مرجع المحاضرة : كتاب هندسة البرمجيات – تأليف د.أحمد شعبان دسوقي – أ.د.السيد محمود الربيعي

محتويات الوحدة

الموضوع	الصفحة
المقدمة	**165**
تمهيد	**165**
أهداف الوحدة	167
1. طبيعة الأخطاء	170
2. التحقق والمصادقة الساكنة والديناميكية	**172**
3.اكتشاف وتصحيح الأخطاء	176
4. التخطيط للتحقق والمصادقة	177
5. فحوصات البرامج	**180**
6. التحليل الآلي الساكن	188
7. اختبار الصندوق الأسود	192
8. اختبار الصندوق الأبيض (التركيبي)	**194**
9. اختبار بيتا	196
10. اختبار اندماج النظام	197
1.10 الاختبار من أسفل إلى أعلى	**198**
2.10 الاختبار من أعلى لأسفل	201
11. تطوير البرمجيات بطريقة الغرفة النظيفة	**204**
الخلاصة	**210**
لمحة مسبقة عن الوحدة التالية	**213**
إجابات التدريبات	**214**
مسرد المصطلحات	**217**
المراجع	**221**

مقدمة

تمهيد

عزيزي الدارس،،

أهلاً بك في الوحدة الخامسة من مقرر "مقدمة في هندسة البرمجيات" وهي تتناول بيان الفرق بين التحقق من صحة البرمجية وبين المصادقة عليها ومعرفة طبيعة الأخطاء الأكثر شيوعاً وانتشاراً في البرامج.

في هذه الوحدة سنتناول:

* طبيعة الأخطاء وفيها نتناول أهم أنواع الأخطاء التي قد تظهر عند تشفير (تكويد) احد المكونات وهي عدم إعطاء قيم بدائية للبيانات و تكرار الحلقات عدد مرات غير صحيح و أخطاء في القيم الحدية.

* التحقق والمصادقة الساكنة والديناميكية وتشمل طريقتين: فحص البرنامج ، واختيار البرنامج.

* اكتشاف وتصحيح الأخطاء وهي العملية التي تحدد مكان الأخطاء حيث يتم تصحيحها.

* التخطيط للتحقق والمصادقة وهنا سنتعرف على نموذج الاختيار والتطوير وسنوضح فيه كيفية استخراج خطط الاختيار من مواصفات النظام وفق تصاميمه.

* فحوصات البرامج وهي مراجعات هدفهااكتشاف الأخطاء التي قد تكون موجودة في البرنامج ، وسنتعرف هنا أيضاً على فرق التفتيش والشروط المسبقة للتفتيش ،وعملية التفتيش ، وقوائم الفحص ، ومعدلات التفتيش.

* التحليل الآلي الساكن:هو أدوات برمجية تمر على جميع محتويات نص البرنامج المصور وتكشف الأخطاء التي فيه ، وسوف نتناول في هذا القسم أيضاً فحوصات التحليل الساكن ،ومراحل التحليل الساكن ،و استخدامات التحليل الساكن.

* اختبار الصندوق الأسود: ابتكار عينة من البيانات تنوب عن (تمثل) كل البيانات المحتملة.

قواعد اختيار بيانات الصندوق الأسود باستخدام تكافؤ التقسيم هي:

- اختبار بيانات تمثل الكل من كل جزء أو قسم.
- اختبار بيانات متاخمة (بالقرب من) للأقسام.

_ اختبار الصندوق الأبيض: يستغل اختبار الصندوق الأبيض معرفة كيف يعمل الإجراء (procedure) حيث يستخدم قوائم شفرة البرنامج.

- اختبار بيتا: يتم طرح احد الإصدارات التمهيدية من البرنامج للعميل والذي يكون على دراية بأن المنتج به عيوب.
- اختبار اندماج النظام: توجد ثلاثة اتجاهات لاختبار النظام هي:

* الاختبار الكبير.
* الاختبار الكبير المطور.
* الاختبار المتزايد.

غير أن هناك طريقتين للاختبار المتزايد هما: الاختبار من الأسفل إلى الأعلى (التصاعدي)، والاختبار من القمة إلى الأسفل (التنازلي).

- تطوير البرمجيات بطريقة الغرفة النظيفة:فلسفة تطوير برمجيات مبنية على تفادي الأخطاء في البرمجة باتباع عملية فحص صارمة.

عزيزي الدارس أرجو أن لا يغيب عن ذهنك عند قراءة مادة الوحدة محاولة الإجابة عن الأسئلة التقويمية ، والعودة إلى النص للتأكد من صحة الإجابة ، كما أرجو عدم الرجوع إلى إجابات التدريبات إلا بعد محاولة حلها أولاً.

نأمل أن تؤدي دراستك إلى هذه الوحدة إلى إثراء فهمك وتعرفك التحقق والمصادقة على صحة البرمجيات

أهداف الوحدة

الهدف من هذه الوحدة هو إعطاء الدارس، مدخلاً لأعمال التحقق والمصادقة على صحة البرمجيات مع تركيز خاص لأساليب التحقق الساكنة، وبنهاية دراسة هذه الوحدة يجب أن يكون الدارس قادراً على أن :

- يشرح الفرق بين التحقق من البرمجية والمصادقة عليها.
- يصف طبيعة الأخطاء الأكثر شيوعا وانتشارا في البرامج.
- يعدد أهم فحوصات البرامج وكيفية اكتشاف الأخطاء بها.
- يشرح لماذا يعتبر التحليل الساكن الآلي للبرامج أسلوباً مهماً وفعالاً للتحقق.
- ينشئ جدول اختبار الصندوق الأسود.
- ينشئ جدول اختبار الصندوق الأبيض.
- يصف اختبار بيتا وفوائده.
- يعدد السمات الأساسية لاختبار الاندماج بنوعيه من أعلى لأسفل ومن أسفل لأعلى.
- يشرح طريقة الغرفة النظيفة لتطوير البرامج ولماذا هي طريقة فعالة وما مميزاتها.
- يميز بين طرق تطوير البرمجيات عن طريق أساليب التحقق

توطئة

التحقق والمصادقة هو الاسم المطلق على عمليات الفحص والتحليل التي من شأنها التأكد من أن برامج الحاسب الآلي مطابقة لمواصفاتها وأنها تفي باحتياجات العملاء الذين يدفعون تكاليف هذه البرامج ، وعمليات التحقق والمصادقة هي عمليات مستمرة طوال عمر البرنامج كله , حيث تبدأ من مرحلة مراجعة المتطلبات وتستمر حتى مراجعات التصميم وفحوصات تحرير نص الشيفرة البرمجية إلى اختبار المنتج, ويجب أن يكون هناك أنشطة تصحيح وتصويب في كل مرحلة من هذه المراحل ، وتقوم هذه النشاطات على فحص نتائج كل مرحلة والتأكد من مطابقتها لما تم تحديده لها ، ويجب التنويه هنا إلى أن التحقق والمصادقة ليسا نفس الشيء رغم أنه قد يتم الخلط بينهما , وهو ما أعرب عنه بوهيم (1979م) حيث عرف كلاً من التحقق **(Verification)** والمصادقة **(Validation)** كمايلي :

❶التحقق : يعني إنتاج برمجية خالية من الأخطاء.

❷المصادقة : تعني التأكد من تلبية البرنامج لمتطلبات العملاء.

إن البرمجيات غاية في التعقيد ، والأسلوب السائد للتحقق من جودتها . في الوقت الراهن . هو الاختبار ، والذي يستهلك نسبة هائلة (تصل أحيانًا إلى 50%) من المجهود اللازم لتطوير النظام, فعلى سبيل المثال يوجد في شركة ميكروسوفت **(Microsoft)** العديد من المختصين بأمور الاختبار كما يوجد كذلك العديد من المختصين بعمليات البرمجة.

ومن المثير للجدل أن التثبت من سلامة البرنامج يشكل مشكلة كبيرة ونحن في حاجة لأسلوب جيد وفعال لحل تلك المشكلة ، وفي الغالب فإنه عند نهاية المشروع فإن أصعب القرارات الواجب اتخاذها هو القرار بين الاستمرار في عملية الاختبار أو

تسليم البرنامج إلى العملاء الراغبين فيه.

ويشمل دور التصديق التأكد من أن البرنامج مطابق للمواصفات التي وضعت له والتي تشمل (المتطلبات الوظيفية وغير الوظيفية المحددة له) , بل تذهب أبعد من ذلك إلى التأكد من مطابقة البرنامج لمواصفاته وحتى إلى إثبات أن البرنامج يقوم بعمل ما يتوقع منه العميل مغايرا لما تم تحديده.

والتصديق على متطلبات البرنامج في وقت مبكر ضروري جداً حيث إنه من السهل حدوث أخطاء ومحذوفات في متطلبات البرنامج وفي هذه الحالات فإن البرنامج النهائي لن يكون مطابقاً لما يتوقع العميل منه, وبالرغم من ذلك ففي الواقع لا يتوقع من عملية التصديق على المتطلبات اكتشاف كافة المشاكل الموجودة في المتطلبات, ففي بعض الأحيان تكتشف بعض الأخطاء والنواقص عند اكتمال التنفيذ فقط.

وسنبدأ هذه الوحدة بمناقشة المشكلة العامة للاختبار ، حيث نناقش فيها أعمال التحقق والمصادقة وكيفية التخطيط له ، ثم كيفية عمل فحوصات البرامج بفرعياتها المختلفة والتي تشمل (فرق الفحص ، وعملية الفحص ،و نموذج عملية الفحص ،و قوائم الفحص ، ومعدلات الفحص) ، ثم نعرج بعد ذلك على عدة اختبارات أهمها اختبار التحليل الساكن الآلي ، واختبار الصندوق الأسود ، واختبار الصندوق الأبيض ، واختبار بيتا.

إن مشكلة اختبار أجزاء كبيرة من البرمجيات والتي تتكون من عدة أجزاء متصلة تعد مشكلة صعبة خصوصًا لو أن الأجزاء متصلة معًا ومرتبطة لتكون وحدة واحدة. والبديل هنا هو اختبار التجزيء والذي من أمثلته التطوير من أعلى لأسفل حيث يفتح هذا الاختبار طريقًا أمام المستخدمين لرؤية إصدار نسخة مبكرة من النظام ، ولذلك فهو مفيد جدًا لاختبار الصلاحية.

إن تطوير البرمجيات عن طريق الغرف النظيفة (إعداد برمجيات خالية من الأخطاء بدلا من اكتشافها وتصحيحها) والتي سيتم التعرض له بالتفصيل في نهاية الوحدة تمثل طريقة مكلفة جدا وتحتاج إلى إمكانيات بشرية هائلة ، وكذلك يمكن اقتراح خطط اختبار عملية أخرى ، والتحقق الرسمي من سلامة برنامج ما باستخدام القوانين الرياضية الصارمة للوصول إلى الصحة والدقة والتي تستخدم لعدة أنظمة وبخاصة أنظمة السلامة منها, ولقد أظهرت الدراسات أنه من المفيد للذين يقومون بعملية الاختبار أن يكونوا أناساً آخرين غير المبرمجين ، فعلى سبيل المثال شركة ميكروسوفت توظف عديداً من فرق المبرمجين (الذين يكتبون البرامج) بالإضافة إلى فرق منفصلة تمامًا من المختبرين (الذين يختبرون هذه البرامج).

1. طبيعة الأخطاء (The Nature of Errors)

سوف يكون من الملائم معرفة كيفية ظهور الأخطاء لأننا فيما بعد سنحاول تجنب هذه الأخطاء أثناء كل مراحل التطوير، وبالمثل فإنه من المفيد معرفة أكثر الأخطاء شيوعًا وانتشارًا لأننا فيما بعد سنبحث عنها أثناء التحقق من جودة البرنامج وخلوه من الأخطاء ومن المؤسف أن البيانات غير الشاملة (التامة) هي فقط ممكنة لإعطاء جمل هلامية (غامضة) بخصوص تلك الأشياء.

