استراتيجيات فحص البرمجيات /2/

Software Testing Strtegies

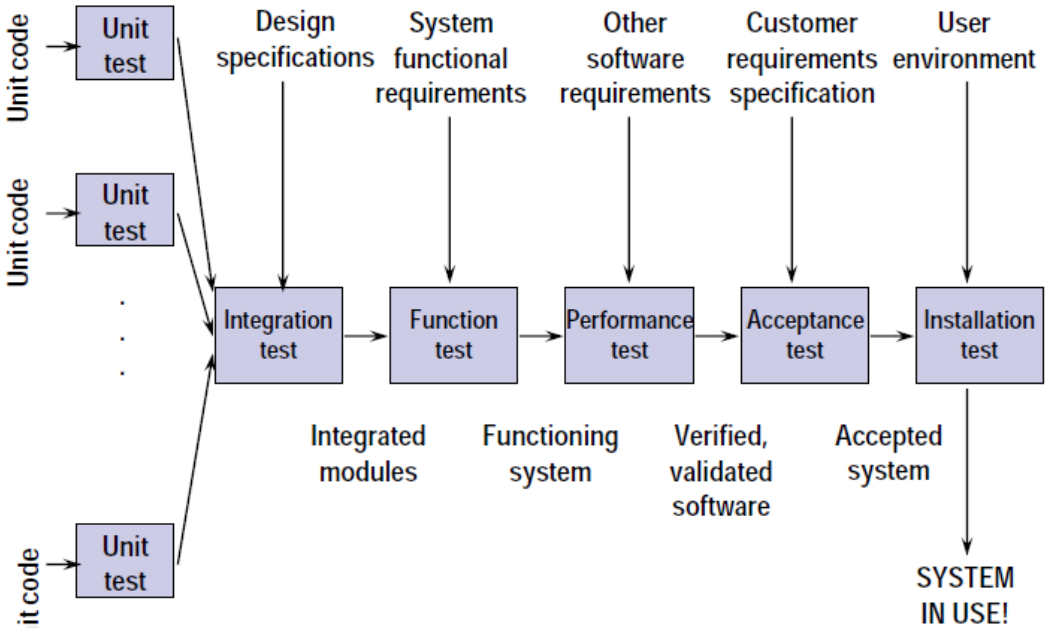
مقدمة:

غاية استراتيجية اختبار البرمجيات تكاملُ طرائق تصميم حالات اختبار برمجية في متتالية من المراحل المخططة جيداً، ينتج منها بناء ناجح للبرمجيات. وبنفس المستوى من الأهمية، تُعدُّ استراتيجية اختبار البرمجيات خريطة الطريق لكل من مطوّر البرمجيات، ومؤسسة ضمان الجودة، والزبون — وهي خريطة تصف المراحل المتبعة في الاختبار: متى تخطط تلك المراحل ومتى تنفّذ؟ وكم يلزم من الزمن والجهد والموارد؟ لذا يجب على استراتيجية اختبار البرمجيات، أيّاً كانت، أن تتضمن تخطيط الاختبار، وتصميم حالات الاختبار، وتنفيذ الاختبار، وتجميع المعطيات الناتجة وتقييمها.

لنتذكر معاً ما هو اختبار البرمجيات بعد الدروس السابقة، الاختبار هو تشغيل للبرنامج وتسجيل النتائج بهدف إيجاد الأخطاء والاختلافات مع المتطلبات. وقد يتم التشغيل تحت ظروف معينة لتقييم بعض جوانب النظام كالأداء والأمن.

تنظيم اختبار البرمجيات:

الاختبار هو مجموعة فعاليات يمكن أن تُخطط مقدماً وتُجرى بصورة نظامية. لهذا يجب تعريف *قالب template* لاختبار البرمجيات –وهو جملة مراحل يمكننا أن نضع فيها طرائق تصميم حالات اختبار محددة– في حالة إجرائية هندسة البرمجيات.



الشكل السابق يحدد التسلسل المقترح لاختبار النظام: حيث يتم اختبار كل جزء للنظام على حدة باستخدام اختبار الوحدة وبعدئذ نختبر بأن مختلف وحدات النظام تعمل مع بعضها البعض. ومن ثم يتم اختبار وظائف النظام مقارنة بالمتطلبات

الوظيفية ومن ثم يتم اختبار المتطلبات غير الوظيفية (الأداء، الأمن، .. حسب طبيعة النظام المطور) وفي النهاية يتم اختبار قبول النظام من المستخدم.

