Automatic Test Cases Generation Using an Advanced GEP Method

Najla Akram Al-Saati Roua Basil
dr.najla_alsaati@uomosul.edu.iq
College of Computers Sciences & Mathematics / University of Mosul
Received on: 22/03/2011 **Accepted on: 21/06/2011**

Abstract

This research aims to provide a practical work on the principle of the Extreme Programming (XP) which is a type of the Agile Software Development Methods which is used in the generation of test-cases using the design information. The resources utilized in the design information presented here are the design diagrams generated using the Unified Modeling Language (UML), as they are considered to be the most commonly used modeling language in these days, and also the newest. These UML diagrams are used to automatically develop a set of high quality test cases which are then used to test the system's code after being written.

The main idea of this work is based on reducing the effort of the testing stage which costs more than 50% of the resources allocated for the whole development process; this cost may include the financial cost, the cost of the resources allocated for the project, and the timeline of the project.

In this work, enhancements have been made to the concept of Gene Expression Programming to ensure the generation of high quality the test cases that are generated automatically, and a solution has been presented for the parallel paths and the loop paths problems that are found in the design.

***Keywords**: Gene Expression Programming, Test Cases, Unified Modeling Language, Extreme Programming.*

التكوين التلقائي لحالات الاختبار باستخدام طريقة مطورة من البرمجة بالتمثيل الجيني

نجلاء اكرم الساعاتي رؤى باسل ابراهيم
كلية علوم الحاسوب والرياضيات / جامعة الموصل

تاريخ استلام البحث: 2011/03/22 تاريخ قبول البحث: 2011/06/21

الخلاصة

يهدف هذا البحث الى توفير تطبيق عملي لمبدأ البرمجة القصوى (Extreme Programming (XP)) في تطوير البرامج والتي هي احد انواع طرق تطوير البرامج السريعة (Agile Software Development Methods) والتي تعمل على توليد حالات الاختبار للنظام باستخدام معلومات المتوفرة من تصميم النظام. المصادر المستخدمة في هذا العمل كمعلومات عن التصميم هي مخططات التصميم التي توفرها لغة التصميم الموحدة (UML) لكونها الاساس في عمليات التصميم المستخدمة حاليا في العالم و اكثرها انتشارا واحدثها. استخدمت هذه المخططات لتطوير مجموعات من حالات الاختبار عالية الجودة بصورة تلقائية لكي يتم استخدامها لاحقا لغرض اختبار البرامج المكونة للنظام بعد كتابتها.

ترتكز الفكرة الاساسية لهذا العمل على تقليل تكلفة مرحلة الاختبار لكونها تكلف اكثر من 50% من تكلفة تطوير المشروع باكمله ، والتي تتمثل بتكلفة المشروع المالية ، المصادر المخططه للمشروع ، والوقت المستغرق للتطوير.

تم في هذا العمل تحسين مبدأ البرمجة بالتمثيل الجيني (Gene Expression Programming) لكي يضمن توليد حالات اختبار عالية الجودة بشكل تلقائي ، وكذلك تم ايجاد حلول لمشاكل المسارات المتوازية والمسارات الدائرية المتعارف عليها في التصميم.

***الكلمات المفتاحية:** البرمجة بالتمثيل الجيني ، حالات الاختبار ، لغة التصميم الموحدة ، البرمجة القصوى.*

1. المقدمة:

تعتبر عملية اختبار البرنامج (Software Testing) من أصعب المهام في مرحلة تطوير النظام (Software Life Cycle) ، حيث أنها تستنفذ بحدود 50% من تكلفة تطوير البرامج والأنظمة [1]، وأن أهم جزء في هذه المرحلة هو عملية اختيار حالات الاختبار بالجودة المطلوبة. تشكل مرحلة اختبار البرامج أول المراحل التي يتم إلغائها من قبل المطورين في حالة ضيق الوقت أو المصادر المخصصة للمشروع، فقد لا يملك المطور الوقت اللازم لتكوين حالات لاختبار البرامج بعد كتابتها مما يؤدي إلى مشاكل كبيرة في حالة كون البرنامج لا يتوافق مع المتطلبات المحددة من قبل المستخدم وذلك لكون عملية التغيير ستشمل مرحلة تحليل المتطلبات وتصميمها ثم أعادة كتابة البرامج مرة أخرى على حسب التغيير الحاصل مما يؤدي إلى ضياع الوقت والمصادر [2]. لذا فأن عملية تكوين حالات الاختبار للبرنامج قبل كتابته توفر الكثير على المطور وكذلك تضمن كون المتطلبات المحددة للنظام قد تم تطبيقها بتفاصيلها وبالجودة المطلوبة.