وتُعد المواصفات مصدراً شائعاً للأخطاء ، فنظام البرنامج له مواصفات عامة مشتقة من تحليل المتطلبات ، بالإضافة إلى أن كل مكون من مكونات البرنامج له مواصفات فردية تخصه مشتقة من التصميم الهندسي ، ومواصفات أي من المكونات قد تكون غامضة (غير واضحة) ، أو غير كاملة (ناقصة) ، أو خاطئة, وأي من تلك المشكلات يجب بالطبع أن تُكتشف وتعالج عن طريق التحقق من جودة المواصفات وخلوها من العيوب قبل عملية تطوير المكون نفسه, لكن بالطبع

فإن هذا التحقق لا يمكن ولن يمكن أن يكون مؤثرًا (فعّالاً) بصورة كاملة ، ولذلك فهناك غالبًا مشاكل بخصوص مواصفات المكونات, هذا ليس كل شيء فهناك مشكلات أخرى بخصوص المواصفات ، أثناء البرمجة فإن مطور المكون قد يسيء فهم مواصفات المكون.

ويوجد نوعٌ آخر من الأخطاء يقع حينما يحوي المكون أخطاء ولا يخضع لمواصفاته وهذا قد يرجع إلى نوعين من المشكلات :

- أخطاء في منطق الشيفرة البرمجية.
- الشيفرة البرمجية لا تغطي كل جوانب المواصفات.

وهذا النوع من الأخطاء يكتشف حين يفشل المبرمج في تقدير وفهم كل تفاصيل المواصفات وبناء على ذلك يقوم بحذف بعضاً من الشيفرة البرمجية الضرورية.

وأخيرًا فإن من أهم أنواع الأخطاء التي قد تظهر عند تشفير (تكويد) أحد المكونات هي :

- عدم إعطاء قيم بدائية للبيانات.
- تكرار الحلقات عدد مرات غير صحيح.
- أخطاء في القيم الحدية.

القيم الحدّية هي قيم البيانات عند أو بالقرب من قيم محددة – فمثلاً لو فرض أن أحد المكونات يجب أن يقرر إذا كان على الشخص التصويت في الانتخابات أم لا استنادًا إلى عمره. فإذا كان سن التصويت هو 18 سنه فإن القيم الحدّية (المتاخمة) بالقرب من القيمة المحددة هي 17، 18، 19.

كما شاهدنا فإن هناك العديد من الأشياء التي يمكن أن تحمل أخطاء وربما لذلك فإنه ليس من المدهش (الغريب) أن التحقق من جودة البرنامج وخلوه من العيوب نشاط يستهلك وقتًا طويلاً.

2. التحقق والمصادقة الساكنة والديناميكية

(Static and Dynamic Verification and Validation) :

تشمل أعمال التحقق والمصادقة عموما طريقتين للفحص والتحليل ، وهما :

①فحص البرنامج **(Software Inspection) :**

ويشمل التحليل والتدقيق على مخططات البرنامج المختلفة مثل مستند المتطلبات، ومخططات التصميم ، وشيفرة البرنامج المصدر ، ويمكن أن تطبق هذه الفحوصات على كافة مراحل العملية, ويمكن أن تعزز هذه الفحوصات بنوع من أنواع التحليل الآلي للمستفيد الأصلي للنظام أو المستندات ذات العلاقة , ويجب ملاحظة أن الفحوصات والتحليل الآلي للبرنامج هي أساليب ساكنة حيث إنها لا تتطلب أن يتم تشغيل النظام.

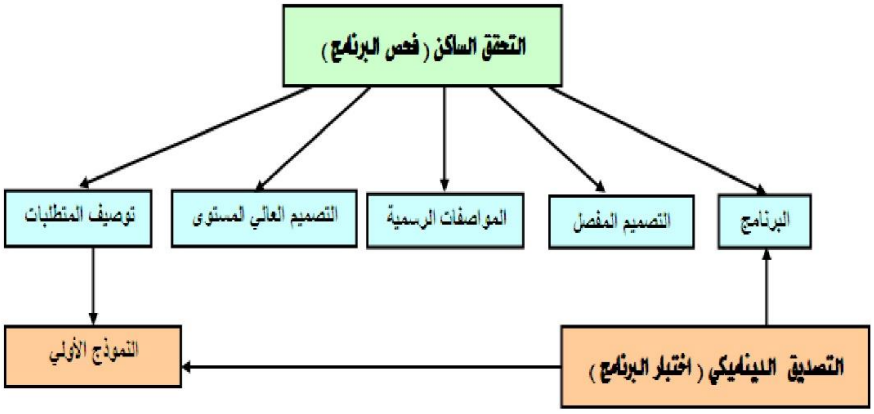
وتتكون تقنيات الفحص من معاينة البرنامج , والتحليل الآلي للشيفرة البرميجة الأصلية ، والتحقق المنهجي الرسمي ، على أن التقنيات الثابتة تتأكد فقط من وجود توازي بين البرنامج وبين مواصفاته ولا يمكنها التثبت من فائدة البرامج التشغيلية , كما أن هذه الأساليب لا يمكن لها أن تكشف الخصائص غير التشغيلية للبرنامج مثل الأداء والاعتمادية ، لذلك فأنه من الضروري القيام بنوع آخر من الاختبار للبرنامج ، وبالرغم من أن معاينات البرنامج الآن تتم بشكل واسع إلا أن تجربة البرامج ما زالت هي الأسلوب السائد للتحقق والتصديق ، وذلك لأن التجربة تنطوي على تشغيل البرنامج بمعطيات وبيانات مماثلة للبيانات التي سيقوم البرنامج بمعالجتها في الواقع ، حيث يتم استنباط وجود أية أخطاء أو مشاكل في البرنامج بمعاينة مخرجات البرنامج والتدقيق عن الأشياء غير العادية , ويمكن القيام بالاختبار خلال مرحلة التنفيذ للتحقق من أن البرنامج يعمل كما أراد له المصمم وذلك بعد الانتهاء من التنفيذ.

ويوضح الشكل 5.1 موضع فحص البرنامج واختباره ضمن عملية النظام , وتوضح الأسهم مراحل العملية التي قد تستخدم فيها هذه التقنيات.

ولذلك فإن فحوصات البرنامج يمكن أن تستخدم في كل مراحل تطوير البرنامج ولكن في المقابل من ذلك ، فإن الاختبار يمكن تطبيقه عندما يكون هناك نموذج أولي أو برنامج قابل للتنفيذ, وإن شاء الله سوف نتناول فحوصات البرامج بصورة مفصلة في الجزء 5.6 من هذه الوحدة.

②اختبار البرنامج **(Software Testing)** :

ويشمل تنفيذ تطبيق البرنامج عن طريق بيانات اختبارية ومعاينة مخرجات البرنامج وسلوكها التشغيلي للتأكد من أن البرنامج يفي بما هو مطلوب منه.



الشكل 5.1 : شكل توضيحي لمراحل التحقق والمصادقة الساكنة والديناميكية

ولذلك فإن فحوصات البرنامج يمكن أن تستخدم في كل مراحل تطوير البرنامج ولكن في المقابل من ذلك ، فإن الاختبار يمكن تطبيقه عندما يكون هناك نموذج أولي أو برنامج قابل للتنفيذ. وإن شاء الله سوف نتناول فحوصات البرامج بصورة مفصلة في القسم الخامس من هذه الوحدة.

③اختبار البرنامج **(Software Testing)** :

ويشمل تنفيذ تطبيق للبرنامج عن طريق بيانات اختبارية ومعاينة مخرجات البرنامج وسلوكها التشغيلي للتأكد من أن البرنامج يفي بما هو مطلوب منه.

وهناك اختبارات متعددة تستخدم في المراحل المختلفة لإعداد البرمجيةهي :

- اختبار الخلل : **(Defect Testing)** : لاكتشاف الاختلاف بين البرنامج وبين مواصفاته ، حيث يكون السبب في هذه الاختلافات . في أكثر الأحيان . لخلل في البرنامج نفسه فتكون التجارب مصممة لإظهار الخلل في النظام بدلاً من محاكاة استخدامه التشغيلي.
- الاختبار الإحصائي : **(Statistical Testing)** : ويستخدم لاختبار أداء البرنامج والاعتماد عليه ولكشف طريقة عمله تحت الظروف التشغيلية المختلفة, وهذه الاختبارات مصممة على أن تعكس مدخلات المستخدم العادي وتكرارها ، ويمكن بعد تشغيل البرنامج تقدير اعتمادية البرنامج ، وذلك باحتساب عدد مرات فشل النظام ، ويمكن الحكم على أداء البرنامج بقياس وقت التنفيذ ووقت استجابة البرنامج بينما يقوم بمعالجة البيانات التجريبية.

وبطبيعة الحال فإنه لا توجد حدود ثابتة وسريعة بين أساليب الاختبار هذه ، فخلال اختبار الأعطال يحصل المختبرون على استشعار الاعتمادية , ومن خلال الاختبار الإحصائي من المتوقع أن يتم اكتشاف بعض الأعطال, لأن الهدف النهائي من عملية التحقق والمصادقة هو العمل على تأسيس الثقة بأن البرنامج صالح للغرض الذي طور من أجله ، وهذا لا يعني أن البرنامج سيكون خالياً من الأخطاء ، ولكن معناه أن البرنامج جيد للاستخدام المرجو منه.

ويعتمد مستوى الثقة المطلوب للتحقق والمصادقة على كل من الأهداف المرجوة من البرنامج ، وتوقعات مستخدمي البرنامج ، والبيئة التسويقية الحالية لهذا البرنامج والتي يمكن توضيحها بالتفصيل كالتالي :

1) وظيفة البرنامج (Software Function) : يعتمد مستوى الاعتمادية المطلوب بأهمية البرنامج للمؤسسة فالاعتمادية المطلوبة لبرنامج للتحكم بمستوى نظام أمني هو أعلى بكثير من الاعتمادية المطلوبة لبرنامج تجريبي تم تطويره لعرض فكرة جديدة.

2) توقعات المستخدمين (User Expectations) : إنه لمؤسف في صناعة البرمجيات أن تتكون لدى الكثير من المستخدمين توقعات متدنية عن البرامج التي يتم تطويرها ولا يكونو مندهشين عندما تفشل هذه البرامج في التشغيل, ويكون المستخدمون مستعدين لقبول هذه الإخفاقات عندما تفوق فوائد استخدام البرامج إخفاقاته, وبالرغم من ذلك فقد تضاءل تحمل المستخدمين لفشل البرامج مما كان عليه في التسعينات, والآن فإن تسليم أنظمة غير موثوق بها أصبح أقل قبولاً, لذلك يجب على شركات تطوير البرمجيات استغراق وقت أطول في أعمال التحقق والمصادقة.

3) البيئة التسويقية (Marketing Environments) : عندما يتم تسويق برنامج ، يجب أن يأخذ البائعين في اعتبارهم البرامج المنافسة ، والأسعار التي يود المستخدم أن يدفعها مقابل هذا النظام ، والجدول المطلوب لتسليم هذا النظام. وعند ما يكون هناك منافسون قليلون في السوق لشركة ما فإن هذه الشركة قد تود إصدار برنامج غير مصحح بشكله النهائي لأنها تريد أن تكون أول الداخلين به إلى السوق, وفي الحالات التي يكون فيها المستخدمون غير مستعدين لدفع مبالغ كبيرة على البرنامج ، فإنهم سيكونون حينئذ مستعدين لقبول بعض الأخطاء في هذه البرامج , ويجب أن تؤخذ جميع هذه العوامل في الحسبان.

