Software Testing and Dabugging

د.محمد الأحمد

Software Testing and Debugging

التكويد :

تحويل التصميم إلى كود ضمن قيود اللغة البرمجية المطلوب التطوير ضمنها.

أهم شيء جعل الكود Readability مقروء لأنه يوجد أناس سيعملون بالتصليح والصيانة وتطوير النظام من بعدنا.

فالهدف ليس السرعة بالتكويد وإنما الكتابة بطريقة مقروءة واتباع نمط معين بالكتابة وكتابة أسماء المتحولات والتوابع والكلاسات بأسماء ذات معنى مفهومة.

المطلوب من المبرمج بعد مرحلة التكويد القيام بمرحلة تسمى Unit Testing (اختبار الكود)

الاختبار يتم على مستوى الوحدة (تابع/كلاس/مجموعة كلاسات)

مرحلة الTesting :

يمكن تقسيمها إلى :

1. Dynamic : من خلال التنفيذ
2. Static (دون تنفيذ) في المراحل الأولى من تحليل وتصميم ويمكن أن تتم خلال التكويد (يتضمن مراجعة الكود والتأكد من الأسماء وتكويد الأفكار بشكل صحيح)

مثلاً في XP أحد المبرمجين يقوم بالتكويد والآخر يقوم بالاختبار والمراجعة.

Quality Management

(إدارة الجودة) :هي umbrella Activity وهي مجموعة نشاطات كل نشاط يشمل كل مراحل التطوير بدءاً من التحليل وانتهاءً بالتصميم وتتم من خلال تقليل الأخطاء عن طريق :

1. Quality assurance: تتم في مراحل التحليل والتصميم من خلال اتباع قوالب معينة، عبارة عن مجموعة من السياسات التي يضعها فريق الجودة ويجري عمله بالتوازي مع فريق التطوير من خلال التدقيق والمراجعة والمراقبة وحتى فريق التطوير يعتبر جزء من فريق الجودة وبالتالي هي استراتيجية لمنع الأخطاء.

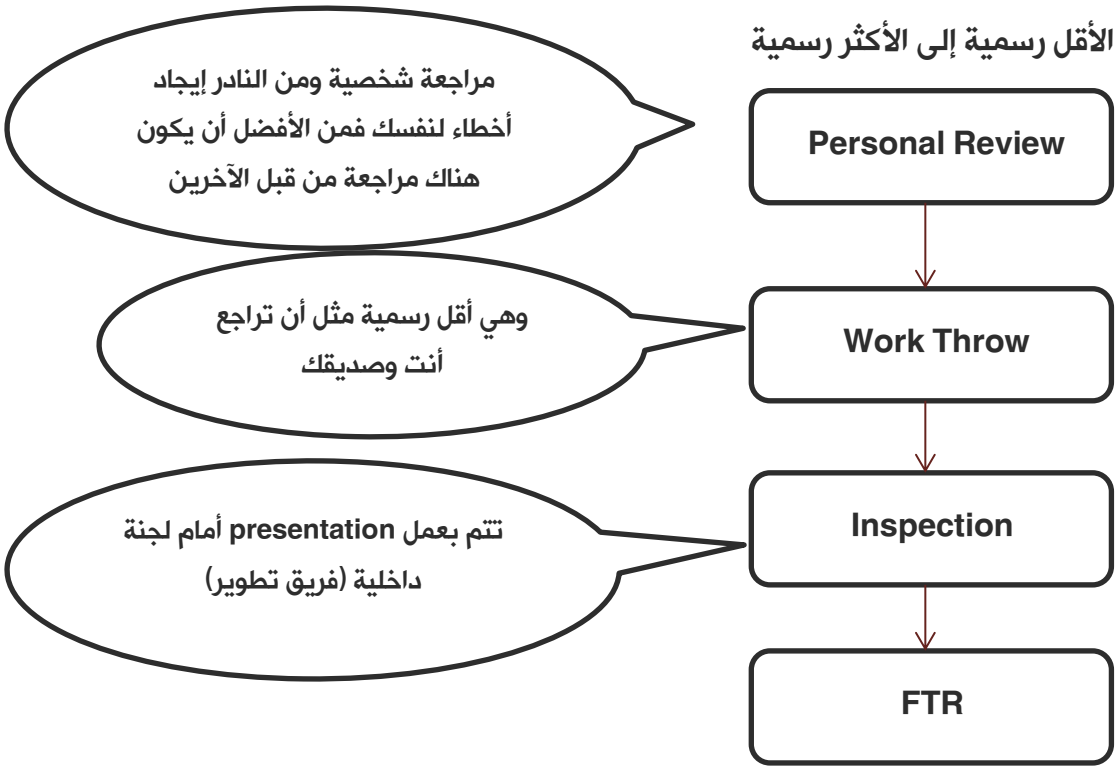
2. Quality control (Testing): هي سياسة تصحيحية للأخطاء وأحد أهم نشاطاتها هي testing حيث الهدف منه إيجاد الأخطاء وتصحيحها من خلال debugging

يعتبر النظام ناجح بقدر ما يتم اكتشاف أخطاء وتصحيحها حيث لا يمكن البرهان على عدم وجود أخطاء.

