفرض لدينا الكود التالي:

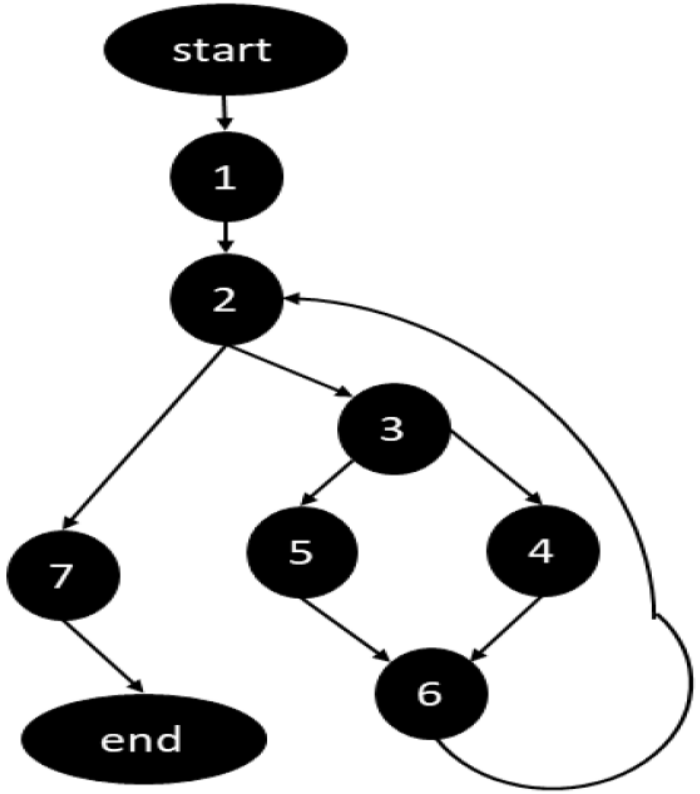
قم بتمثيله باستخدام مخططات Control Flow Graph) CFG)

أوجد التعقيد الحلقي Cyclomatic Complexity

أوجد المسارات المستقلة Independent Paths

ماهي حالات الاختبار التي تقترحها هنا؟؟

```
1.input(x,y)
2.while(x>0)
{
    3.if(y>0)
       4.y=y+1;
     else
       5.y=y-1;
     6.x = x-1;
}
7.output(y)
```



التعقيد الحلقي

$V(G) = e - n + 2P = 10-9+2=3$

المسارات المستقلة

- المسار المستقل هو مسار يحوي وصلة على الأقل غير موجودة في المسارات السابقة.

Start-1-2-7-end

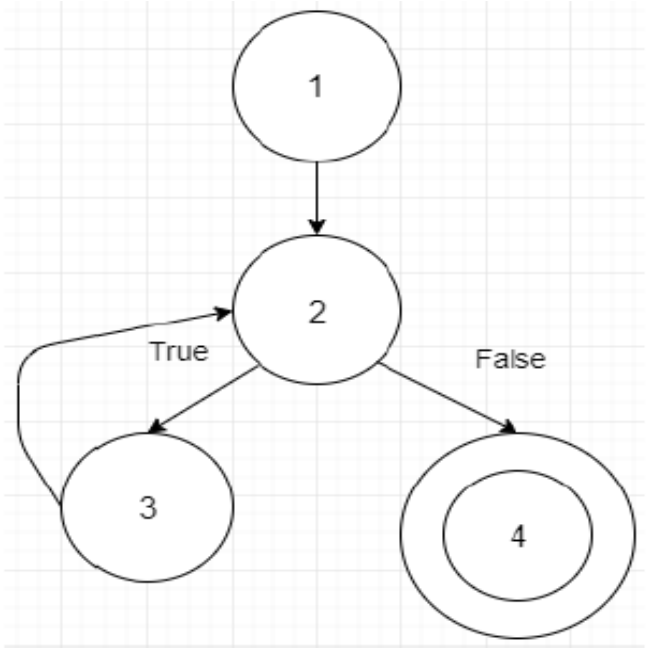
Start-1-2-3-4-6-2-7-end

Start-1-2-3-5-6-2-7-end

عندما **يطلب حالات اختبار** تحقق تغطية عبارة بسبة 100%، وبعد إيجاد المسارات التي تحقق التغطية المطلوبة، **يجب إعطاء قيم للدخل** والتي تجعل البرنامج يسلك هذه المسارات أي الحل هو قيم فعلية للدخل)