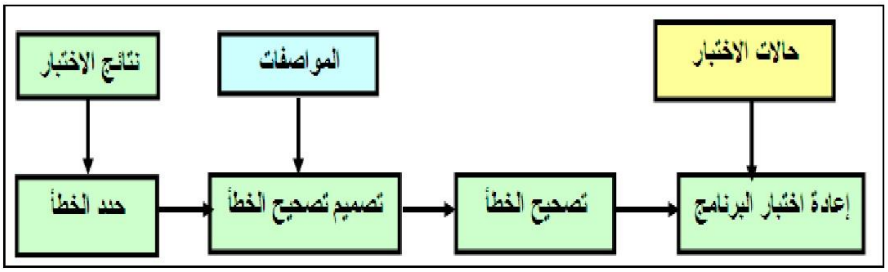
أسئلة تقويم ذاتي

1. صنف أهم الأخطاء شيوعا وانتشارا.
2. لماذا ليس من الضروري أن يكون البرنامج خاليا تماما من العيوب قبل أن يصل للعملاء؟

3. اكتشاف وتصحيح الأخطاء

(Testing and Debugging)

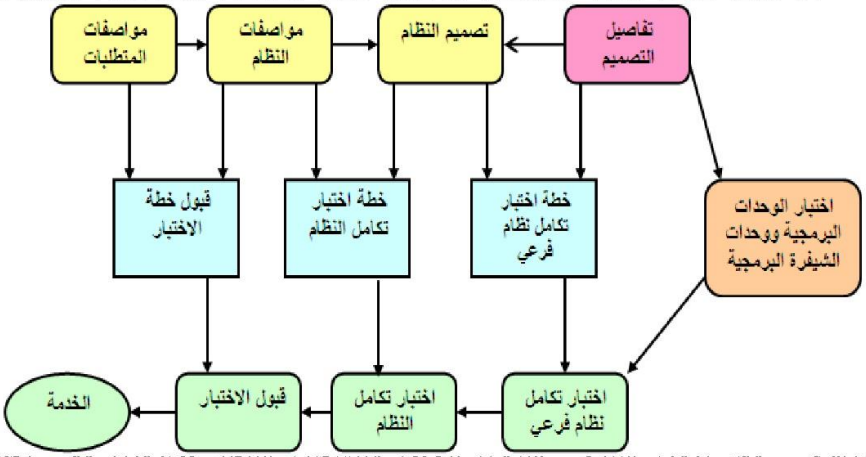
في العادة يتم اكتشاف الأخطاء الموجودة في برنامج ما حيث يتم تصحيح الأخطاء ويجب عندئذ تعديل البرنامج لتصحيح هذه الأخطاء , وعملية التصحيح تدمج دائما مع أعمال التحقق والمصادقة الأخرى إلا أن الاختبار أو بصفة عامة التحقق والمصادقة والتصحيح هي عمليات مختلفة لا يجب أن يتم دمجها ، حيث إن التحقق والمصادقة هي عملية اكتشاف وجود أخطاء في البرنامج ، أما عملية التصحيح ، كما هو واضح في شكل 5.2 ، فهي العملية التي تحدد مكان هذه الأخطاء حيث يتم تصحيحها.



شكل 5.2 : شكل توضيحي لعملية تصحيح الأخطاء

https://manara.edu.sy/

ولا توجد طريقة سهلة لتصحيح البرنامج حيث يتفحص المهندسون المصدر كلاً



شكل 5.3 : شكل توضيحي لنموذج "V" للاختبار والتطوير

طريقة تنفيذ البرنامج مفيدة.

4. التخطيط للتحقق والمصادقة

(Verification and Validation Planning) :

التحقق والمصادقة عملية مكلفة ، فبالنسبة لبعض الأنظمة الكبيرة مثل أنظمة الوقت المباشر **(Online-Systems)** التي لها أهداف غير وظيفية معقدة , فإن نصف تكاليف التطويرتقريباً تصرف على التخطيط بهدف الحصول على الفائدة القصوى من هذه الفحوصات والاختبارات وبهدف السيطرة على تكاليف عملية التحقق والمصادقة.

- **نموذج الاختبار والتطوير (V-Model Development) :**

يجب أن يبدأ التخطيط للتحقق والمصادقة في وقت مبكر من عملية تطوير البرنامج , ويوضح النموذج في الشكل 5.3 كيفية استخراج خطط الاختبار من مواصفات النظام وفق تصاميمه , ويدعى هذا النموذج في بعض الأحيان نموذج في (V) (أقلب شكل 5.3 على جانبية لرؤية الـV) .

https://manara.edu.sy/

ويوضح هذا الشكل أيضاً كيفية تقسيم أنشطة التحقق والمصادقة إلى عدد من المراحل بحيث تقاد كل مرحلة من الاختبارات التي حددت لها لاكتشاف مطابقة البرنامج لمواصفاته وتصميمه ، ويجب أن تشمل عملية التخطيط للتحقق والمصادقة التوازن بين كل من الأساليب الديناميكية والثابتة ، وكذلك أن توضح المعايير والإجراءات الخاصة بفحوصات واختبارات البرامج وإعداد الكشوفات التي بواسطتها يتم القيام بهذه الفحوصات وأن dتم تحديد خطة الاختبار , ويعتمد الجهد النسبي المخصص للفحوصات والاختبارات على نوع النظام الذي يتم تطويره والخبرة السابقة للمنشأة في هذا المجال فكلما كانت أهمية النظام كبيرة كلما كان ضرورياً تخصيص جهد إضافي أكبر للتقنيات الثابتة.

إن التخطيط للاختبار يتضمن وضع معايير لعملية الاختبار بدلاً من شرح اختبارات المنتج ، ذلك لأن خطط الاختبار ليست فقط مستندات إدارية لأنها مصممة لتستخدم من قبل مهندسي البرامج القائمين على تصميم وإجراء هذه الاختبارات , كما أن هذه الخطط تسمح للموظفين الفنيين بأن تتكون لديهم فكرة عن الصورة الكلية لاختبارات النظام مما يسمح بوضع عملهم في هذا السياق ، كما توفر هذه الاختبارات للقائمين على المشروع المعلومات التي تضمن توفر موارد الأجهزة والبرامج المناسبة لفريق الاختبار.

- **هيكل خطة اختبار البرمجيات (Structure of a Software Test Plan) :**

يجب أن تأخذ هذه الخطة في الاعتبار كميات كبيرة من الأعمال المضاعفة (حالات الطوارئ) بحيث يتم الإحاطة الكاملة بالتأخيرات الخاصة على التصاميم والتنفيذ ويتم تحويل الموظفين الذين تم تخصيصهم لعمليات الاختبار إلى أنشطة أخرى ، وقد قام الباحث هاتون (1986م) بشرح جيد لخطط الاختبارات وعلاقتها بخطط الجودة الأكثر عمومية, وكما هو الحال للخطط الأخرى فإن خطة الاختبار ليست مستندا ثابتاً ويجب أن تتم مراجعتها بانتظام ، ذلك لأن الاختبارات أنشطة معتمدة على استكمال التنفيذ فإذا كان جزء من البرنامج غير مكتملاً فإنه لا يمكن

القيـام بالاختبـار المتكامـل ، والجـدول رقـم (1) يوضـح العناصـر الأساسـية لخطـة اختبار برمجية.

جدول رقم (1) يبين بنية خطة اختبار البرمجية (Structure of a Software Test Plan)

الفرعية	وصفها
عملية الاختبار The Testing Process	وتشمل وصف للمراحل الرئيسية لعملية الاختبار.
تعقب المتطلبات Requirements Traceability	إن ما يهم المستخدمين هو أن يفي البرنامج بما هو مطلوب منه ويجب أن يتم التخطيط للاختبار بحيث يتم اختبار كافة المتطلبات بشكل مستقل.
البنود المختبرة Tested Items	يجب أن تحدد بنود العملية للبرمجية التي يجب أن تختبر.
جدولة الاختبارات Testing Schedule	جدول كلي للاختبارات وتخصيص الموارد لهذا الجدول ومن الواضح أن هذا مرتبط بالجدول الشامل لتطوير المشروع الأكثر عموماً.
إجراءات تدوين الاختبارات Test recording Procedures	أنه غير كافٍ أن يتم إجراء الاختبارات فقط ، يجب أن تدون نتائج الاختبارات بشكل منظم وبنحو يسمح بالتدقيق في عملية الاختبار للتأكد من أنه تم القيام بها بشكل سليم.
متطلبات الأجهزة والبرمجيات Hardware and Software Requirements	يجب أن يحدد في هذا الجزء البرمجيات اللازمة وعتاد الحاسب المطلوب.
القيود Constrains	يجب أن يتم في هذه الفرعية التوقع للقيود التي تؤثر على عملية الاختبار مثل قلة عدد الموظفين.

5. فحوصات البرامج (Program Inspections)

فحوصات البرامج هي مراجعات هدفها اكتشاف الأخطاء والتي قد تكون موجودة في البرنامج ، وقد تم تطوير فكرة القيام بفحص رسمي أول مرة في شركة (IBM) في السبعينات وجرى شرحها من قبل الباحث فاجان (1979م) حيث إنها الآن الطريقة المستخدمة بشكل واسع للتحقق ، ومنذ ذلك الحين تم تطوير العديد من البدائل للطريقة الأصلية المطورة من قبل فاجان , **وبرغم ذلك فإن كل هذه الطرق مبنية على الفكرة الأصلية لفاجان والمتمثلة في أن يقوم فريق مكون من أفراد بتخصصات مختلفة بالقيام بمراجعة الشيفرة البرمجية المصدر سطراً سطراً.**

تدريب (1)

ما هو الفرق الأساسي بين فحوصات البرنامج وبين الأنواع الأخرى من مراجعة الجودة.

- **فرق التفتيش (الفحص) (Inspection Teams) :**

إن عملية الفحص هي عملية ذات طابع رسمي ويقوم بها في العادة فريق مكون من أربعة أشخاص حيث يقوم أعضاء الفريق بتحليل الشيفرة البرمجية والكشف عن أية أخطاء فيه (إن وجدت) , وقد اقترح الباحث فاجان توزيع الأدوار كمايلي : كاتب (مبرمج الشيفرة) ، وقارئ ، وفاحص ، ومنسق ، وذلك على أعضاء الفريق بحيث يقوم القارئ بقراءة نص البرنامج بصوت عالٍ لأعضاء الفريق , ويقوم الفاحص بإجراء الاختبارات من منظور اختباري (يجد الأخطاء والنواقص والمتعارضات) ويقوم المنسق بالترتيب للعملية (يرأس الجلسات ويديرها ، ويدون الأخطاء المكتشفة) وبفعل تطور خبرة المنشآت في مجال الفحوصات فقد ظهرت اقتراحات أخرى لتشكيل الفريق , ففي مناقشة لإدخال عملية التطوير الناجحة لشركة HP قام الباحثان جراوي

وفان إبلاك (1994م) باقتراح ستة أدوار لأعضاء الفريق كما هو موضح في الجدول رقم (2) ، حيث يمكن أن يأخذ العضو الواحد أكثر من دور واحد في الفريق ، وكذلك حجم الفريق قد يكون مختلفاً من فحص إلى أخر.

الجدول رقم (2) : مكونات أعضاء فريق الفحص السداسي

الدور	الوصف
المؤلف أو المبرمج	المبرمج أو المصمم المسئول عن إنتاج البرنامج أو المستند ، مسئول عن تصحيح الأعطال المكتشفة خلال عملية الفحص.
الفاحص	يكتشف الأخطاء والمحذوفات وعدم الانسجام في البرامج والمستندات.
القارئ	يعيد صياغة الكود أو المستند في جلسة اجتماعات الفحص.
المسجل(الكاتب)	وهو الذي يسجل نتائج مقابلة الفحص.
رئيس الفريق أو المنسق	يدير العملية ويعمل على تسهيل الفحوصات، ويرسل تقارير عن نتائج العملية إلى كبير المنسقين.
كبير المنسقين	مسئول عن إجراء تحسينات وتحديث الكشوفات وتطوير المعايير.

وقد لاحظ جرادي وفان إبلاك إن هناك حاجة دائماً لدور القارئ وفي هذا الصدد قاما بتعديل الاقتراح الأصلي لفاجان والذي كان فيه جزء مهم من العملية متعلقاً بقراءة النص بصوت عالي, كما أن جيب وجراهام (1993م) لا يريان ضرورة لوجود قارئ ويقترحان أن يتم اختيار فريق الفحص بشكل يعكس اختلافات وجهات النظر (مثل مختبر مستخدم إدارة جودة ...الخ).

- الشروط المسبقة للتفتيش **(Inspection Pre-condition)** :

وقبل أن يبدأ الفحص للبرنامج فإنه من الضروري :

① أن يكون هناك مواصفات دقيقة.

https://manara.edu.sy/

②أن يكون أعضاء فريق الفحص على دراية بمعايير المؤسسة.