التقنيات الموجودة في static testing

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.

أهمية الـstatic Review : توفر علينا كلفة إصلاح الأخطاء لو انتقلت إلى مرحلة testing dynamic يصبح أصعب فاكتشاف الأخطاء بنفس المرحلة يكون أقل كلفة وأقل وقت ومن الصعب أيضاً ايجادها بغير مرحلة مما يبقي هذه الأخطاء ضمن النظام وتنكشف عند الوصول إلى الزبون مما يعطي انطباع سيئ عن العمل.

تلعب المراجعة دور فلتر (تصفية) للأخطاء ويتم اكتشاف الأخطاء مرحلياً.

Dynamic testing

تأتي بعد التكويد مباشرة (بالتنفيذ) ويتم عن طريق تصميم حالات اختبار.

أهم الاستراتيجيات:

- BBT: Black box Testing
- WBT: Whit box Testing
- GBT: Grey box Testing

WBT : اشتقاق حالات الاختبار من الكود والخوارزميات ويجب في هذه الحالة فهم الكود جيداً ووضع ال Test Case والذي يشمل ال exception و input data

BBT : ممنوع الاقتراب من الكود وفهمه فقط يتم اشتقاق الحالات من الSpecification

GBT: وسمي Grey لان بعض أجزاء النظام قد يكون جاهز بشكل كامل فلا يمكن أن يظهر إلا بشكل Binary فلا نستطيع في هذه الحالة تصحيح الكود وهناك أجزاء من النظام نحن من قمنا بتطويرها فنختبره عبر WBT ، فهو يشمل WBT بالإضافة إلى BBT فيصبح لدينا GBT.

كل تطبيقات الويب تختبر *GBT* لأنه لدينا كثير من الأجزاء الجاهزة.

مهما تكون الاستراتيجية يجب أن نضع test case وهو عبارة عن input data + التوقع expectation نقوم بداية بإدخال البيانات input Data ثم نقارن ال Result مع التوقع expectation ففي حال التطابق يكون الاختبار فشل ولو كان هناك اختلاف بالنتيجة يكون قد اكتشفنا خطأ ونجح الاختبار.

مثال : ادخال عددين صحيحين وإخراج جمعهما

الدخل	التوقع
$a, b \in Z$.	$a + b = c$
2,3	2+3=5

مثال : اختبار login

Login	×	×	✔	✔
Password	×	✔	×	✔

في الحالات الثلاث الأولى يجب ألا يسمح بالدخول وفي الحالة الأخيرة يسمح بالدخول وغير ذلك يوجد خطأ في الكود يجب تصحيحه.

أنواع الأخطاء :

- **Error** : هو خطأ بشري. مثلاً :خطأ في التصميم، خطأ في التكويد، خطأ في فهم المتطلبات.

- **Fault** : هو Error ولكن داخل البرمجية وهو خطأ كامل قد يظهر وقد لا يظهر ،وهو مفهوم داخل ال software.
- **Failure** : هو تنفيذ الFault وهو مفهوم تنفيذي.
- **Bug** : الأخطاء المحصورة في الكود.
- **Defect** : الأخطاء التي حدثت عند الزبون (عند تسليم النظام).

مثال:

