بسم الله الرحمن الرحيم

مقرر هندسة البرمجيات
١٤٩٤

الوحدة السابعة

فحص البرمجيات أو اختبارها
Software Testing

إعداد: م. هناء قشطة

الفصل الدراسي الأول

٢٠٢١م-٢٠٢٢م

أهداف اللقاء

- شرح طبيعة الأخطاء التي تحدث فيها البرمجيات.
- استخدام أساليب اختبار البرمجيات وطرق تحليلها والأسلوب الديناميكي والأسلوب الثابت.
- استعراض الوسائل التي يمكن استخدامها في اختبار البرمجيات.
- كتحليل تدفق البيانات خلال النظام، وفحص وظيفي للبرمجيات.
- مناقشة استراتيجيات الاختبار، وتكامل البرمجيات.
- شرح مقاييس البرمجيات.

٢. طبيعة الأخطاء

- **تتعرض البرمجيات لأنواع عديدة من الأخطاء:**
  - **منها المنطقية و التي تنجم عن سببين:**
    - البيانات المدخلة في البداية تكون في أفضل صور الاختصار.
    - حدوث تكرار Loops لعدد غير صحيح من المرات.
- و قد تكون الأخطاء ناجمة عن تصميم نظام برمجي يخالف المواصفات المطلوبة، و ذلك بسبب سوء الفهم بين المستفيد و مصممي النظام، و تنجم هذه الأخطاء عن **ثلاثة أسباب:**

١. الغموض.

٢. عدم التكامل.

٣. الأخطاء في تحديد المتطلبات التي يريدها المستفيد.

فقد يتفق الطرفان على صيغة معينة، ولكن كلاً منهما يفهم هذه الصيغة من زاوية معينة، و تبدأ مراحل التطوير و تتراكم تلك الأخطاء حتى ينجم نظام يختلف بصورة كاملة عن النظام الفعلي المطلوب.

٢. طبيعة الأخطاء

- من أجل تجنب الأخطاء، لابد من التحقق من صحة البرمجيات و تدقيقها خلال كل مرحلة من مراحل تطويرها، و ذلك للتأكد من أن البرمجيات تلبي المواصفات المتفق عليها، و أنها تلبيها كما يجب.
- **التحقق و تدقيق صحة البرمجيات لهما هدفان:**
  - اكتشاف العيوب في النظام.
  - تقييم إمكانية استخدام النظام عمليًا.
- **التحقق من صحة البرمجيات و تدقيقها يتكون من شقين:**
  - **التحقق من صحة البرمجيات Validation:**

يتضمن مجموعة من الأساليب للتأكد من بناء النظام المطلوب حقًا (أي أن المنتج يغطي كل متطلبات الزبون).

  - **تدقيق البرمجيات Verification:**

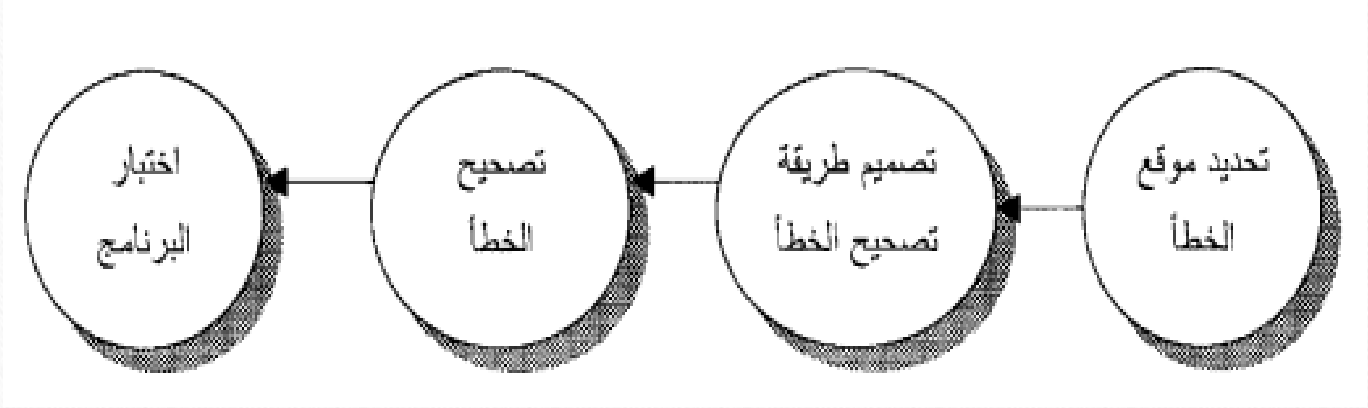
يشتمل أساليب للتأكد من بناء النظام بصورة صحيحة و سليمة (من الناحية البرمجية و التقنية).