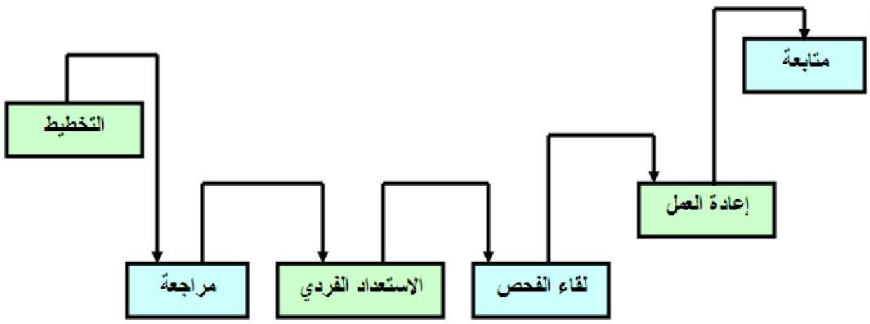
③أن يكون هناك نسخة صحيحة من حيث قواعد اللغة وحديثة من البرنامج الذي يراد فحصه.

④يجب أن تتوقع الإدارة أن عملية الفحص ستزيد من التكلفة في بداية عملية إعداد البرنامج.

⑤يجب أن لا تستخدم الإدارة عملية الفحص لتقويم العاملين.

- عملية التفتيش (الفحص) **(The Inspection Process)** :

وهناك نموذج عام لعملية الفحص موضح في الشكل 5.4 ، حيث يمكن الاقتباس وتكييف هذا النموذج وفق متطلبات المنشأة المعنية , ويكون المنسق مسئولاً عن التخطيط لعملية الفحص ، وهذا يتضمن اختيار أعضاء فريق الفحص والترتيب لغرفة الاجتماعات, وأن يتأكد من اكتمال المواد المراد فحصها والتأكد من اكتمال المواصفة الخاصة به , كما يتم تقديم البرنامج المراد فحصه إلى الفريق خلال مرحلة المراجعة ويقوم مؤلف البرنامج بوصف ما يجب أن يقوم به البرنامج وتلي هذه المرحلة مدى استعداد الأفراد بحيث يقوم كل عضو من أعضاء الفريق بدراسة الد



شكل 5.4 : شكل توضيحي لعملية الفحص

https://manara.edu.sy/

ويجب أن يكون الفحص ذاته قصيراً نسبيا (لا يزيد عن ساعتين) ويجب أن يكون فحصاً بصفة كاملة لاكتشاف الأخطاء الشواذ وعدم المطابقة للمواصفة والمعايير ، ويجب الاَّ يقوم فريق الفحص بتقديم أي اقتراح حول كيفية تصحيح هذه الأخطاء أو الفرضية على تغير العناصر.

وبعد إجراء الفحص يتم تعديل البرنامج وتصحيحه من قبل مؤلف ومن ثم يقرر المنسق ما إذا كان إجراء فحص آخر ضرورياً ، وفي المقابل فقد يقرر أنه لا توجد ضرورة لإجراء فحص كامل مرة أخرى وأَنَّ المشاكل قد تم تصحيحها ويقوم بعد ذلك بالموافقة على إصدار البرنامج.

ويجب أن يتم القيام بعمل كشف للأخطاء الشائعة في البرامج وذلك بالتنسيق والاستعانة بالموظفين ذوي الخبرة ، وأن يتم تحديث هذه الكشوفات كلما زادت الخبرة في عملية الفحص ، كما يجب طبع كشوف فحص مختلفة لكل نوع من لغات البرمجة , وتختلف هذه الكشوف من لغة برمجة إلى أخر؛ وذلك لأن كل لغة برمجة لديها مستوى مختلف من مستويات الكشف عن الأخطاء ، فمثلاً برنامج مكتوب بلغة آدا (Ada) يتم الكشف عن عدد العناصر كاملة بينما لا يقوم برنامج سي بذلك.

- **قوائم الفحص (Inspection Checklists) :**

ينصح جيلب وجراهام (1993م) أن تقوم كل منشأة بتطوير ووضع الكشوف الخاصة بها ويجب أن تكون هذه الكشوفات مبنية على المعايير والممارسات المحلية ولابد أن يتم تحديثها بشكل دوري كلما ظهرت أنواع أخرى من الأخطاء. والجدول رقم (3) يوضح المعاينات (الفحصوات المطلوبة) التي يمكن إجراؤها خلال مرحلة الفحص.

الجدول رقم (3) : يشمل قائمة الفحوصات التي يمكن إجراؤها خلال مراحل الفحص

فئة الخطأ (Fault Class)	الفحص المطلوب (Inspection Check)
أخطاء بيانات Data Faults	• هل كل متغيرات البرنامج تم وضع قيمة بدائية لها قبل استخدامها؟ • هل كل الثوابت تم تسميتها؟ • هل الحد العلوي للمصفوفة يساوي حجم المصفوفة أم هو أقل ؟ • هل هناك احتمال ظهور أرقام أعلى من الحد المسموح به؟
أخطاء تحكم Control Faults	• لكل جملة شرط هل الحالة صحيحة؟ • هل لكل مسار تأكيد لانتهائه؟ • هل عدد الأقواس في الجمل المركبة سليم؟
أخطاء دخل/ خرج Input/Output Faults	• هل كل المتغيرات الداخله مستخدمة؟ • هل كل متغيرات الخرج لها قيمة قبل إخراجها؟ • هل المدخلات غير متوقعة تسبب مشاكل؟
خطأ واجهة Interface Faults	• هل كل دالة وطريقة استدعاء لها العدد الصحيح من المتغيرات؟ • هل هناك توافق في أنواع المتغيرات؟ • هل المتغيرات في الترتيب الصحيح؟ • لو كانت المتغيرات متشاركة في الذاكرة فهل لها نفس موديل المشاركة في الذاكرة؟
أخطاء تخزين Storage Management Faults	• إذا حدث تغيير في تركيبة وصله هل تم إجراء تغيرات صحيحة لكل الوصلات؟ • لو تم استعمال ذاكرة ديناميكية هل تم تخصيصها بشكل صحيح؟ • هل يتم إلغاء تخصيص بوضوح بعد ألا يكون مطلوباً؟
أخطاء استثنائية إدارية Exception Management Faults	• هل كل احتمالات حالات الأخطاء أخذت في الاعتبار؟

وكلما اكتسبت المنشأة خبرة إضافية في عملية الفحص فإنه يمكن الاستفادة من نتائج هذه الفحوص كوسيلة للتحسين ، كما يمكن تحليل الأخطاء التي تم اكتشافها في هذه المرحلة وقد يكون بمقدور فريق الفحص ومطور البرنامج شرح أسباب حدوث هذه الأخطاء ، ويجب العمل على تعديل العملية بشكل لا تتكرر معه هذه الأخطاء كلما أمكن ذلك.

وقد لاحظ الباحثان جيلب وجراهام أن عدداً من المنشآت قد تخلت عن اختبار الوحدة لصالح الفحوصات حيث اكتشفت هذه المنشآت أن الفحوصات كانت فعالة لاكتشاف الأخطاء مما جعل التكاليف المتعلقة باختبار الوحدة غير مبرر ، وكما سيتم مناقشته في وقت لاحق من هذا الفصل فإن فحوصات البرامج قد استبدلت اختبار الوحدة في عملية تطوير برامج الحاسب الآلي باستخدام الغرف النظيفة.

- معدلات التفتيش **(Inspection Rates)** :

وقد لوحظ أن كمية الشيفرة البرمجية التي يمكن فحصها في زمن محدد يعتمد على خبرة فريق الفحص الذي يقوم بهذا العمل ، وعلى لغة البرمجة المستخدمة ، وعلى التطبيق الرئيسي ، وخلال قياس عملية الفحص لدى شركة **(IBM)** وجدت معدلات الفحص التالية :

- يمكن مراجعة 500 عبارة من عبارات نص الشيفرة البرمجية في الساعة خلال مرحلة المراجعة.
- يمكن خلال الإعداد الفردي تنفيذ حوالي 125 عبارة من نص الشيفرة البرمجية في الساعة.
- يمكن فحص حوالي 90 إلى 125 عبارة من نص الشيفرة في الساعة خلال الاجتماع.
- تكلفة فحص 500 سطر في الشيفرة البرمجية يعادل مجهود أربعين رجل – ساعة **(40 Man-Hours)**.

وقد تم التأكيد على هذه الأرقام بواسطة بيانات تم جمعها من شركة (AT&T) (برنارد وبرايس 1994م) حيث أظهر قياس أعمال الفحص نتائج مماثلة.

ويقترح الباحث فاجان أن يكون الوقت المستغرق في الفحص حوالي الساعتين ، وذلك لأن كفاءة اكتشاف الأخطاء تهبط بعد هذا الوقت , لذلك فمن المفروض أن تكون أعمال الفحص عمليات متكررة وتجري على مكونات صغيرة كل مرة خلال مرحلة تطوير البرنامج,وبقيام فريق فحص مكون من أربعة أشخاص وجد أن تكاليف فحص مائة سطر من الشيفرة البرمجية يعادل تقريباً شخصاً واحداً ليوم واحد , وهذا يعني أن الفحص ذاته يستغرق حوالي ساعة واحدة وأن كل فرد في الفريق يجب أن يأخذ حوالي ساعة إلى ساعتين للإعداد والتفحص , وفي المقابل فإن تكاليف اختبار البرامج متغايرة وتعتمد على عدد الأخطاء في البرنامج ، وبرغم ذلك كان الجهد المطلوب للفحص أقل من نصف الجهد الذي يستلزمه الاختبار.

ومن ناحية أخرى يجب أن يتأكد المديرون من أن هناك فاصلاً واضحاً بين أعمال الفحص وبين ممارسات التقويم الخاصة بالأفراد ، ولا يجب أن تستخدم نتائج الفحوصات التي تظهر أخطاء معياراً لتقويم المهندسين ، ويجب أن يكون القائمون على الفرق بشكل على توفير الدعم عند اكتشاف الأخطاء ولا يكون فيها أي معاتبة تتعلق بهذه الأخطاء.

6. التحليل الآلي الساكن (Automated Static Analysis)

إن محللات البرامج (Programs Analyzers) الساكنة هي أدوات برمجية تمر على جميع محتويات نص البرنامج المصدر وتكشف الأخطاء التي فيه والكائنات الشاذة به ولا تتطلب هذه المحللات تشغيل البرنامج ولكنها تتخلل نص البرمجية ومن ثم تحدد الأنواع الخاطئة في عباراتها, وتقوم هذه المحللات باكتشاف ما إذا كانت هذه العبارات قد أُسست بشكل سليم أنها تقوم بالاستنتاجات عن مسار التحكم في البرنامج ، وفي الكثير من الحالات تقوم بحساب كافة مجموعة القيم لبيانات البرنامج , كما تقوم بالإعداد لإمكانيات اكتشاف الأخطاء التي يقدمها مترجم اللغة.

- فحوصات التحليل الساكن (Static Analysis Checks) :

إن القصد من التحاليل الآلية الساكنة هو جذب انتباه الشخص القائم على التحقق للكائنات الشاذة في البرنامج مثل استخدام متغيرات لم يتم استهلالها , أو متغيرات لم يتم استخدامها , أو بيانات قد تفوق الحدود , وقد تم تبيان بعض الأخطاء التي يمكن اكتشافها بواسطة المحللات الساكنة في الجدول رقم (4). ومع أن هذه ليست بالضرورة حالات خطأ فإنه في الكثير من الأحيان تكون هذه الكائنات الشاذة هي نتيجة أخطاء برمجية. ويمكن القول إن التحاليل الآلية الساكنة تفيد لمساعدة عملية الفحص وتعد مساندا لها وليس بديلا عنها.

جدول رقم (4) : ويشمل فحوصات التحليل الآلي الساكن (Static Analysis Checks)

فئة الخطأ Fault Class	فحص التحليل الساكن Static Analysis Check
أخطاء البيانات Data Faults	• متغيرات مستخدمة قبل إجراء الاستهلال. • متغيرات تم الإعلان عنها مصرحة ولكن غير مستخدمة. • متغيرات تم الإعلان عنها مرتين ولكن لم تستخدم بين المهام. • تجاوزات لحدود المصفوفات. • متغيرات غير معلن عنها.
أخطاء في التحكم Control Faults	• كود لا يمكن الوصول إليه. • تفرع غير شرطي إلى حلقات.
أخطاء دخل / خرج Input/Output Faults	• أنتجت المتغيرات مرتين بدون تدخل مهمة
أخطاء في الواجهة البينية Interface Faults	• عدم توافق نوع المتغيرات. • عدم توافق عدد المتغيرات. • عدم استخدام الدوال. • وظائف أو إجراءات لا يتم استدعاؤها.
أخطاء إدارة الذاكرة Storage Management Faults	• مؤشرات غير مخصصة. • حسابات المؤشر.