في معظم الأحيان تكون مواصفات البرمجيات (Software Specification) ممثلة بمخططات (diagrams) أو مكتوبة باستخدام لغات موحدة لكتابتها (Formal Language Specifications)، وقد تكون مكتوبة باستخدام مواصفات اللغات الطبيعية (Natural Language Specifications). أما أحدث وأوسع هذه الأنواع استخداما في العالم في الوقت الحالي فهي باستخدام التصاميم المكونة بلغة UML (Unified Modeling Language) والتي تم اعتمادها في هذا العمل، كذلك تم تطوير مبدأ البرمجة بالتمثيل الجيني (Gene Expression Programming) لكي يعمل على تكوين أجيال من حالات الاختبار من مرحلة التصميم. التطوير الحاصل على مبدأ التمثيل الجيني تجسد في تغيير مبدأ تمثيل الكروموسوم، وعلى التغيير الحاصل في دالة المستخدمة لإيجاد أفضل الحالات للاختبار، وكذلك طبيعة الناتج من عملية التوليد.[5]

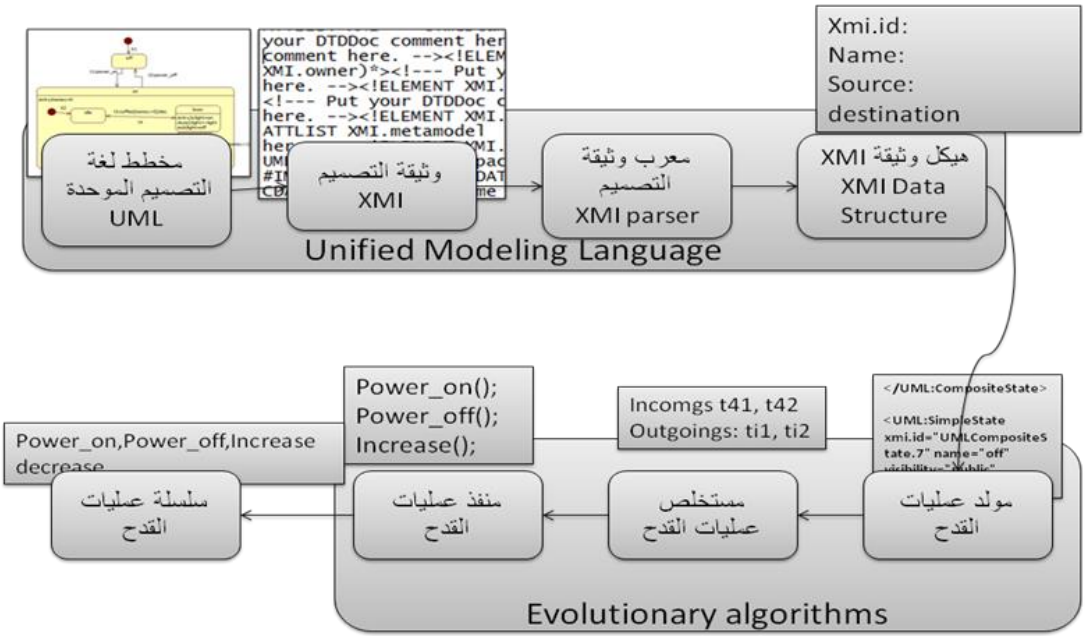
قسم العمل في هذا البحث إلى ستة أقسام : القسم الأول يعنى بتوضيح آلية العمل التي تم اقتراحها والعمل بها في هذا البحث ، والقسم الثاني يوضح مبدأ البرمجة القصوى (eXtreme Programming XP) ويتمحور القسم الثالث حول مرحلة اختبار النظام أو البرامج المكونة للنظام المطلوب ويهتم القسم الرابع بتوضيح UML، ويرتكز القسم الخامس على شرح مبدأ استخدام البرمجة بالتمثيل الجيني المطورة، أما الجزء الأخير فيوضح النتائج التي تم الحصول عليها.