٢. طبيعة الأخطاء

- تفيد الأساليب المستخدمة في مرحلة التحليل، مثل التجارب، في اكتشاف الأخطاء، و لكن في بعض الأحيان لا يمكن اكتشاف توافق النظام مع المواصفات إلا بعد وضعه حيز التطبيق.
- تستخدم أساليب ثابتة و أساليب ديناميكية لتأمين سلامة البرمجيات.
  - **الأسلوب الثابت:** يتعلق بتحويل النظام بصورة عامة مثل تفحص المواصفات، تفحص البرنامج و متابعته (شكل١ ص١٦٢).
  - **الأسلوب الديناميكي:** يتضمن اختبار البرنامج من خلال تنفيذه و ملاحظة النتائج (شكل٢ ص١٦٢).
- لا يمكن استبدال الأساليب الديناميكية بالأساليب الثابتة، و ذلك:
  - للتأكد من أن التعديل الجديد قد نفذ بصورة سليمة.
  - للتأكد من عدم استحداث عيوب جديدة.

٢. طبيعة الأخطاء

- يتم التحقق من صحة البرمجيات من خلال التأكد من خلوها من الأخطاء، حيث يتم تنفيذ البرنامج باستخدام بيانات متنوعة مع مراقبة نتائجه، و هذا الاختبار يقوم بتحديد العيوب فقط، أما عملية اكتشاف الأخطاء و تصحيحها فتسمى تشخيص الأخطاء Debugging.
- **تشخيص الأخطاء و تصحيحها Debugging:** مجوعة من الأساليب لاكتشاف الأخطاء و تصحيحها، و تتضمن عملية التشخيص و التصحيح عدة خطوات:



حيث يتم تحديد الخطأ، ثم تعديل البرنامج بتصحيحه، ثم يعاد اختبار البرنامج للتأكد من صحة التعديلات الجديدة.

٢. طبيعة الأخطاء

اختبار البرمجيات و مراحله:

- البرمجيات اليوم كبيرة و متعددة الجوانب، و لا يمكن اختبارها كتلةً واحدة، حيث يتم اختبار أقسامها كل على حدة:
  - البرنامج قد يتكون من مجموعة من البرامج الجزئية Sub-System.
  - البرنامج الجزئي قد يتكون من مجموعة من المقاطع Module.
  - المقطع قد يتكون من مجموعة من البرامج الفرعية Unit.
- **عملية الفحص تتم على شكل مراحل كما يلي (شكل٤ ص١٦٤):**
  - **فحص البرامج الفرعية Unit Testing:**

فحص البرامج الفرعية بصورة انفرادية للتأكد من سلامتها، و أنها تؤدي المهمة المطلوبة.

- **فحص المقاطع Module:**

المقطع هو مجموعة من البرامج الفرعية التي تؤدي بمجملها وظيفة معينة، حيث يتم فحصها بصورة مستقلة.

- **فحص البرامج الجزئية Sub-System:**

بعد دمج مجموعة المقاطع، نحصل على برنامج جزئي، يتم فحصه و وضعه قيد التنفيذ بصورة مستقلة عن بقية الأنظمة الجزئية.