- مراحل التحليل الساكن **(Stages of Static Analysis)** :

وتشمل مراحل التحليل الساكن :

① تحليل مسار التحكم **(Control Flow Analysis)** : يتم في هذه المرحلة تحديد الحلقات متعددة نقاط الدخول والخروج والشيفرة البرمجية التي لا يمكن الوصول إليها ، وهي شيفرة محاطة بعبارات تحويل **(GOTO)** أو جزء من عبارة يكون شرط التحكم فيه لا يمكن أن يكون صحيحاً.

② تحليل استخدام البيانات (Data Base Analysis): هذه المرحلة تلقي الضوء على كيفية استخدام المتغيرات في البرنامج حيث يدقق على المتغيرات التي لم يجرى لها استهلاك سابقا (قيم بدائية) , والمتغيرات المكتوبة مرتين دون أن يكون بينها تعيين ، والمتغيرات المصرح بها والمعلن عنها ولكنها لا تستخدم في البرنامج , كما أن تحليل استخدام البيانات يكتشف أيضاً الاختبارات غير الفعالة حيث تكون حالة الاختبار زائدة ، علماً بأن الحالات المكررة هي حالات لا تتغير فيها القيمة وهي دائماً أما صحيحة أو غير صحيحة.

③ تحليل الواجهة (Interface Analysis) : يكشف هذا التحليل توافق الروتين والإعلان عن المتغيرات فيه واستخداماتها , وهذا التحليل غير ضروري إذا ما استخدمت لغات قوية مثل لغة الجافا ، حيث يقوم مترجم هذه اللغات بهذا التحليل , ويمكن أن تكتشف تحاليل الواجهات الأخطاء المتعلقة في اللغات ضعيفة المستوى مثل لغة الفورتران ولغة سي , كما يمكن لهذا التحليل اكتشاف الدوال والبرامج الفرعية والإجراءات المصرحة ولا تستدعي من قبل البرنامج أو نتائج هذه الوظائف التي لا يتم استخدامها.

④ تحليل تدفق المعلومات (Information Flow Analysis) : يتم في هذه المرحلة تحديد العلاقة بين متغيرات الإدخال ومتغيرات الإخراج ، ورغم عدم الكشف عن الأشياء الشاذة إلا أن القيم المستخدمة في البرنامج توضح بشكل واضح في قوائم وبذلك يمكن اكتشاف الأخطاء خلال مراجعة الشيفرة البرمجية أو في مرحلة الفحوصات , ويمكن أن توضح تحاليل مسار المعلومات الشروط التي تؤثر على قيمة متغير معين.

⑤ تحليل المسار (Path Analysis) : هذه المرحلة الخاصة بتحليل المضمون تحدد كل المسارات المحتملة في البرنامج وتحدد العبارات المنفذة في كل مسار , حيث يوضح هذا التحليل نمط التحكم في البرنامج.

إن تحليل تدفق المعلومات وتحليل المسار ينتجان كمية كبيرة من المعلومات , على أن هذه المعلومات لا تلقى الضوء على الحالات الشاذة ولكنها تقدم البرنامج من وجهة نظر أخرى , وبسبب حجم المعلومات الكبير الذي ينتج فيه فإن هذه المراحل من التحليل الساكن ربما تستبعد من العملية حيث تستخدم فقط المراحل الأولى التي تكشف الحالات الشاذة بشكل مباشر.

- **استخدامات التحليل الساكن (Uses of Static Analysis) :**

المحللات الساكنة ذات فائدة كبيرة عندما تستخدم لغات مثل سي التي لا تتمتع بتنويع قوي والتدقيق الذي يمكن أن يقوم به مترجم لغة سي المحدود , لذلك فهناك احتمالات كبيرة لوجود أخطاء في البرنامج يمكن اكتشافها بواسطة إدارة التحليل آلياً ، وهذا ضروري جدا بصفة خاصة عندما تكون لغة مثل سي (و سي ++) مستخدمة في تطوير برامج حساسة جداً , وتحتوي أنظمة يونيكس ولينكس على محللات ساكنة تدعي(لينت) خاصة بالبرامج المكتوبة بلغة سي , وتقدم لينت تدقيقاً ساكناً موازياً يمكن أن يقدمه مترجم لغة قوية التنويع مثل لغة جافا.

ولا يمكن أن تُستبدل الأدوات الآلية بالمحللات الآلية الساكنة في عملية الفحص لان هناك بعض أنواع الأخطاء التي لا تكتشف بواسطة المحللات الساكنة ، فمثلاً يمكن لهذه المحللات أن يكشف عن المتغيرات التي لم يرسل أمر استهلاك ولكنها لا تكشف عن عدم صحة أمر الاستهلاك ، وفي اللغات ضعيفة التنويع مثل سي , تقوم المحللات الساكنة باكتشاف الدالات التي يوجد بها أرقام خاطئة وأنواع الأوامر بها خاطئة ايضاً ولكنها لا تكشف عن الأمر الخاطئ.

ولا يوجد شك في أن المحللات الساكنة للغات مثل سي هي أسلوب فعال لاكتشاف أخطاء البرنامج حيث تبقى بعض الأخطاء اللغوية غير معروفة بواسطة المترجم ، ولكن رغم ذلك فإن مصممي اللغة وللغات الحديثة مثل جافا قاموا

باستبعاد بعض الميزات التي كانت موضع الأخطاء في هذه اللغات حيث يتم إرسال أوامر الاستهلاك لكل المتغيرات ، ولا توجد عبارات (GOTO) مما يقلل من فرص القيام بالأخطاء , كما أن إدارة الذاكرة تتم بطريقة آلية , وهذا المنهج الخاص بتجنب الأخطاء بدلاً من اكتشافها هو الأسلوب الأكثر فعالية في تحسين اعتمادية البرنامج لذلك المحللات الساكنة المتممة قد لا تكون مناسبة لبرامج جافا.

7. اختبار الصندوق الأسود

(Black Box (Functional)Testing)

اختبار الصندوق الأسود هو ابتكار عينة من البيانات تنوب عن (تمثل) كل البيانات المحتملة (استخدام كل البيانات المحتملة في الاختبار يطلق عليه الاختبار المجهد أو الشاق) ثم نجري البرنامج وندخل البيانات ونرى ما سيحدث. واختبار الصندوق الأسود مزعوم لأنه لا معرفة له بأعمال البرنامج تستخدم كجزء من الاختبار. نحن فقط نفكر في المدخلات والمخرجات ، والبرنامج نفكر فيه على أنه غير مرئي داخل صندوق أسود.و اختبار الصندوق الأسود يسمى أيضًا الاختبار الوظيفي (اختبار المهام) لأنه يستخدم فقط المعرفة بمهام البرنامج.و كمثال فإننا سوف نفكر في إجراء اختبار ينفذ عملية الضرب باستخدام اختبار الصندوق الأسود. مواصفات الإجراء أنه يضرب قيمتين ويعيد منتج الرقمين.

نحن نحتاج إلى بعض البيانات التي تمثل الكل ،فقد تعتقد أن أي قيمة تماثل أي قيمة أخرى لذلك كل ما نحتاج عمله هو اختيار أي قيمتين لتمثيل كل القيم الممكنة وهكذا فإننا نختار مجموعة واحدة من بيانات الاختبار.

رقم الاختبار	البيانات	النتيجة المتوقعة
1	8 ، 12	96

على سبيل الحذر قد نشعر بالقلق لأن هذا محدود جدًا، مدركين أن الأرقام السالبة مختلفة نوعًا ما عن الأرقام الموجبة، فاختر ممثلاً عن الأعداد السالبة

ونستخدم كل الدمج.

رقم الاختبار	البيانات	النتيجة المتوقعة
1	12 ، 8	96
2	8 ، 7-	56-
3	20- ،12	240 -

مازال الاختبار غير كامل نسبيًا فقد نجد أن الرقم صفر له خصوصية خاصة ولذلك يجب اختبار البرنامج متضمنًا الرقم صفر.

رقم الاختبار	البيانات	النتيجة المتوقعة
1	12،8	96
2	7،8-	56-
3	12،20	240-
4	7-،0	0
5	7،0-	0
6	0،8	0
7	8،0	0
8	0،0	0

وهذا ما يكمل اختبار البرنامج طبقًا لمبدأ اختبار الصندوق الأسود.

لقد أوضحنا سبب حاجتنا إلى ثمانية مجموعات من بيانات الاختبار، وهذه المجموعات الثمانية معا عبارة عن المخرجات المتوقعة من الاختبار وتكون مواصفات الاختبار, هذا الاتجاه لابتكار بيانات اختبار الصندوق الأسود هو استخدام تكافؤ التقسيم؛ بمعنى النظر إلى طبيعة بيانات المدخلات لتحديد خواص مشتركة مثل تلك الخواص يسمى تقسيم أو تجزئة.

باختصار، إن قواعد اختيار بيانات اختبار الصندوق الأسود باستخدام تكافؤ التقسيم هي :

1) تقسيم كل قيم بيانات المدخلات.
2) اختيار بيانات تمثل الكل من كل جزء أو قسم (بيانات متكافئة) .
3) اختيار بيانات متاخمة (بالقرب من) الأقسام.

تدريب (2)

عزيزي الدارس، إذا كان لديك دالة مدخلاتها ثلاثة أرقام صحيحة (integers) ومخرجها الرقم الصحيح الأكبر, ارسم جدول الصندوق الأسود لاحتمالات المدخلات لهذه الدالة.

8. اختبار الصندوق الأبيض (التركيبي)

(The Box (Structural) Testing)

يستغل اختبار الصندوق الأبيض معرفة كيف يعمل الإجراء (Procedure) حيث يستخدم قوائم شيفرة البرنامج (سطور البرنامج المكتوب), ومعرفة كيف يعمل الإجراء أي بنيته (تركيبه) تستخدم كقاعدة لابتكار بيانات الاختبار وبصورة مثالية يكتب المختبر النتائج المتوقعة للاختبار ومواصفات الاختبار ثم يجري تشغيل الإجراء (Execute it) وإدخال البيانات ثم مقارنة النتائج (المخرجات) مع النتائج المتوقعة.

لو أننا حاولنا الاختبار بتنفيذ كل ممر خلال الإجراء سوف نواجه سريعًا نفس المشكلة التي واجهتها في محاولة اختبار البرنامج اختبارًا شاقًا (مُجهدًا) عن طريق اختبار الصندوق الأسود, لكن هناك العديد من الحلول ،و حيث إن الإجراء يحتوي على حلقات فإن عدد الممرات المحتملة يتفجر (يزداد). لكن الحل البديل لخفض عدد حالات الاختبار هو استخدام اختبار العبارة لاختبار إمكانية تصويت شخص من

عدمه كمثال والتي تأخذ الشكل :

(سن تصويت) الجمهور.

لو السن 18 عد إلى (« يمكنه التصويت ») .

وإلاّ عد إلى (« لا يمكنه التصويت » .

إن مبدأ اختبار عبارة الصندوق الأبيض هو أن كل عبارة في البرنامج يجب أن تنفذ في وقت ما أثناء الاختبار ، في هذا الإجراء عبارة « لو » يتم تنفيذها دائمًا أيًا كانت البيانات لكن هناك قيمتان مختلفتان للبيانات نحن في حاجة إليهما لتنفيذ النتيجتين المحتملتين للمقارنة وبالتالي تكون بيانات الاختبار المناسبة هي :

رقم الاختبار	البيانات	المخرجات/ النتائج
1	12	لا يمكنه التصويت
2	21	يمكنه التصويت

لاحظ أن اختبار الصندوق الأسود احتاج نفس نوع البيانات بالضبط لاختبار هذا الإجراء لكن تلك ليست الحالة بوجه عام, في اختبار الصندوق الأبيض نفس الجدال يُثار كما في اختبار الصندوق الأسود وهناك حالات خاصة تحتاج إلى فحص ومعاينة خاصة وبالتالي فإن اختبار الصندوق الأبيض يجب أيضًا أن يختار بيانات الاختبار التي :

- تكتشف القرارات التي تتحكم في اختيار الممرات .
- تهتم بأحداث الاختبار التي تؤخذ في حالات خاصة.