2. آلية العمل:

1.2 هيكل العمل المقترح:

يمكن توضيح آلية العمل التي تم استخدامها في هذا البحث من خلال المخطط (1)، إذ تبدأ مراحل البحث باستخدام الوثائق الخاصة بالتصميم والممثلة بوثائق التصميم الموسعة XMI (XML Meta model Interchange) من اجل إجراء عمليات المعالجة عليها ومن ثم استخلاص المعلومات المطلوبة في عملية التنفيذ للتصميم واكتشاف المسارات، تلك المسارات يمكن استخدامها لضمان إنتاج حالات لعملية اختبار البرنامج بحيث تكون بالجودة المقررة وتفي الغرض المطلوب. [3, 11]

يبين المخطط (1) آلية العمل التي تقسم إلى مرحلتين ، المرحلة الأولى تُعنى بمعالجة تصاميم UML والمرحلة الثانية تعمل على تكوين حالات الاختبار باستخدام المبدأ المطور عن البرمجة بالتمثيل الجيني (Advanced Gene Expression Programming Approach).



المخطط (1). آلية العمل المستخدمة في البحث

وقد تم تقسيم المرحلة الأولى من العمل إلى عدة أقسام كل منها يعمل كالتالي:

1- التصميم باستخدام لغة التصميم الموحدة UML Diagram:

يتم في هذا القسم إدخال التصميم الذي سيتم التعامل معه في العمل وقد تم اختيار مخطط حالة الآلة (State Machine Diagram) لكونه يتعامل مع التغير في حالة النظام من حالة لأخرى تبعا للعمليات التي يتم تنفيذها على النظام ممثلة بعمليات القدح (Triggers).

2- وثيقة التصميم XMI :

تستخدم هذه الوثيقة في جميع أنواع البرامج التي تتعامل مع لغة التصميم الموحدة لغرض تناقل المعلومات من تصميم لآخر. وهي عبارة عن مجموعة من الإيعازات التي توضح معلومات عن التصميم المستخدمة مثل أسماء الأدوات المستخدمة في التصميم مع الرقم التعريفي لكل منها كذلك معلومات عن الشكل الخارجي للتصميم من ناحية اللون والحجم ونوع الخط المستخدم. يستفاد من هذه الوثيقة في استخلاص المعلومات المطلوبة لغرض تنفيذ التصميم قبل تحويله إلى برنامج (أي أن التنفيذ هنا يكون على تصميم وليس على برنامج) وكذلك من اجل استخلاص عمليات القدح (Triggers) والانتقالات (Transitions) والحالات (States) ومعلومات أخرى يستفاد منها في هذا العمل. هناك أنواع مختلفة من الإصدارات لهذه الوثيقة، كل إصدار يختلف عن غيره اختلافات جذرية إلا أن نوعية المعلومات المعطاة تكون المثل في كلها. هذه الاختلافات جاءت من اختلاف نوعية التطبيقات التي تستخدم هذه الوثيقة ، تم في هذا العمل اعتماد الإصدار رقم (1.1) لوثيقة التصميم الموسعة XMI.

3- XMI Parser:

الإدخال لهذا القسم من العمل هو وثيقة التصميم الموسعة ((XMI Document(XMI) حيث يتم تعريف هذه الوثيقة لغرض التخلص من المعلومات الزائدة عن الحاجة مثل معلومات الشكل الخارجي للتصميم واللون والحجم ونوع الخط وكثير من المعلومات التي تستخدم فقط من اجل شكل التصميم الخارجي، أما المعلومات

المهمة مثل أسماء العمليات والإدخالات لها والإخراجات والإجراءات المستخدمة في كل عملية فيتم الاحتفاظ بها لأجل المعالجات التالية.

4- هيكل التصميم XMI Data Structure:

يتم هنا تحويل المعلومات المستخلصة من القسم السابق إلى هيكل منظم يحتوي لكل من الانتقالات المستخلصة الحقول التالية:

الرقم التعريفي للانتقال Transition identification number

اسم الانتقال Transition Name

مصدر الانتقال Source

هدف الانتقال Target

الحالة التي سيتم التغير لها بعد تنفيذ هذا الانتقال State

هذا المعلومات سيتم استخلاصها لكل من الانتقالات الموجودة في النظام.