٢. طبيعة الأخطاء

اختبار البرمجيات و مراحله:

- عملية الفحص تتم على شكل مراحل كما يلي (تابع):
  - **فحص المنتج النهائي System:**

بعد دمج البرامج الجزئية مع بعضها، ينتج المنتج النهائي أو النظام الكامل، فيتم فحصه، و التأكد من سلامته، و قد تظهر الأخطاء في المنتج النهائي بسبب التداخل بين المكونات المختلفة للنظام.

  - **فحص القبول Acceptance Testing:**

هذا الفحص الأخير قبل وضع النظام قيد التطبيق، حيث يتم في هذا الفحص تغذية النظام ببيانات حقيقية بدلاً من البيانات المشابهة.

٢. طبيعة الأخطاء

اختبار البرمجيات و مراحله:

- **فحص ألفا Alpha Testing:**

يطلق على فحص القبول بفحص ألفا، و تسمى مجموعة الفحوص من بداية تطوير النظام حتى قبول المستفيد و موافقته عليه بفحص ألفا.

- **فحص بيتا Beta Testing:**

مجموعة من الاختبارات التي تجري على المنتج الذي يتم تسويقه، حيث في بعض الأحيان تقوم الشركات بعرض منتجاتها للتسويق، و بيعها إلى العديد من الزبائن بقصد فحصها و اختبارها و تحديد العيوب فيها، و بعد تلقي الشركة ردود فعل الزبائن يتم تطوير النظام، و عرضه إما للفحص مرة أخرى، أو للتسويق.

٢. طبيعة الأخطاء

اختبار البرمجيات و مراحله:

- عندما يتم اكتشاف العيوب في مرحلة من مراحل الفحص يعدل البرنامج، و يفحص من جديد، و قد يتطلب الأمر تعديل البرنامج في المراحل السابقة و اللاحقة (شكل ٥ ص١٦٥).
- **فحص الانحدار:**

قد يؤدي اكتشاف العيوب و تصحيحها إلى اكتشاف عيوب أخرى، و لذلك يجب إعادة الفحص بعد كل تعديل، و يسمى هذا النوع من الفحص بفحص الانحدار.

حيث يعاد الفحص في كل مرة يجري فيها تعديل البرمجيات، و ذلك للتأكد:

- من أن التعديل الجديد قد نفذ بصورة سليمة.
- من عدم استحداث عيوب جديدة بعد إتمام التعديل الجديد.

٢. طبيعة الأخطاء

تقنيات اختبار البرمجيات

- يجب أن يكون فحص البرمجيات شاملاً، بحيث يتم تتبع كل جملة من جمل البرنامج، و كل مسار من مساراته.

- **يوجد أسلوبان لذلك:**
  - الفحص الوظيفي Functional Testing أو فحص الصندوق الأسود Black-Box Testing، و الذي يتعلق بموافقة مواصفات البرنامج.
  - الفحص التركيبي Structural Testing أو فحص الصندوق الأبيض White-Box Testing، و الذي يتعلق بتنفيذ البرنامج.

٢. طبيعة الأخطاء

تقنيات اختبار البرمجيات

- **الفحص الوظيفي Functional Testing:**

يتضمن استخدام مواصفات البرنامج لمعرفة نتائج الاختبار.

يعرف كذلك بالصندوق الأسود، لأن التحقق من صحة البرنامج يتم من خلال معرفة المدخلات و المخرجات فقط، و يتم اختبار المدخلات التي يعتقد أنها تسبب اكتشاف الأخطاء، أو نموذج من البيانات التي يمكن أن تمثل كل أنواع البيانات الممكن استخدامها (شكل ٦ ص ١٦٧).