1 - اختبار الوحدة (Unit Test)

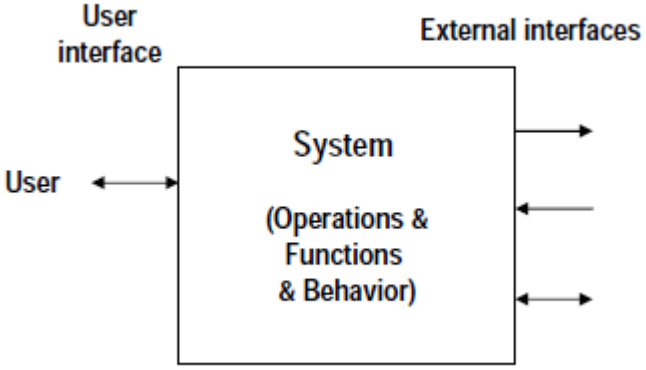
تم شرحه في الملف السابق.

2 - اختبار التكامل (Integration Test)

تم شرحه في الملف السابق.

3 - اختبار الوظائف (Function Test) أو Validation Test:

عند بلوغ اختبار التكامل أوجَه، وتكون البرمجيات كلها قد جمّعت بشكل حزمة، والأخطاء المتعلقة بالتواجه قد اكتشفت وصُححت، يمكن حينئذٍ إجراء سلسلة أخيرة من الاختبارات البرمجية – *اختبار إقرار الصلاحية validation testing*.



يقوم مهندس اختبار أو مهندسو الجودة باختبار البرمجيات المكاملة بالاعتمادعلى المتطلبات للتأكد من بناء المنتج الصحيح (إقرار صلاحية لوظائف البرمجية). حيث يتم التركيز على:

- دخل / خرج النظام.
- وظائف النظام.
- واجهات عمل النظام مع الأنظمة الأخرى.
- واجهات المستخدم.
- سلوك النظام العام.

4 - اختبار الأداء (Performance Test)

تعّد برمجياتُ نظم الزمن الحقيقي، والنظم المضمَّنة، التي توفر الوظائف المطلوبة ولكنّها لا تتطابق مع متطلبات الأداء، غيرَ مقبولة. يُصمَّم ***اختبار الأداء*** *performance testing* لاختبار أداء زمن تنفيذ البرمجيات من منظور نظام متكامل. يُجرى اختبار الأداء في كل مرحلة من إجرائية الاختبار. يمكن حتى على مستوى الوحدة تقدير أداء مجتزأ منفرداً خلال اختبارات الصندوق الأبيض. لكن، حتى نستطيع التحقّق من الأداء الحقيقي للنظام، يجب انتظار تكامل عناصر النظام كافة.

يقوم مهندس اختبار أو مهندس الجودة باختبار أداء النظام عند التشغيل للنظام المتكامل ضمن سياق إقرار صلاحية المتطلبات غير الوظيفية. ويتم اختبار:

- سرعة التشغيل للوظائف = الزمن اللازم لحساب نتيجة طلب (الأصغرية / الأعظمية / الوسطى)
- معدل الإنتاجية = عدد الطلبات المنفذة خلال وحدة الزمن (الأصغرية / الأعظمية / الوسطى)

- التأخير = المسافة الزمنية بين إرسال الطلب والحصول على النتيجة (الأصغرية / الأعظمية / الوسطى)
- استخدام الموارد = نسبة موارد النظام المستخدمة (الأصغرية / الأعظمية / الوسطى)
- توافر النظام (على مستوى المكونات / على مستوى كامل النظام)
- وثوقية النظام (على مستوى المكونات / على مستوى كامل النظام)
- قابلية التوسع (على مستوى المكونات / على مستوى كامل النظام)
- معدلات النجاح والفشل

غالباً ما تقترن اختبارات الأداء باختبار الإجهاد، وتحتاج إلى التجهيز بأدوات قياس برمجية وعتادية. فمن الضروري غالباً قياس نسبة استخدام الموارد (أي دورات المعالج) بدقة. يمكّن التجهيزُ بأدوات قياس من مراقبة أدوار التنفيذ، وتدوين الحوادث (أي المقاطعات) وقتَ حدوثها، وأخذ عيِّنات من حالات الآلة وفق قاعدة نظامية. بتجهيز النظام بأدوات قياس، يمكن أن يَكشفَ المختبرُ عن حالات تؤدي إلى تقهقر النظام وإخفاقه المحتمل.

اختبار الإجهاد (Stress Testing): حيث تقييم النظام خارج حدود المواصفات المطلوبة (حجم المطيات أو معدل ورودها كبير جداً) والهدف هو جعل النظام يفشل، مما يسمح باكتشاف نقاط الضعف في البرنامج.