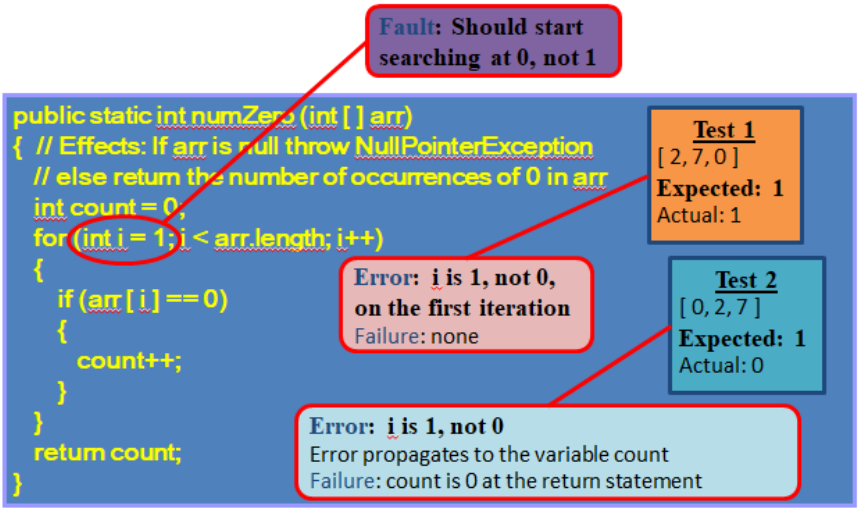
LOC	Code
1	program double ();
2	var x,y: integer;
3	begin
4	read(x);
5	y := x * x;
6	write(y)
7	end

Failure : $x = 3$ يكون $y = 9$ هذا خطأ لأن الناتج الصحيح هو 6

Fault : العملية الصحيحة في السطر الخامس هي $+$ بدل $*$

Error: خطأ في الطباعة المبرمج كتب $+$ بدل $*$ أو خطأ مفاهيمي المبرمج لا يعرف كيفية مضاعفة عدد

A Concrete Example



Dynamic Testing

Test to pass	Test to Fail
الداتا صحيحة لن تعطي أخطاء لو أعطت خطأ ما يكون يوجد خطأ	الداتا غير صحيحة يجب أن يعطي خطأ ويعطي رسالة خطأ

المطلوب تنفيذ شامل لإيجاد جميع الاحتمالات الممكنة حتى يكون النظام خالٍ من الأخطاء.

مستويات الاختبار Testing level

نبدأ من المراحل الأصغر حتى الأكبر:

1. Unit Testing (function, class)

يقوم بها المبرمج ويستخدم WBT ويمكن أن يستخدم BBT لكن WBT هي الأساسية
بعد كتابة كل function و Class يجب أن يقوم باختباره.
المطلوب : كشف الأخطاء في ال Unit وإصلاحها .

2. Integration Testing

اختبار عمل الأنظمة الجزئية مع بعضها البعض حتى يصبح النظام مع بعض ،يقوم به فريق testing group
يساعدهم المبرمج بشكل ثانوي ، يستخدم ال GBT .
يمكن معرفة كيفية ربط الأنظمة الجزئية ببعضها عن طريق العودة إلى معمارية النظام .
المطلوب : كشف الأخطاء في تواصل الأنظمة الجزئية مع بعضها البعض ،.

مثال:

Function يستدعي Function آخر أو Class يستدعي Class آخر وهكذا.

3. System Testing

يقوم به فريق الاختبار(testing group) ، يستخدم بالمطلق BBT ، وهو مجموعة كبيرة من الاختبارات التي تختبر عمل النظام والقيود عليه (المتطلبات غير الوظيفية) .

مثال:

نختبر security الخاصة بالنظام, نختبر Reliability الخاصة بالنظام ,نختبر الوظائف الخاصة بالنظام، نختبر Performance الخاص بالنظام ،وكلها عن طريق الـ BBT حسب الـ specification .

4. Accepture Testing

يقوم به الزبون (المستخدم) بشكل عام.
قبل تسليم النظام للزبون يتم الاختبار من النوع BBT للتأكد من الخدمات المتفق عليها وهل عملها صحيح ؟؟
وله نوعان :
α : يقوم به المستخدم ببيئة التطوير (ضمن الشركة) .

β :نفس الاختبار ولكن ضمن بيئة العمل .

ملاحظة: لكل من الWBT والBBT أهميته ولا يمكن تفضيل أحدهما على الآخر وهما متكاملان ونحن بحاجة لكليهما للتأكد من نجاح النظام كل واحد منهما يستطيع كشف نوع معين من الأخطاء ولا يستطيع الثاني اكتشافه.

مثال : BBT يكتشف الخدمات المفقودة ، WBTيدخل على الكود ويكتشف الأخطاء.