٢. طبيعة الأخطاء

تقنيات اختبار البرمجيات

- **الفحص التركيبي Structural Testing:**

يعرف كذلك بالصندوق الأبيض، لأن الفاحص يستطيع معرفة معلومات عن المدخلات و عن البرنامج نفسه و عن المخرجات، و ذلك بتفحص كل مسار من مسارات البرنامج.

يتضمن هذا الفحص اختبار:

- كل جملة من جمل البرنامج.
- كل جملة شرطية للحالات الصائبة و الخطأ.

تعتبر طريقة Cyclomatic Complexity هي أفضل طريقة لإتمام هذا الفحص، حيث يتم من خلالها تحديد عدد مرات الفحص، فقيمة Cyclomatic تمثل عدد المسارات التي تحتاجها لإجراء فحص شامل لكل العبارات دون الدخول في حالات تكرار.

٢. طبيعة الأخطاء

تقنيات اختبار البرمجيات

و يمكن استخدام المخططات كمخططات تدفق البيانات لإجراء هذا الاختبار(شكل٧ ص١٦٨).

مثال١ ص١٦٨.

يتناسب عدد المسارات مع حجم البرنامج، يفيد الاختبار التركيبي في البرامج الفرعية، و قد لا يصلح للبرنامج الكبير الناتج عن دمج مجموعة من البرامج، أو المقاطع.

٣. استراتيجيات اختبار البرمجيات (تكامل الفحص)

- **يتم تدقيق البرمجيات (Verification) على مرحلتين:**
  - **الأولى:** أخذ أجزاء من المنتج أو نماذج عنها بشكل مفصل و فحصها.
  - **الثانية:** فحص المنتج ككل و تسمى مرحلة تكامل الفحص.
- **يعتمد اختيار طريقة فحص البرمجيات على عدة أمور منها:**
  - طبيعة تنظيم البرنامج والتطبيق.
  - العمل الهندسي الخاص بالمشروع تحت التنفيذ.
- **تتضمن استراتيجيات اختبار البرمجيات طريقتين:**
  - الطريقة التنازلية Top-down Testing.
  - الطريقة التصاعدية Bottom-up Testing.

٣. استراتيجيات اختبار البرمجيات (تكامل الفحص)

- **الفحص التنازلي Top-down Testing:**

يبدأ الفحص التنازلي من المستوى الأعلى، حيث يتم اختبار البرامج الجزئية في المستوى الأول، ثم البرامج في المستوى الثاني، ثم الفروع حتى آخر برنامج فرعي في آخر مستوى، (شكل٩ ص١٧١).

- يقترن هذا الفحص بمفهوم التجزئة، حيث يتم فحص المقاطع البرمجية و اختبارها حال تحويل التصميم إلى لغة برمجة.
- **الفحص التنازلي له عدة ميزات:**
  - يفيد في اكتشاف الأخطاء في مراحلها الأولى، مما يقلل من تكاليف التصميم و التحويل.
  - الحصول على نظام هيكلي شبه جاهز للعمل، مما يعطي للمصممين حالة اطمئنان على تقدم العمل، و رؤية الهيكل العام للنظام المقترح و عرض صحته و تدقيقها.

٣. استراتيجيات اختبار البرمجيات (تكامل الفحص)

- **للفحص التنازلي من ناحية أخرى معوقات:**
  - قد يصعب تطبيق هذا النوع من الفحص، لأن الاختبارات المتعلقة بالمقاطع في المستويات الدنيا تتطلب بيانات حقيقية من أجل فحصها بصورة دقيقة، و قد لا يمكن توفير هذه البيانات في الأوقات المبكرة.
  - صعوبة ملاحظة نتائج المستويات العليا، لأنها في أغلب الأحيان ليست مخصصة لعرض نتائج، لذلك على الفاحص تطويع هذه المستويات ظاهريًا لعرض نتائج معينة.