مثال ٣

```
int fact(int n)
{
  1  int p=1
  2  for(int i=1;i<=n;i++)
  3  p=p*i
  4  return p
}
```



ماهي حالات الاختبار التي تقترحها هنا؟؟

تحديد المسارات:

1,2,4

1,2,3,2,4

عدد المسارات: cc=2

ما الفرق بين الاختبار الستاتيكي (Static Testing) و الاختبار الديناميكي (Dynamic Testing) في هندسة البرمجيات؟

الاختبار الستاتيكي (Static Testing)

الاختبار الستاتيكي هو عملية **فحص البرمجيات بدون تنفيذ البرنامج نفسه**. بمعنى آخر، تركز على **التحقق من كود البرنامج أو المستندات أو التصميمات** **قبل أن يتم تشغيلها وذلك بهدف** اكتشاف الأخطاء في المراحل المبكرة من التطوير وتقليل تكلفة إصلاح الأخطاء قبل ان تكبر وتزداد تكاليف إصلاحها لاحقًا.

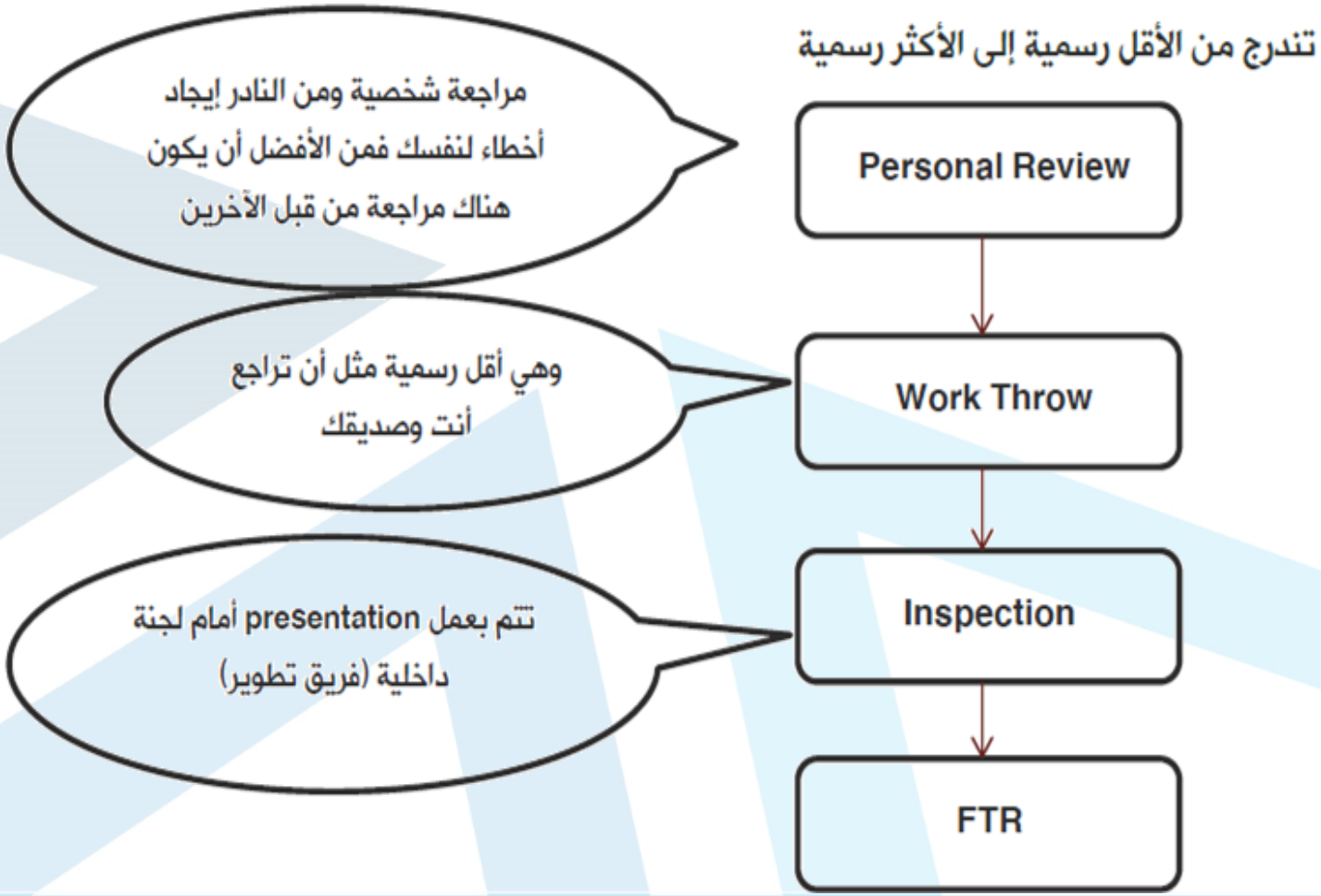
وتعد المراجعات Reviews من اهم طرق الاختبار الستاتيكي وتتضمن:

- مراجعة الكود Code Review
- مراجعة التصميم Design Review
- مراجعة المتطلبات Requirements Review
- مراجعة المستندات Documentation Review

ويحوي الاختبار الستاتيكي التقنيات التالية:

work throw , FTR ,inspecting, Personal Review

تندرج من الأقل رسمية إلى الأكثر رسمية



FTR

Formal Technical Review مراجعة تتم ضمن لجنة من فريق الجودة وتتم عند نهاية كل مرحلة وهي عبارة عن جلسة يقوم فيها فريق التطوير بعرض حول ما تم عمله والتحدث عن الأحداث المهمة ولا يمكن الانتقال من مرحلة إلى مرحلة إلا بأخذ موافقة من فريق الجودة.

بعض الأخطاء التي يمكن أن تظهر عند اختبار الكود بشكل Static:

- Parameter type mismatches: عدم توافق نمط المتغير مع نمط القيمة المسندة له.
- Uncalled functions and procedures: الدوال والإجرائيات مكتوبة وغير مستدعاة.
- Possible array bound violations: تجاوز في حدود المصفوفات
- Syntax Checking: اختبار اتباع القواعد اللغة البرمجة المستخدمة في الكود .
- Use of variables before defining them: استخدام متغيرات قبل تعريفها.
- Variables that are declared but never used: متغيرات معرفة ولكن لم يتم استخدامها.
- Undeclared Variables: متغيرات مستخدمة ولم يتم التصريح عنها.
- Use of variables after they have been killed: استخدام متحولات بعد انتهاء حياتها.

الاختبار الديناميكي (Dynamic Testing)

الاختبار الديناميكي هو عملية **فحص البرمجيات عن طريق تنفيذ البرنامج**، لمعرفة مدى توافق النتائج مع المتطلبات المتوقعة.

ويستخدم للتحقق من أن البرنامج يعمل كما هو مطلوب واكتشاف الأخطاء أثناء تشغيل النظام. وكذلك لتقييم الأداء والكفاءة.

ومن الأمثلة على الاختبار الديناميكي

١. **اختبار الوحدات (Unit Testing) :** اختبار كل وحدة أو دالة على حدة. مثال: اختبار دالة حساب مجموع الأعداد.

٢. **اختبار التكامل (Integration Testing):** اختبار تفاعل الوحدات معًا.

٣. **اختبار النظام (System Testing):** اختبار النظام كاملًا للتحقق من تلبية المتطلبات.

٤. **اختبار القبول (Acceptance Testing):** يتم بواسطة العميل للتحقق من أن النظام يلبي احتياجاته..

مثال توضيحي

لنفترض برنامجًا يحسب متوسط درجات الطلاب.

- **الاختبار الستاتيكي:**
  - فحص الكود للتأكد من أن المعادلة صحيحة، لا توجد أخطاء نحوية أو متغيرات غير معرفة.
- **الاختبار الديناميكي:**
  - تشغيل البرنامج باستخدام مجموعة من الدرجات، والتحقق من أن المخرجات تطابق المتوسط المتوقع.

المعيار	الاختبار الستاتيكي	الاختبار الديناميكي
التنفيذ	بدون تشغيل البرنامج	مع تشغيل البرنامج
الهدف	فحص الكود والمستندات	فحص وظائف البرنامج أثناء التشغيل
الكشف عن الأخطاء	أخطاء التكويد، التصميم، الأسلوب	أخطاء التشغيل والمنطق
التكلفة	منخفضة (لان الكشف يتم مبكرًا)	أعلى إذا تم الكشف في مرحلة لاحقة
الأدوات	Code Review, Static Analysis Tools	JUnit, Selenium, LoadRunner


HAVE ANY
QUESTIONS AND DOUBTS