يقوم اختبار الإجهاد باختبار النظام، بطريقة تتطلّب موارد بكميات، أو بتواتر أو بأحجام غير طبيعية. يمكن على سبيل المثال (1) تصميم اختبارات خاصة لتوليد 10 مقاطعات في الثانية إذا كان المعدل الوسطي هو مقاطعة أو مقاطعتين؛ (2) زيادة معدل إدخال المعطيات بمقدار كبير، لتحديد مدى استجابة وظائف الدخل؛ (3) تنفيذ حالات اختبار تتطلب حُلّ الذاكرة أو موارد أخرى ؛ (4) تصميم حالات اختبار تستطيع التسبّب لنظام تشغيل افتراضي بالتقلُّب thrashing؛ (5) إنشاء حالات اختبار يمكن أن تسبب بتفتيش مفرط عن المعطيات المقيمة على القرص. يسعى المختبِر أساساً لكسر البرنامج.

هناك نظير لاختبار الإجهاد هو تقنية تسمّى *اختبار الحساسية sensitivity testing*. قد يتسبب مجال صغير جداً من المعطيات، المحتواة داخل حدود المعطيات الصالحة من برنامج، بمعالجة شاذة أو خاطئة أو بتقهقر عميق للأداء في بعض الحالات (تحدث في الخوارزميات الرياضية عموماً). الحالة هذه تماثل الشذوذ في التوابع الرياضية. يسعى اختبار الحساسية لكشف تركيبات المعطيات في صفوف الدخل الصالحة، التي تتسبب في عدم الاستقرار أو في معالجة غير مناسبة.

5 - اختبار القبول (Acceptance Test)

من المستحيل عملياً على مطور البرمجيات أن يستقرأ كيفية استخدام الزبون للبرنامج. قد يفهم المستخدم تعليمات الاستخدام بشكل مغلوط. وقد يستخدم تركيبات غريبة من المعطيات. لذلك يقوم المستخدم باختبار النظام ككل وظيفياً ومن النواحي غير الوظيفية.

يُجري اختبارَ ألفا الزبونُ، في موقع المطور. وفيه تُستخدم البرمجيات بإعداداتها الطبيعية، ويقف المطور "فوق كتف" المستخدم، ينظر ويدوّن الأخطاء ومشاكل الاستخدام. تجرى اختبارات ألفا في بيئة مراقَبة.

يُجري اختبارات بتّا المستخدمُ النهائي (أو المستخدمون النهائيون) للبرمجيات، في موقع أو أكثر من مواقع الزبون. لا يكون المطور موجوداً عموماً، على عكس اختبار ألفا. لذا، فإن اختبار بتّا هو تطبيق "مباشر" للبرمجيات، في بيئة لا يمكن أن يتحكم المطوّر فيها. يقوم الزبون بتسجيل جميع المشاكل (الواقعية والتخيّلية) التي يمكن مصادفتها خلال اختبار بتّا، ويرفع تقارير عنها إلى المطوّر، على فَتَرات نظامية. ثم يقوم مطور البرمجيات، نتيجة للمشاكل المُعْلَم بها خلال اختبار بتّا، بإجراء التعديلات، والتحضير لإصدار المنتَج البرمجي إلى قاعدة الزُّبُن الكاملة.

ويتم التركيز على:

- وظائف النظام والأداء.
- اختبارات الوثوقية والاستعادة.
- سلوك النظام تحت شروط معينة (اختبارات الحساسية).
- تكامل البرمجيات والبنى المادية.
- التكامل مع البرمجيات الأخرى.

اختبار الاستعادة (Recovery Test):

ينبغي للعديد من النظم الحاسوبية أن تعاود العمل بعد حدوث عطل، وأن تتابع المعالجة ضمن مدة محددة مقدماً. في بعض الحالات يجب على النظام تحمل الأخطاء، أي يجب ألاّ تسبب معالجة الأعطال إيقاف وظيفة النظام الكلية. في حالات أخرى يجب إصلاح عطل النظام ضمن مهلة زمنية محددة، وإلا فقد تحدث أضرار اقتصادية قاسية.

اختبار الاستعادة *recovery testing* هو اختبار للنظام، يرغم البرمجيات على الإخفاق بأساليب عديدة، ويتحقق أن "الاستعادة" قد نفّذت بصورة لائقة. إذا كانت الاستعادة آلية (ينفّذها النظام نفسه)، فيجب تقييم إعادة الاستبداء، وآليات نقاط التدقيق، واستعادة المعطيات، وإعادة الإقلاع، للتَّحَقُّق من صحتها. وإذا تطلّبت الاستعادة تدخل عناصر بشرية، فإنه يجب تقدير الزمن الوسطي للإصلاح، لنحدِّد أهو ضمن الحدود المقبولة.