٣. استراتيجيات اختبار البرمجيات (تكامل الفحص)

- **الفحص التصاعدي Bottom-up Testing:**

يبدأ الفحص التصاعدي من المستوى الأدنى، حيث فحص المقاطع في المستويات الدنيا أولاً، ثم ربطها مع بعضها حسب الحاجة في المستويات الأعلى منها، (شكل١٠ ص ١٧٢).

- يفيد الفحص التصاعدي في توفير مجموعة من الفعاليات (البرامج الفرعية أو المقاطع) التي تكون جاهزة للاستخدام، و لكن هذه تحتاج إلى كتابة مقاطع لاختبارها، و قد يتطلب الأمر توزيع مقاطع الاختبار هذه على المستفيدين، من اجل تجريب المقاطع باستخدام بيانات متباينة.
- يعتبر الاختبار التصاعدي أفضل من الاختبار التنازلي، بالرغم من اشتماله على بعض الصعوبات التي تواجه الفاحصين خلاله، و ذلك لأن الفحص التصاعدي يأخذ بعين الاعتبار إمكانية إعداد برامج فرعية أو مقاطع جاهزة، و هذا يؤدي إلى تهيئة اللبنات الأساسية في البناء، و بعد ذلك تركيبها للوصول إلى البناء الكامل.

٣. استراتيجيات اختبار البرمجيات (تكامل الفحص)

- هناك أساليب أخرى لفحص البرمجيات، **مثل:**

١. الفحص الخيطي (Thread Testing).

٢. فحص التحمل (Stress Testing).

٣. استراتيجيات اختبار البرمجيات (تكامل الفحص)

متطلبات اختبار البرمجيات

- هذا الأسلوب يستخدم في فحص المتطلبات، و المواصفات، و التصميم و الخطط، و كل ما يتعلق بنتائج دورة حياة النظام البرمجي.
- **للقيام بفحص البرنامج بصورة فعالة يتم اتباع ما يلي:**
  - يجب أن تكون مواصفات البرمجيات متوفرة، و يسهل الوصول إليها، للرجوع إليها عند الحاجة.
  - معرفة فريق الفحص بالمعايير المعتمدة لدى المؤسسة.
  - النسخة المنقحة من البرنامج الخالي من الأخطاء القواعدية.
  - رغبة الإدارة بقبول نتائج الفحص الثابت.
  - اعتبار الفحص ركنًا من أركان التحقق من البرمجيات و تدقيقها، و أن لا تؤثر نتائج الفحص في توظيف العاملين في المشروع.
  - تعرض نتائج الفحص على العاملين في المشروع من أجل تعديلها، و يعاد الفحص مرة أخرى للتأكد من سلامة التعديلات.

٣. استراتيجيات اختبار البرمجيات (تكامل الفحص)

تكاليف اختبار البرمجيات

- كان فحص البرمجيات في بداية عصر الحاسوب لا يعتمد على طرق و أساليب معروفة، **حيث:**
  - كان يتبع أسلوب قراءة البرنامج و المرور عبر مقاطعه المختلفة، ثم تحديد موقع الخطأ و تصحيحه.
  - و كان هذا يكلف جهدًا كبيرًا، و العامل الاقتصادي لم يكن له دور ي اتخاذ القرار.
- في العصر الحالي تطورت صناعة البرمجيات بصورة واسعة، **حيث:**
  - اعتمدت طرقًا فنية في التحليل و التصميم و الاختبار.
  - بالتالي أصبح للعامل الاقتصادي دور رئيسي.
- تحسب تكاليف الفحص بإيجاد مجموع الفترات الزمنية التي يستغرقها الفحص، و يضاف إليها التكاليف غير المباشرة التي تؤثر سلبًا أو إيجابًا في عملية الفحص.