Unit :

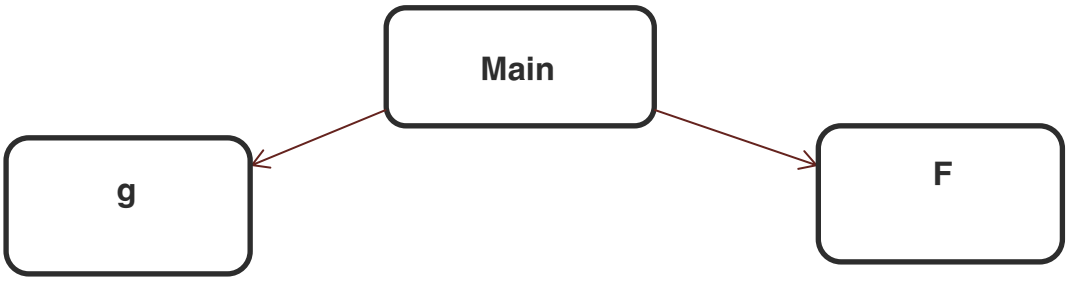
يمكن استخدام open Source و tool جاهزة لتوفير وقت الاختبار مثل J-Unit للجافا ويمكن أن يتم يدوياً على نفس البيئة المستخدمة للتطوير.

Integration:

يتم عن طريق استراتيجيتين مهمتان:

1. **Big Bang الانفجار الأعظم** : تتطلب أن تكون جميع الوحدات المكونة للنظام جاهزة وفقاً للمعمارية بالتعاون مع Hardware ويجب أن تكون جميع الUnit جاهزة ثم نقوم بتركيبهم إذا عمل النظام إذاً لا يوجد أخطاء وهي استراتيجية فاشلة لأنه فقط يرى إذا كان النظام يعمل أو لا ويحتاج لأن تكون جميع الunit جاهزة وهو أمر صعب جداً.
2. **الاستراتيجية التزايدية** : وهي الأفضل وتتم على ثلاث مستويات :
   - **Top-down**
   - **Bottom-up**
   - **Sandwich**

النظام على شكل أنظمة جزئية مرتبة في مستويات



- **Top-down** : (النقص في الأجزاء السفلى)

التابع F لو كان غير جاهز نستخدم فكرة stub وعندما تتوفر الأجزاء الناقصة نعيد اختبارها

فكرة stub: وهي عبارة عن كود مزيف dummy code يحاكي عمل الأجزاء الناقصة وهو عبارة عن low level model يساعد في اختبار التكامل top-down

- **Bottom-up** : (النقص في الأجزاء الأعلى)

نقوم بعمل Dummy code ولكن يدعى Driver .

فكرة Driver: وهي عبارة عن كود مزيف dummy code يحاكي عمل الأجزاء الناقصة وهو عبارة عن high level model يساعد في اختبار التكامل bottom-up

- Sandwich : (النقص في الأجزاء الوسطى)

هو مزيج من الطريقتين السابقتين بعمل stup و driver معا .

System

A- التقنيات المستخدمة في BBT

1. تقنية صفوف التكافؤ.
2. تقنية الحدود.
3. الخبرة.

1 تقنية صفوف التكافؤ

المطلوب : فهم فضاء الدخل ثم نقسمه إلى صفوف يجب أن تكون غير متقاطعة وجميعها تشكل فضاء الدخل

مثال : تابع الجذر

$$R :]-\infty, 0[\ ]0, +\infty[$$

نأخذ عنصر من المجال الأول (لا يوجد جذر) وهو مجال test to fail

نأخذ عنصر من المجال الثاني وهذا يكفي (يوجد جذر) وهو مجال test to pass

أي نأخذ عنصر من كل صف ونجري الاختبار عليه ولا يوجد طريقة معينة لتقسيم الفضاء (فقط نفهم فضاء الدخل ونحاول تقسيمه) .

2 تقنية الحدود :

نأخذ الحدود بين الصفوف فمعظم الأخطاء تكون عند الحدود

مثال : بالجذر نأخذ الدخل 5 لأنه من الحدود ونأخذ 4+ , 4-

3 الخبرة :

نعتمد على الخبرة لاستنتاج حالات الاختبار كل مسألة لها حالات معينة

مثال : login

Id	✔	✔	×	×
Password	✔	×	✔	×

قد لا يتم إدخال أي ID, Password فهي حالة تتكرر كثيراً ولا ينتبه لها المبرمج ،ولكن تراكم الخبرة تجعلنا نقوم باختبارها.

B- التقنيات المستخدمة في WBT: تقنية واحدة وهي التغطية

المطلوب تغطية كل تعليمة موجودة بالكود (تنفيذها) على الأقل مرة واحدة التعليمة التي لا تنفذ لا يمكن معرفة كونها صحيحة أو خاطئة ولها أنواع:

1. تغطية تعليمة.
2. تغطية الفرع والشرط.
3. تغطية المسار.