وكما هو الحال في اختبار الصندوق الأسود فهذا يسمى اختبار القيم المتاخمة (الحدّية) لاحظ أن اختبار العبارة سيجد الخطأ في إجراء المنتج المقدم مسبقًا في مثال عند مناقشة المشاكل العامة للاختبار.

تدريب (3)

الدالة التالية تعطي القيمة الكبرى لثلاثة أرقام صحيحة حدد جدول بيانات الصندوق الأبيض لهذه الدالة :

```
Int a,b,c
Int largest;
If (a>b)
      If(a>c)
            Largest =a;
Else
Largest c:
Else
If(b>c)
          Largest b;
           Else
Largest c:
```

9. اختبار بيتا (Beta Testing)

في اختبار بيتا يتم طرح أحد الإصدارات التمهيدية من البرنامج للعميل والذي يكون على دراية بأن المنتج به عيوب. ويُطلب من المستخدمين الإبلاغ عن العيبوب والاخطاء أو تسجيلها حتى يمكن تحسين المنتج قبل موعد طرحه في الأسواق ولقد اشتق الاسم « اختبار بيتا» من الحرف الثاني من الحروف اليونانية حيث يوصل الاسم فكرة أنه إجراء الاختبار الثاني والذي يتبع الاختبار داخل المنظمة التي قامت بالتطوير.

10. اختبار اندماج النظام

(System Integration Testing)

فيما سبق اعتبرنا أن التثبت من جودة الوحدة الواحدة أي إثبات جودة مكون واحد فقط من مكونات البرنامج أو طريقة واحدة أو شيء ما وهكذا. وافترضنا بصورة كلية أن هذا المكون عبارة عن جزء صغير جدًا. وهذه هي الخطوة (المرحلة) الأولى في التحقق من جودة أنظمة البرامج ، والتي بالطبع تتكون من عشرات أو مئات المكونات الفردية. ومهمة اختبار الأنظمة الكاملة تسمى اختبار النظام أو اختبار الاندماج.

إن تطوير البرامج عن طريق تنفيذ التصميم الهندسي يكون بعد وضع المواصفات المفصلة لكل الأجزاء فالمشروع يُستأنف بواسطة التصميم والشيفرة المكونات الفردية ، ولكن المشكلة التي تظهر الآن هي « كيف يمكننا اختبار هذه المكونات وكيف يمكننا اختبار النظام؟».

للإجابة على هذا السؤال حاول التمعن في الاتجاهات الثلاثة التالية لاختبار النظام.

- الاختبار الكبير (Big Bang) : أي وضع كل المكونات معًا بدون اختبار سابق ثم اختبار النظام بالكامل.
- **الاختبار الكبير المطور (Improved Big Bang)** : أي اختبار كل مكون من المكونات بصورة فردية ثم وضع كل المكونات معًا واختبار النظام بالكامل.
- **الاختبار المتزايد (Incremental)** : أي بناء النظام جزءًا جزءًا واختبار النظام الجزئي في كل مرحلة.

الاتجاه الأول أي الاختبار الكبير أو اختبار الأجزاء كوحدة واحدة يمثل كارثة حيث لا توجد طريقة سهلة لمعرفة أي الوحدات كانت سبب الخطأ ، وهناك أيضًا مهمة شاقة لاكتشاف الأخطاء. أما الطريقة الثانية فتعتبر أفضل نسبيًا لأنه غير

وضع المكونات معًا فإننا يصبح لدينا الثقة فيه كمكونات فردية. والآن فإن ظهور أي خطأ من المحتمل أن يكون سببه التداخل والتفاعل بين المكونات وهناك بالتحديد تظهر مشكلة كبيرة في اكتشاف الأخطاء.

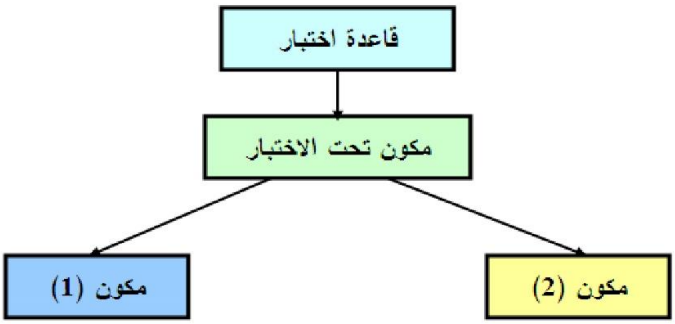
والبديل الأمثل هو استخدام أسلوب الاختبار المتزايد وفيه يتم أولاً اختبار مكون واحد من مكونات النظام ثم توصل وحدة ثانية مع الأولى ثم نختبر النظام ككل. وهكذا فإن إي خطأ يظهر فمن المحتمل أن يكون إما في الوحدة الجديدة أو في طريقة تداخل وتركيب الاثنين معًا. ثم نستمر في العمل بهذه الصورة بإضافة مكون واحد فقط في كل مره. وفي كل مرحلة أي خطأ سيظهر نفسه بسبب الوحدة الجديدة المضافة أو بسبب تركيبه مع النظام, وبهذه الطريقة فإن اكتشاف الأخطاء يصبح أسهل.

أن هناك طريقتين للاختبار المتزايد وهما الاختبار من الأسفل إلى الأعلى (التصاعدي) والاختبار من القمة إلى الأسفل (التنازلي) ، وفيما يلي مناقشة الطريقتين.

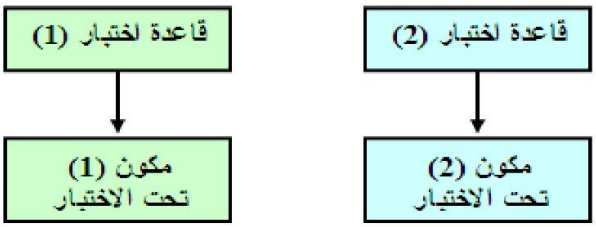
1.10 الاختبار من أسفل إلى أعلى

(Bottom-up Testing)

وهو أسلوب يبدأ بالمستوى الأدنى (الأسفل) للمكونات في النظام؛ وهي المكونات التي يستخدمها كل شيء آخر ولكنها في حد ذاتها لا تستخدم أي شيء, والمهمة الأولى هي إنشاء « حزام اختبار » أو « قاعدة اختبار » لكل مكون من المكونات, وهذا البرنامج المنشأ وظيفته الوحيدة هي تفعيل المكونات الخاضعة للاختبار (تحت الاختبار) بطريقة تتوافق مع دورها الحقيقي في النظام بالكامل, وبالتأكيد فإنه من المطلوب وجود بيانات اختبار خاصة. الشكل 5.5 يعرض مكونين في المستوى الأدنى أو الأسفل في النظام وأحزمة الاختبار الخاصة بهما.



شكل 5.5 : الاختبار من أسفل إلى أعلى كمرحلة وسطي



شكل 5.6 : الاختبار من أسفل إلى أعلى للمكونات الأدنى

ومما يجدر الإشارة إليه سريعًا هو أن أحزمة الاختبار في حد ذاتها قد تكون برنامجاً معقداً كبير الحجم ويتطلب تطويرًا ووقتًا للاختبار وكذلك فإن الإعداد لبيانات الاختبار قد تتطلب مجهودًا كبيرًا. وهاتان الملاحظتان تمثلان سقفًا كبيرًا للمشروع رغم أنهما لا يمثلان جزءً من النظام الكامل.

حينما يتم اختبار المكونات الأدنى من النظام بهذه الطريقة فإن الأجزاء تتحد لتكوين نظام فرعي يتم اختباره بطريقة مماثلة ، مرة أخرى باستخدام أحزمة الاختبار ، كما هو موضح بالشكل 5.6 ، ويستمر الإجراء حتى يتم اختبار النظام الكامل ككل في نهاية الأمر.

وطريقة الاختبار من أسفل إلى أعلى تعاني من العيوب الآتية :

①الوقت طويل نسبيًا في إنشاء أحزمة الاختبار وبيانات الاختبار ،والأسوأ من ذلك أن كلا منهما يتخلص منه عند إتمام عملية الاختبار, وهذا يتم تشبيهه بعمل النجار الذي يصنع لنفسه مجموعة من أطقم الأدوات والمعدات الخاصة لبناء منزل ما ثم يحطم هذه المعدات والأدوات بعد انتهاء بناء المنزل (المقصود من التشبيه هو المجهود الضائع). بدلاً على ذلك فإن قواعد الاختبار وبيانات الاختبار يتم الإبقاء عليها والاحتفاظ بها لاستخدامها عند الحاجة إلى إضافة التحسينات أو التعديلات على النظام للتأكد من أن المكونات التي كانت تعمل في السابق مازالت تعمل وفي هذه الحالة فإن مادة الاختبار يجب أن تخزن وتتم صيانتها دوريًا وهو ما يتطلب لجهدًا كبيرًا أيضًا.

② الأخطاء التي تكتشف في مراحل الاندماج لاختبار النظم الفرعية تتطلب التكرار للعملية بالكامل من تصميم ووضع شيفرة واختبار الوحدة. والأنظمة الفرعية أو المكونات قد يكون من الواجب إعادة عملها مرة أخرى كلما تم اندماج النظام ولا عجب في أن اختبار النظام يستغرق النسبة الأكبر (ما بين الثلث والنصف) للمعدل الزمني للمشروع. ولكن السؤال هنا لو أنَّ كل الأجزاء الفردية تعمل بصورة صحيحة فلماذا لا تعمل جميعها عند اتحادها ؟. الإجابة هي بالطبع تحديدًا في التداخلات التي تكشفها الأخطاء وهناك موقف أكثر تعقيدًا حيث تعالج الأجزاء البيانات المشتركة أو تراكيب البيانات.

③ اختبار النظام بالكامل يتم بالقرب من نهاية المشروع ، لذلك فإنه غالب الأخطاء الكبيرة في تصميم النظام لا تكتشف حتى اكتمال الوقت المحدد، وهذه الاكتشاف قد يؤدي إلى إعادة بناء الأجزاء الكبيرة للنظام في وقت حرج جدًا .

④ لا يوجد نظام عمل مرئي حتى اكتمال المرحلة الأخيرة وهي اختبار النظام,

ومن الحقيقي أن هناك مكونات ونظم فرعية تم اختيارها ولكن لا يوجد شيء يمكن إيضاحه وعرضه للعميل باعتباره رؤية محددة لما قد يقوم به النظام فعليًا.

كل هذه المشاكل يمكن أن تزيد المشكلات التي يكافح فريق العمل لوضع نهاية لها, ولا عجب أن الساعات الطويلة من العمل المتواصل تؤدي إلى الأخطاء التي تظهر كما لو أن الظروف تتآمر ضد المشروع ، لكن الاختبار من القمة للأسفل يساعد على تفادي بعض هذه المشكلات كما سنشرح الآن.

2.10 الاختبار من أعلى لأسفل

(Top-down Development)

تُعد طريقة قيمة لتطوير البرمجيات حيث تعمل وفقا للقواعد التالية :

① يبدأ التصميم للمكونات العليا للبرنامج أو النظام ، والتي تمثل في الحقيقة واجهة الاستخدام للنظام.

② يتم تشفير (تكويد) المكونات العليا للنظام.

③ يتم توليد بيانات الاختبار للمكونات العليا.

④ يستمر التنفيذ عن طريق اختبار مكونات ذات مستوى أدنى (كانت في البداية بقايا جذرية) لتكوين وتشفير وتجسيد النظام . بصورة عامة فإنه في أي مرحلة من مراحل التطور يوجد :

- مكونات الجزئية التي تحت الاختبار.
- البقايا الجذرية.

وعملية التطور هذه لن تستمر لأن هناك صعوبات قد تواجهها مثل :

أولاً : لن يتم التطور على أساس صلب (مستوى ثم مستوى) فإن بعض المكونات ذات المستوى الأدنى تحتاج إلى تصميم ثم تشفير واختبار في مراحل متقدمة ، ومثال

https://manara.edu.sy/

لذلك تطوير أجزاء البرنامج الذي يتفاعل مع المستخدم, حيث يظهر تصميم وشكل الشاشة للمستخدم الأخير للبرنامج الذي ربما يطلب التعديل.