٣. استراتيجيات اختبار البرمجيات (تكامل الفحص)

تكاليف اختبار البرمجيات

- حساب التكاليف لم يكن بالأمر اليسير، خاصةً عندما يتم اختبار نسخ متعددة، و في آن واحد.
- الوقت المصروف في عمليات الفحص يكون عاليًا في بداية الدورة، و يستمر بالانخفاض بنمط معين، و يرتفع مرة أخرى في نهاية فترة الفحص (شكل ١١ ص١٧٤).

٤. مقاييس البرمجيات Software Metrics

- تعتبر مقاييس البرمجيات من المقاييس المهمة لأنها تغطي مساحة تغطي مساحة واسعة من الحسابات الخاصة بالمنتج، فهي يمكن أن تعطي تخمينًا لنوعية السيطرة، و مستوى التعقيد، و هي تساعد في تحسين نوعية المنتج.
- **يوجد نوعين من المقاييس و هي:**
  - مقاييس التعقيد.
  - مقاييس النوعية.

٤. مقاييس البرمجيات Software Metrics

مقاييس التعقيد:

- تستخدم للحصول على تنبؤ المعلومات التي تخص صحة البرمجيات و إدارتها من خلال التحليل الذاتي للبرنامج المصدري، و يوفر قياس التعقيد إمكانية العودة على البرمجيات في مراحل الفحص و الإدامة.
- أبسط أنواع مقاييس التعقيد هي مقاييس الحجم: حيث تحسب عدد أسطر البرنامج (Line of Code (LOC))، فكلما زاد عدد الأسطر زاد التعقيد، و هي بالتأكيد طريقة لا تعطي صورة واضحة عن مدى تعقيد البرنامج.
- يوجد طريقتين أساسيتين لحساب مقاييس التعقيد، **هما:**
  - طريقة هوليستيد.
  - طريقة مكابي.

٤. مقاييس البرمجيات Software Metrics

مقاييس هولستيد

- يعد مقياس هولستيد من المقاييس الشاملة التي تساعد على تقدير متانة البرمجيات و قوتها جزءًا بعد جزء.
- يعتمد المقياس على تعيين قوانين كمية لمراحل تطوير برمجيات الحاسوب.
- فيما يلي مجموعة من المتغيرات التي تمثل كل منها حالة معينة:
  - n1 عدد العمليات Operators غير المكررة الموجودة في البرنامج (مثل الجمع، و الطرح، و العمليات المنطقية، و غيرها).
  - n2 عدد المعاملات Operands غير المكررة في البرنامج.
  - N1 العدد الكلي للعمليات.
  - N2 العدد الكلي للمعاملات.

٤. مقاييس البرمجيات Software Metrics

مقاييس هولستيد

مثال/ اقرأ البرنامج التالي المكتوب بلغة سي، ثم احسب عدد العمليات غير المكررة، و عدد المعاملات غير المكررة، و العدد الكلي للعمليات، و العدد الكلي للمعاملات.

```
int A[10];
int  total=0;   float avg=0;
{
for (int w=0; w<10; w++)
{
cin >> A[w];
total+=A[w];
}
avg=total/10;
}
```

٤. مقاييس البرمجيات Software Metrics

مقاييس هولستيد

N2	عدد المعاملات غير المكررة N2	N1	عدد العمليات غير المكررة N1
2	A	3	[]
3	total	5	=
1	avg	2	{
3	W	1	For
1	0	6	;
2	10	1	+
		2	}
		1	/
12	6	21	8

٤. مقاييس البرمجيات Software Metrics

مقاييس هولستيد

- المقاييس التي تستخدم لتقدير درجة التعقيد هي طول البرنامج، و الحجم (يختلف باختلاف لغة البرمجة المستخدمة)، و نسبة الحجم .
- **من الجدول السابق يتضح أن:**