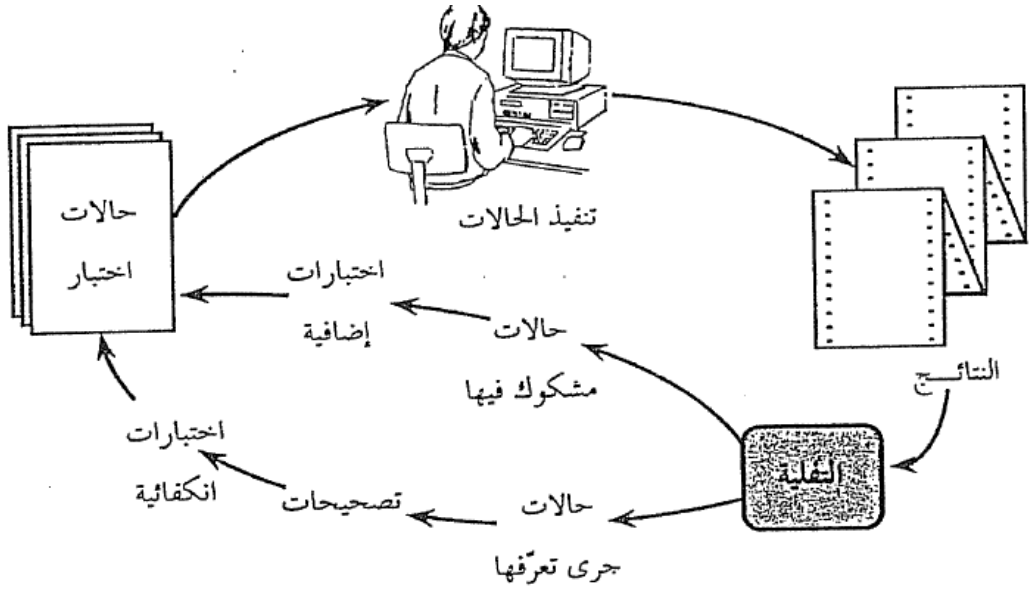
6 - اختبار النظام (System Test) أو يدعى Installation Test:

عادة ما تكون البرمجيات جزءاً من نظام أكبر يعتمد على الحاسب. يقوم مهندس الاختبار او مهندس الجودة باختبار تكامل النظام مع البنى المادية والبرمجيات الأخرى.

التفلية Debugging:

تحدث *التفلية debugging* نتيجةً لاختبار ناجح. أي، عندما تَكشِف حالةُ اختبار خطأً ما، فإن التفلية هي الإجرائية التي ينتج عنها إزالة الخطأ. ومع أن التفلية تستطيع، بل يجب، أن تكون إجرائية منظّمة، إلا أنها ما تزال فناً أكثر من أي شيء آخر. غالباً ما يواجه مهندسَ النظام، وهو يقيّم نتائج اختبار ما، مؤشرٌ "عرَضيٌّ" لمشكلة برمجية. أي يمكن أن يكون المظهَر الخارجي والسبب الداخلي للخطأ، غير مرتبطين، أحدهما بالآخر، بعلاقة واضحة. الإجرائية الفكرية التي تربِط العرَضَ بالسبب وتُدرك إدراكاً ضعيفاً، هي التفلية.

Debugging ليس اختباراً ولكنه ينتج عن الاختبار. وإنما بعد تنفيذ حالات الاختباروعند اكتشاف عدم التطابق بين النتائج الفعلية والنتائج المتوقعة، نبدأ Debuggingلاكتشاف سبب عدم التطابق.



ضبط الجودة Quality Control:

هي سلسلة من الاختبارات tests والمراجعات reviews خلال دورة تطوير البرمجيات للتأكد من أن المنتج يطابق المواصفات.

المراجعة Review:

مراحل المراجعة:

- عرض لوثائق (خطة المشروع / التصميم /الكود / ...).
- توثيق تعليقات فريق المراجعة.
- مناقشة التعليقات لتحديد الأفعال المناسبة (تصحيح / تعديل / ...).
- تقرير عن تقدم العمل في الوثيقة موضوع المراجعة.

العمل بعد المراجعة:

- متابعة التصحيحات والتعديلات والأداء واختبار الأجزاء المعدلة.
- إعداد التقارير.

مثال عن قائمة لمراجعة وثائق تصميم البرمجيات:

- *Are all significant functions shown in design?*
- *Are all significant attributes specified in design?*
- *Are all names related to purpose and type and are they unambiguous?*
- *Are all relationships between classes specified?*
- *Do all functions have the data necessary for the function to execute?*

مثال عن قائمة لمراجعة كود بلغة جافا:

- *Are any while or if conditions closed with semicolon ";" ?*
- *Are all variables declared ?*
- *Does every "{" have a matching "}"?*
- *Does every equality comparison have a double "="?*
- *..*