ثانياً : يفضل بعض المبرمجين طرقاً مختلفة في التعامل فهم يخافون من ترجمة بعض التصاميم إلى رموز يتم اختبارها حيث يجدون نتيجة لتطوير مكونات ذات مستوى أدنى فإن هناك خطأ ما في المكونات ذات المستوى الأعلى ، وهذا يتطلب إعادة التصميم وإعادة التشفير والاختبار, والطريقة البديلة التي تبقى العديد من الفوائد هي إكمال التصميم لكل نظام أو برنامج قبل البدء في التشفير والاختبار من أعلى لأسفل.

أخيراً :هناك رأي آخر معارض للحد من استخدم هذه الوسيلة فهو من السهل للمبرمج أن يختبر المكونات كلاً على حدة بدلاً منها باعتبارها جزءً من النظام ذى المستوى الأعلى. وبالطبع في بعض الأحيان يكون من الصعب أن نختبر من أعلى لأسفل لعدم توفر معلومات الاختبار ، وفي مثل هذه الحالة يفضل تجسيد المكون الجديد في وقت مناسب لاختبار حدودها البينية في المكان الجديد.

هناك بعض الملاحظات في التنفيذ من أعلى لأسفل وهي :

① الاكتشاف المبكر للعيوب الكبيرة : حيث إنه من أهم المشاكل التي تواجه التطوير بأكمله مما يتطلب إعادة تشكيل الأجزاء الكبيرة من النظام, بجانب أن التطوير من أعلى لأسفل يضمن أن المكونات الكبيرة يتم اختبرها أولاً حيث يمكن تصحيح الأخطاء دون إعادة الكتابة,و بذا فإن هذه الطريقة تساعد في تقليل الجهد المبذول وإنجاز العمل في وقته, ويعتقد البض أن الإجراءات ذات المستوى الأعلى من البرنامج هي المهمة بينما يظن البض أن ذات المستوى الأدنى يسهل تجاهلها رغم أنها الأهم.

② المصداقية والاعتمادية : عندالاختبار من أعلى لأسفل فإن مكونات برنامج الحاسب الآلي يتم اختبارها عدة مرات مما يسهل اختبار الأجزاء ذات المستوى الأدنى ويسهل اكتشاف الأخطاء.

③ إزالة الأخطاء : بهذه الطريقة يسهل معرفة موقع الأخطاء, ولأن الجذور يتم استبدالها كل واحدة على حده بمكون جديد فإن المشكلة قد تكون في أحد هذه المكونات الجديدة أو في سطحها المشترك مع آخرى ذات مستوى أعلى وعلى العكس في حالة من أسفل لأعلى فإن العيوب تكون في المكون الجديد أو في الأسطح الجديدة (وهذا النوع من الاختبار يسمى الاختبار التزايدي).

④ الجانب المعنوي : في هذه الطريقة يكون العمل محسوساً وفي مرحلة مبكرة يتم التطوير مما يطمئن المديرون والعملاء.

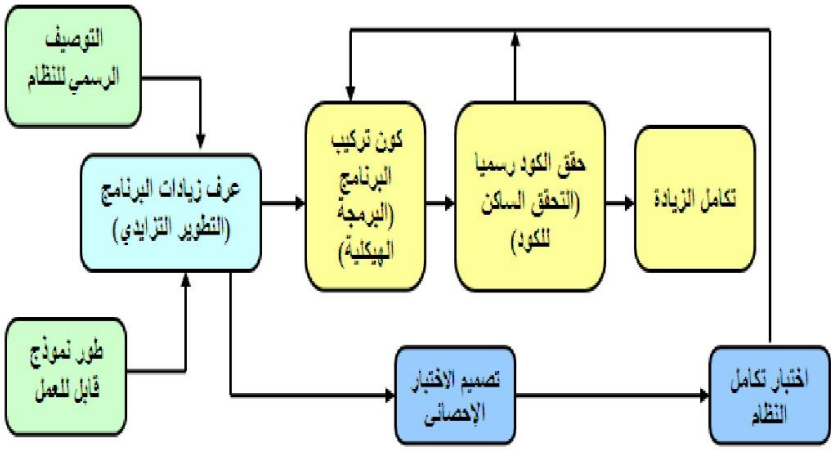
⑤ الطباعة المبكرة : في هذا النوع يمكن اعتبار نص التنفيذ من أعلى لأسفل الأولى للبرنامج كطباعة أولية للمستخدم بحيث يستطيع المستخدم التعليق على البرنامج ويقترح التغييرات قبل العمل.

⑥ زمن البرنامج : يكون في هذا النوع من أعلى لأسفل الزمن المستغرق أقل نسبة لسببين ورد ذكرهما ، إضافة إلى بناء أساس اختباري ومعلومات اختبارية . كما أن الأساس الاختباري لا يحتاج إلى تشييد حيث يكون قد تم تشييده عن طريق الجزء ذي المستوى العلوي في البرنامج الذي تم اختباره أما البقايا الجذرية فتحتاج لذلك وتكون بسهولة تتناسب مع هذه الوسيلة خاصة المشاريع التي يعمل فيها عدة مبرمجين . يبدأ تطوير هذه الطريقة بالتنفيذ الكامل للمستويات العليا للبرنامج وهذا عادة يتم بواسطة مهندس برمجة واحد ، أما بقية المبرمجين في هذه المرحلة يمكن أن يخصص لكل واحد منهم تطوير المستويات السفلى المتوقعة مع تحديد العديد من المشاكل التي تحدث في التطوير من أسفل لأعلى. لذا يفضل تأخير هذا التوزيع حتى إكمال إجراءات المستويات العليا للبرنامج.

11. تطوير البرمجيات بطريقة الغرفة النظيفة

(Clean Room Software Development)

تطوير البرمجيات بطريقة الغرفة النظيفة (ميلز وآخرون 1987م ، لينجر 1994م ، بووال وآخرون 1999 م) هي فلسفة تطوير برمجيات مبنية على تفادي الأخطاء في البرمجية باتباع عملية فحص صارمة ، الهدف منها تطوير برمجية خالية من الأخطاء. إن تسمية " الغرفة النظيفة" مشتقة من التشبيه بوحدات تصنيع أشباه الموصلات حيث يتم في هذه الوحدات (الغرفة النظيفة) تجنب وجود أخطاء بالتصنيع في بيئة فائقة النظافة ، وتعتمد فلسفتها على تفادي الخلل أثناء التطوير بدلا من استبعاده ، وسنقوم بمناقشة هذا الموضوع في الوحدة لأن هذه الطريقة استبدلت اختبار الوحدة لأجزاء النظام بواسطة الفحوصات لمعاينة انسجام هذه الأجزاء مع مواصفاتها, ويظهر في الشكل 5.7 نموذج لعملية تطوير البرامج بأساليب الغرف النظيفة المشتقة من الوصف الذي قام به الباحث لينجر (1994م).



شكل 5.7 : عملية تطوير البرامج بأساليب الغرف النظيفة

- **خصائص عمليات الغرف النظيفة.**

(Clean-room Process Characteristics) :

إن طريقة الغرفة النظيفة مصممة لدعم الفحص الصارم للبرنامج ، حيث يتم إنتاج نموذج نظام مبني على الحالة يستخدم كمواصلة للنظام وتتم تنقيته بواسطة عدد من النماذج ليصبح برنامجاً قابلاً للتشغيل , كما أن المنهج المستخدم في التطوير يعتمد تحولات مُعرَفة بشكل جيد يحاول فيها الاحتفاظ بالوضع الصحيح عند كل تحول إلى التمثيل الأكثر عموماً وتدعم فحوصات البرنامج بعدد من المعادلات الحسابية الصارمة توضح أن النتائج من البرنامج المعدل منسجم في مدخلاته.

إن المعدلات الحسابية المستخدمة في عمليات الغرف النظيفة أقل متانة من البراهين الرياضية المنهجية , ذلك أن البراهين الرياضية المنهجية مرتفعة التكاليف في تطويرها لأنها تعتمد على معرفة القواعد المنهجية للغة البرمجة لكي يتم تطوير نظريات تطابق البرنامج ومواصفته المنهجية ، ومن ثم يجب أن تبرهن هذه النظريات رياضياً باستخدام برامج كبيرة ومعقدة , وبسبب تكلفتها العالية والمهارات المتخصصة المطلوبة , فإن البراهين تطور دائماً للتطبيقات الحساسة الخاصة بالسلامة أو الحساسة أمنياً.

وبصفة عامة يعتمد منهج تطوير البرمجيات عن طريق الغرف النظيفة على خمسة خصائص هامة يمكن تلخيصها كمايلي :

① التوصيف الرسمي للبرمجية **(Formal Specification)** : توصف البرمجية التي يتم تطويرها بصفة رسمية (منهجية) ويستخدم نموذج انتقال الحالة الذي يظهر الاستجابة للمؤثرات للتعبير عن المواصفة.

② البرمجة الهيكلية **(Structure Programming)** : يستخدم فقط عدد محدد من مركبات التحكم والبيانات التجريدية حيث إن عملية تطوير البرنامج هي عملية

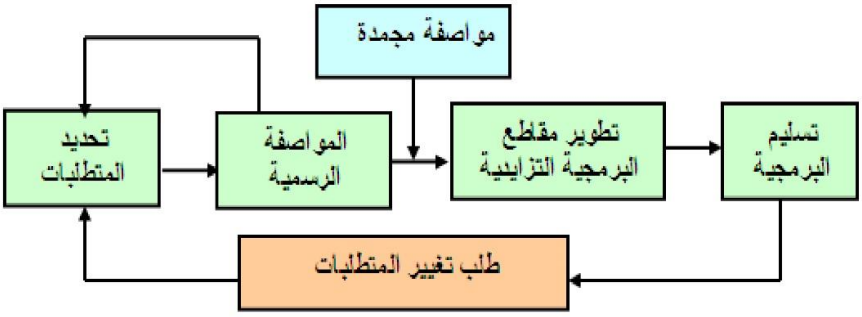
تنقية تصاعدية للمواصفة ويتم استخدام عدد محدد من المركبات , الهدف منها تطبيق تحولات تحفظ سلامة المواصفة ، وبالتالي لترميز البرنامج.

③ التحقق الساكن (Static Verification) : يتم التحقق الساكن للبرمجة المطورة باتباع فحوصات صارمة للبرمجية ، ولا يوجد هنا اختبارات للوحدة أو الموديلات لمكونات الشيفرة.

④ الاختبار الإحصائي للنظام (Statistical Testing) : يتم اختبار البرمجية الإضافية المدمجة إحصائيا , لتحديد اعتماديته ، وهذه التحاليل الإحصائية مبنية على منظور تشغيلي يتم تطويره بالتوازي مع مواصفة النظام.

⑤ التطوير التزايدي (Incremental Development) : وفيها يتم تجزي البرمجية التي يتم تطويرها إلى جزئيات يحقق كل جزء منها على حدة ويصادق عليه باستخدام طريقة الغرفة النظيفة ويتم عمل هذه التقسيمات مع تضمين مدخلات العميل في وقت مبكر من العملية.

إن التطوير التزايدي , الموضح في الشكل 5.8 ، يختص بإنتاج وتسليم برمجية علي مقاطع تزايدية منفصلة , ويمكن أن يتم تشغيل المقطع بأوامر يقوم بإصدارها المستخدم ويكون المقطع نظاماً قائماً بذاته ومحدداً ، ويقوم المستخدمون بتقديم آرائهم عن النظام وباقتراح التغييرات المطلوبة. إن تطوير المقاطع التزايدية مهم جداً لأنها تقلل من توقف التشغيل لعملية التطوير الناجمة عن التغييرات التي يطلبها المستخدمون للمتطلبات , فعندما تتم المواصفة في وحدة واحدة توقف التغييرات الخاصة بالمتطلبات التي يطلبها المستخدمون (وهو أمر لابد منه) للمواصفة وعملية التطوير بكاملها , وبالمقابل , فمع التطوير المقطعي فإن المواصفة للمقطع تكون مجمدة بينما يتم قبول طلبات التغير الخاصة بباقي النظام ويتم تسليم مقطع البرمجية عند الاستكمال.