كل كود هو مجموعة من مسارات التنفيذ والمطلوب هو تحديد هذه المسارات ورسمها ضمن control flow Graph وهو بيان له بداية واحدة ونهاية واحدة يمثل كل مسارات تنفيذ الكود.

تقنية المسارات المستقلة :

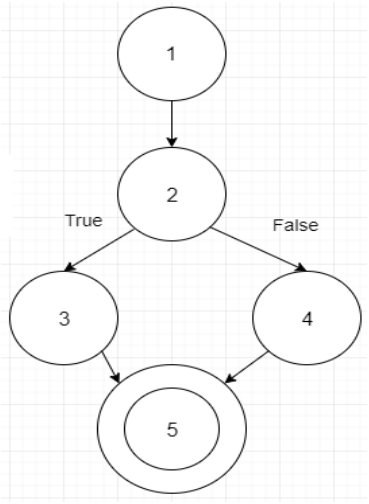
خوارزمية الاختبار باستخدام تقنية المسارات المستقلة :

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.

CFG



2. تحديد المسارات ،

1,2,3,5

1,2,4,5

وعدد المسارات : هو عدد يمثل تعقيد الكود **Cyclomatic Complexity**.

1) عدد المسارات المستقلة $= cc$

2) $cc = P + 1$

حيث ال p: عدد العقد المتفرعة

3) $cc = R + 1$

حيث ال R : عدد المناطق المغلقة

4) $cc = E - N + 2$

حيث ال N : عدد العقد

E : عدد الوصلات

5) مصفوفة التجاور:

وهي مصفوفة مربعة بعدها هو عدد العقد تعبر عن عدد الوصلات بين العقد

$$\begin{array}{cc} & \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \end{pmatrix} \\ \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{pmatrix} & \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

نجمع عناصر كل سطر ثم نطرح 1

السطر الأول : $1 - 1 = 0$

السطر الثاني: $2 - 1 = 1$

السطر الثالث: $1 - 1 = 0$

السطر الرابع: $1 - 1 = 0$

السطر الخامس هو سطر صفري لا يدخل في الحساب

نجمع النواتج عن العمليات السابقة ونجمع للناتج 1 فيكون :

$$1 + 1 = 2$$

على مثالنا السابق CFG ناتج كل الطرق السابقة يساوي 2 .

المقاييس البرمجية:

سهل الاختبار < 10

متوسط الاختبار < 20 < 10

صعب الاختبار < 40 < 20

مستحيل الاختبار < 40

3. يجب وضع حالات اختبار لكل مسار :

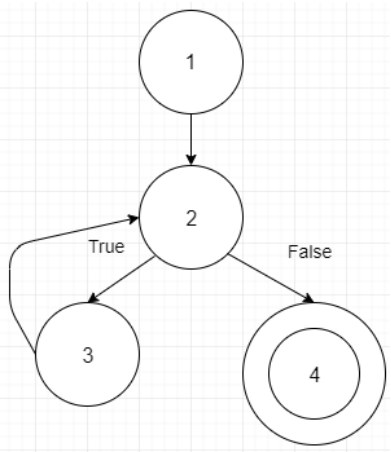
مثال : مع حلقات :

```
int fact(int n)
{
  1  int p = 1
  2  if(n=0)
  3  return p
     else
  4  return n*fact(n-1)
}
```

تكرارياً:

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

CFG للتابع التكراري:



تحديد المسارات:

1,2,4

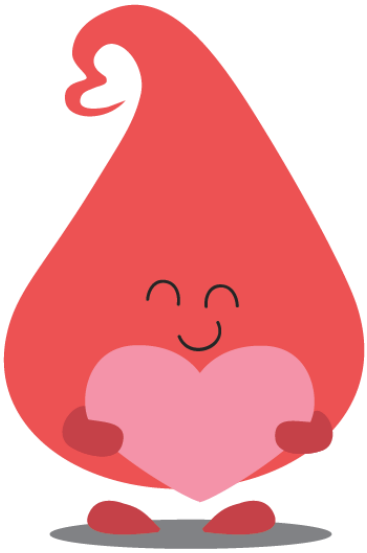
1,2,3,2,4

عدد المسارات: cc=2

Debugging

البحث عن الخطأ:

- خوارزمياتها البحث الشامل.
- تتبع الخطأ.
- بعد تصحيح الخطأ نقوم بالاختبار العودي وذلك بإعادة عمليات الاختبار للتأكد من عدم وجود أخطاء أثناء التصحيح.



تحلّى بالصبرْ