طول البرنامج (N) = $n1 \log2(n1) + n2 \log2(n2)$

$= 8 \log2(8) + 6 \log2(6)$

حجم البرنامج (V) = $N * \log2(n1+n2)$

$= 33 * \log2(14)$

نسبة الحجم (L) = $(2/n1 \times (n2/N2))$

٤. مقاييس البرمجيات Software Metrics

مقاييس مكابي

- يقيس مقياس مكابي عدد التفرعات المنطقية لنموذج واحد من البرمجيات.
- يستخدم هذا المقياس لسببين رئيسيين **هما**:
  - يقدم توصية لعدد الفحوصات المثلى المطلوبة للبرمجيات.
  - يستخدم خلال كل مراحل حياة البرمجيات بدءًا من التصميم، و ذلك للمحافظة على البرمجيات و جعلها واقعية و قابلة للفحص و سهلة القيادة.
- يعتمد على تدفق السيطرة في البرنامج، حيث أن تحديد عدد المناطق في الشكل يعد أحد المقاييس للعمليات المنطقية. وتعرف المنطقة بأنها المساحة المحصورة بين مستويات المخطط.

٤. مقاييس البرمجيات Software Metrics

مقاييس مكابي

- عدد المناطق يتناسب مع عدد القرارات و التكرارات في البرنامج، و بالتالي فإن هذا المقياس يعطي صورة عن الفحوص و مصداقية البرمجيات.
- يقترح مقياس مكابي علاقة بين عدد المهمات و صعوبة الفحوص، فكلما ازداد حجم المخطط عن عشرة عقد (رؤوس أو دوائر أو مهمات)، كلما زادت صعوبة تنفيذ الفحوص بصورة سليمة و فعالة.

٤. مقاييس البرمجيات Software Metrics

مقاييس النوعية

- من سمات التصميم الجيد للبرمجيات:
  - الازدواجية الضعيفة Low Coupling.
  - الترابط القوي High Cohesion.
- التصميم الجيد يخضع لاعتبارات عديدة منها:
  - معرفة تركيبة النظام بسهولة.
  - سهولة تعديل النظام ليلبي المتطلبات المستجدة.
- من المقاييس الأخرى، **قابلية تحوير المقطع البرمجي**، ليلائم تطبيقات عديدة، و لقياس التحوير، لابد من قياس الترابط القوي و الازدواجية الضعيفة، حيث أنهما سمات قوة تطويع المقطع أو تحويره.

٤. مقاييس البرمجيات Software Metrics

مقاييس النوعية

- تقاس **درجة تعقيد المقطع البرمجي Complexity Component** بعدة وسائل، منها:

$$C = a \times E + b \times N$$

  - E عدد أضلاع مخطط تدفق البيانات.
  - N عدد مرات الإشارة إلى البيانات من خارج المقطع البرمجي.
  - a,b هما ثوابت.
- من أساليب قياس **الازدواجية**: اعتبار المقطع البرمجي كأنه صندوق أسود في المخطط الهيكلي، و يحسب عدد الخطوط الواصلة إليه (In) و الخارجة منه (Out).
  - فالخطوط الواصلة (الداخلة) عبارة عن مقياس الازدواجية، أي كلما كان عدد الخطوط أكبر كلما كانت الازدواجية أعلى.
  - و الخطوط الخارجة مقياس درجة التعقيد، فكلما كان عدد الخطوط الخارجة أعلى كلما كانت درجة تعقيد المقطع أعلى.

٤. مقاييس البرمجيات Software Metrics

مقاييس النوعية

- يوجد صورة أخرى بدلاً من حساب عدد الخطوط، يتم حساب:

التدفق المحلي للبيانات + عدد تراكيب البيانات التي يقوم المقطع البرمجي بتحديثها

- يتضمن تدفق البيانات حساب عدد المتغيرات التي تجري عليها عمليات تحديث، و عدد البرامج الفرعية المستدعاة من داخل المقطع البرمجي، و على هذا الأساس تحسب **درجة تعقيد المقطع البرمجي** كالتالي:

$$C = L \times (In \times Out)^2$$

حيث L عدد اسطر المقطع البرمجي.

Questions or Comments?