شكل 5.8 : شكل توضيح لعملية التطوير التزايدي

إن منهج التطوير المقطعي في عملية الغرفة النظيفة هو لتسليم الأجزاء الحساسة في مقاطع يتم تسليمها في وقت مبكر, وتضاف الوظائف الأقل حساسية الى المقاطع اللاحقة ، وبذلك تتوفر للعميل فرصة تجربة هذه المقاطع الحساسة قبل أن يتم تسليم النظام الكامل , فإذا اكتشف وجود مشاكل في المتطلبات فإن العميل يقوم بإخطار فريق التطويربذلك ويطلب إصداراً من المقطع , وهذا معناه أن الوظائف الأكثرأهمية بالنسبة للعميل تلقي المراجعة والتصحيح الأكبر , كما أن المقاطع الأخرى تضاف إلى المقاطع الموجودة سابقاً متى ما يتم الانتهاء منها ومن ثم تتم تجربة النظام المتكامل , ولذلك فإن المقاطع الموجودة سابقاً تجرب مرة أخرى بأوضاع اختبارات جديدة كلما تمت إضافة آخر.

- **فرق عمل الغرفة النظيفة (Cleanroom Process Teams) :**

هناك ثلاثة فرق عندما يتم القيام بتطوير أنظمة غرف نظيفة كبيرة الحجم وهي :

① فريق المواصفات : مسئول عن تطوير وصيانة مواصفة النظام , هذا الفريق يقوم بإنتاج المواصفات المتعلقة بالعملاء (تحديد المتطلبات) والمواصفات الحسابية للتحقق وفي بعض الحالات وعند الانتهاء من المواصفات فإن الفريق يشارك في عملية التطوير.

② فريق التطوير : هذا الفريق مسئول عن التطوير والتحقق للبرمجية , ولا يتم تشغيل البرمجية خلال مرحلة التطوير ويستخدم أسلوب منهجي هيكلي للتحقق مبني على فحص الكود معزز بمحاولات تصحيحية.

③ فريق التحقق والمصادقة : هذا الفريق مسئول عن تطوير مجموعة من الاختبارات الإحصائية لاختبار البرمجية بعد تطويرها ويتم بالتوازي معها تطوير حالات الاختبار وتستخدم حالات الاختبار للشهادة على اعتمادية البرمجية ويمكن أن يستخدم نموذج الاعتمادية.

- ميزات أساسية للبرمجيات المنتجة بالغرف النظيفة :

إن استخدام طريقة الغرف النظيفة أدت إلى تطوير برمجيات قليلة الأخطاء وليست أكثر تكلفة من الأنواع التقليدية ، وقد ناقش الباحثان كوب وميلز (1990م) عدداً من المشاريع التي استخدمت فيها هذه الطريقة وكان فيها عدد قليل من الأخطاء في الأنظمة التي تم توريدها ، وقد كانت تكاليف هذه المشاريع مماثلة مع المشاريع الأخرى التي استخدمت فيها الطرق التقليدية.

إن التحقق الساكن فعال في التكلفة باستخدام أسلوب الغرف النظيفة ، وذلك لأن النسبة الكبيرة من الأخطاء يتم اكتشافها قبل التنفيذ ولا تدخل في البرمجية المطورة , وقد وجد الباحث ينجر (1994م) أن نسبة الأخطاء المكتشفة في المتوسط , كانت فقط 3.2 خطأ لكل ألف سطر من نص شيفرة المصدر خلال اختبار المشاريع المطورة باستخدام أسلوب الغرفة النظيفة , كما أن التكاليف الإجمالية للتطوير لم ترتفع وكان الجهد المطلوب في الاختبار وتصحيح الأخطاء أقل في هذه الحالات.

كما قام الباحثون سلبي وآخرون (1987م) بمقارنة التطوير (بالاستعانة بطلبة لمطوري البرنامج) باستخدام أساليب الغرف النظيفة والطرق التقليدية , حيث

وجد أن البرامج التي تم تطويرها كانت في مجملها أكثر جودة من تلك التي تم تطويرها باستخدام الأساليب التقليدية . شيفرة المصدر كانت تحتوي على تعليقات وقد كانت أبسط في التركيب إضافة إلى الالتزام بالمواعيد المحددة.

إن التطوير باستخدام أساليب الغرف النظيفة يعمل جيدا عندما يقوم به مهندسون متخصصون وملتزمون ، وهذا مقتصر فقط على المنشآت المتقدمة تقنياً. إن التقارير عن النجاحات المحققة بطريقة الغرف النظيفة في الصناعة ، كانت في أغلبها وليست كليا من المطورين ، لذلك فإننا لا نعلم ما إذا كان بالإمكان تطبيق العملية لمنشآت تطوير البرمجيات الأخرى علماً بأن هذه المنشآت لديها دائماً عدد أقل من المهندسين ،و لذلك فإن تطبيق أساليب الغرف النظيفة في هذه المؤسسات لا يزال حتى الآن تحدياً.

الخلاصة

اشتملت هذه الوحدة على:

– تعريف كل من : التحقق: وهويعني إنتاج برمجية خالية من الأخطاء.

المصادقة:وهي تعني التأكد من تلبية البرنامج لمتطلبات العملاء.

– طبيعة الأخطاء: تعرفنا هنا على أهم أنواع الأخطاء التي قد تظهر عند تشفير (تكويد) أحد المكونات:وهي عدم إعطاء قيم بدائية للبيانات ،و تكرار الحلقات عدد مرات غير صحيح ،و أخطاء في القيم الحدية.

– التحقق والمصادقة الساكنة والديناميكية: وتشمل طريقتين للفحص والتحليل:

* فحص البرنامج:هذا ويشمل التحليل والتدقيق على مخططات البرنامج المختلفة مثل مستندات المتطلبات ،و مخطط التصميم ، وشيفرة البرنامج المصدر.

* اختبار البرنامج:هذا ويشمل تنفيذ تطبيق البرنامج عن طريق بيانات اختيارية ومعاينة مخرجات البرنامج ، وهناك اختبارات متعددة منها : اختبار الخلل ، والاختبار الإحصائي.

– اكتشاف تصحيح الأخطاء:وهو العملية التي تحدد مكان الأخطاء حيث يتم تصحيحيها.

– التخطيط للتحقق والمصادقة:وقد تعرفنا هنا على عدد من النماذج:

* نموذج الاختبار والتطوير: حيث يتم تقسيم أنشطة التحقق والمصادقة إلى عدد من المراحل بحيث تقاد كل مرحلة من الاختبارات التي حددت لها اكتشاف مطابقة البرنامج لمواصفاته وتصميمه.

* هيكل خطة اختبار البرمجيات: تأخذ هذه الخطة في الاعتبار كميات كبيرة من الإعمال المضاعفة (حالات الطوارئ) بحيث يتم الإحاطة الكاملة بالتأخيرات الخاصة على التصاميم التنفيذ.

– فحوصات البرامج: مراجعات هدفها اكتشاف الأخطاء التي قد تكون موجودة في البرنامج ، وتعرفنا هنا على:

* فرق التفتيش (الفحص): يقوم بها في العادة فريق مكون من أربعة أشخاص.

* الشروط المسبقة للتفتيش:وهي أن يكون هناك مواصفات دقيقة ،و أن يكون أعضاء فريق الفحص على دراية بمعايير المؤسسة و أن يكون هناك نسخة صحيحة من حيث القواعد وحديثة من حيث البرنامج الذي يراد فحصه ،كما يجب أن تتوقع الإدارة أن عملية الفحص ستزيد من التكلفة،و يجب أن لا تستخدم الإدارة عملية الفحص لتقويم العاملين.

* عملية التفتيش (الفحص).

* قوائم الفحص.

* معدلات التفتيش.

– التحليل الآلي الساكن:وهو أدوات برمجية تمر على جميع محتويات نص البرنامج المصدر وتكشف الأخطاء التي فيه . وتعرفنا هنا أيضاً على:

* فحوصات التحليل الساكن: ويقصد بها جذب انتباه الشخص القائم على التحقيق للكائنات الشاذة في البرنامج مثل استخدام متغيرات لم يتم استهلالها.

* مراحل التحليل الساكن: ويشمل : تحليل مسار التحكم ، تحليل استخدام البيانات ، تحليل الواجهة ، تحليل تدفق المعلومات ، تحليل المسار.

* استخدامات التحليل الساكن.

– اختبار الصندوق الأسود:وقد تعرفنا هنا على قواعد اختيار بيانات اختبار الصندوق الأسود باستخدام تكافؤ التقسيم و هي:

* تقسيم كل قيم بيانات المدخلات * اختيار بيانات تمثل الكل من كل جزء أو قسم (بيانات متكافئة) * اختيار بيانات متاخمة (بالقرب من) الأقسام.

اختبار الصندوق الأبيض (التركيبي): يستغل اختبار الصندوق الأبيض معرفة كيف يعمل الإجراء (procedure) حيث يستخدم قوائم شفرة البرنامج، اختبار الصندوق الأبيض يجب أيضاً أن يختار بيانات الاختبار التي:

* تكتشف القرارات التي تتحكم في اختيار الممرات.

* تهتم بأحداث الاختبار التي تؤخذ في حالات خاصة.

– اختبار بيتا: وفيهيتم طرح احد الإصدارات التمهيدية من البرنامج للعميل والذي يكون على دراية بأن المنتج به عيوب.

– اختبار اندماج النظام: توجد ثلاثة اتجاهات لاختبار النظام:

* الاختبار الكبير:وهو وضع كل المكونات معاً بدون اختبار سابق ثم اختبار النظام بالكامل.

* الاختبار الكبير المطور: وهواختبار كل مكون من المكونات بصورة فردية ثم وضع كل المكونات معاً واختبار النظام بالكامل.

* الاختبار المتزايد:وهو بناء النظام جزءاً جزءاً واختبار النظام الجزئي في كل مرحلة,غير أن هناك طريقتين للاختبار المتزايد: الاختبار من الأسفل إلى الأعلى (التصاعدي)، والاختبار من القمة إلى الأسفل (التنازلي).

– تطوير البرمجيات بطريقة الغرفة النظيفة: وهوفلسفة تطوير برمجيات مبنية على تفادي الأخطاء في البرمجة باتباع عملية فحص صارمة ، وتتلخص خصائص الغرف النظيفة في:

التوصيف الرسمي للبرمجة ،و البرمجة الهيكلية ،و الاختبار الإحصائي للنظام ، والتطوير التزايدي.

المراجع

أولاً : المراجع العربية

[1] مهندس عبد الحميد بسيوني ، "أساسيات هندسة البرمجيات" ، دار الكتب العلمية للنشر والتوزيع ، القاهرة ، 2005 م.

ثانياً : المراجع الإنجليزية :

[2] Ian Somerville , "Software Engineering", Addison Wesley, 2001.

[3] Ronald J. Leach,, "Introduction to Software Engineering", CRC Press, 1999.

[4] Douglas Bell , "Software Engineering A Programming Approach", 3rd Edition, Addison Wesley.

[5] **Shari Pfleeger, "Software Engineering - Theory and Practice", 2nd Edition.**

ثالثاً : مواقع على شبكة الإنترنت تم الاستفادة منها :

[6] www.icis.hq.dla.mil/systemdoc/FD/plan/chapter2.htm ,
" Chapter 2 - Software Verification and Validation ".

[7] http://csdl.computer.org/comp/mags/so/1989/03/s3010abs.htm ,
" Software Verification and Validation: An Overview"

[8] http://www.fabricadesoftware.cl/documentos/ESA/PSS0510.pdf
" Guide to software verification and validation"

[9] WWW. sunset.usc.edu/~neno/cs477_2003/April10.ppt
"Verification and Validation"

[10] www.comp.lancs.ac.uk/computing /
resources/IanS/SE7/Presentations/PPT/ch22.ppt
"Verification and Validation"

[11] www.cse.mrt.ac.lk/lecnotes/cs302/ch19-v-v.ppt
"Verification and Validation"

[12] www.i-u.de/schools/heinz/ EVOOC/2002-3-IT260/Slides/Session-10.ppt
"Verification and Validation"

[13] www.int.gu.edu.au/courses7014 /int/lectures/QualityPlan2.ppt
" Software Quality Planning "