5- مولد عمليات القدح Trigger Generator:

يجري هنا استخلاص أسماء عمليات القدح المستخدمة في النظام من وثيقة التصميم الموسعة XMI Document، الهدف هو التعرف على عمليات القدح المستخدمة لغرض الاستفادة منها في معرفة المسارات الممكنة في النظام الممثل بنماذج UML.

6- مستخلص عمليات القدح Trigger Extractor:

وهو يعمل على اخذ سلسلة من عمليات القدح التي تم التعرف عليها في القسم السابق لغرض تنفيذها في التصميم وملاحظة مدى جودتها.

7- منفذ عمليات القدح Trigger Executor:

تستخدم هنا المعلومات التي تم استخلاصها من معرب XMI (XMI Parser) لأجل تنفيذ التصميم على السلسلة المقترحة من القسم السابق لاختبار حالة النظام تجاه السلسلة المدخلة ، هذه المرحلة تعمل على تحديد دالة اللياقة لكل من السلاسل المنفذة والتي تحسب من خلال عدد عمليات القدح التي يتم تنفيذها فعلا في هذه السلسلة من العمليات.

8- سلسلة عمليات القدح Trigger Sequence:

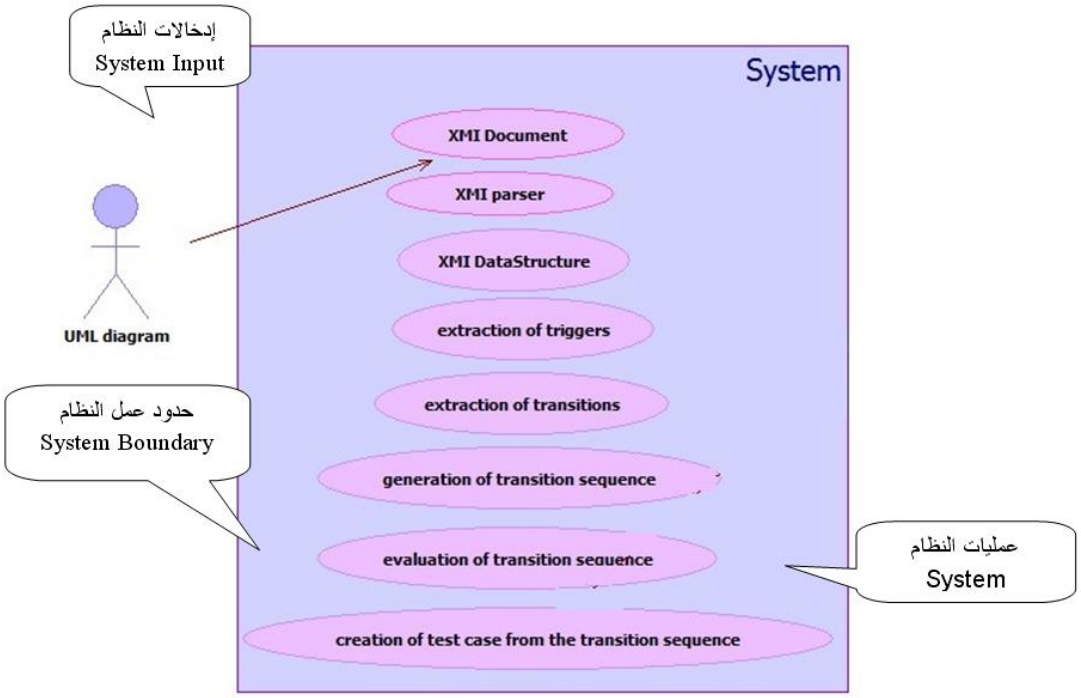
في آخر قسم من العمل تحدد سلاسل التنفيذ ذات الجودة العالية (التي تمتلك دالة لياقة عالية) لغرض اعتمادها كسلسلة للتنفيذات في مرحلة اختبار النظام. استحصل في هذا العمل على سلاسل تنفيذية ذات جودة عالية تغطي 90% من النظام وهي أفضل ما تم الحصول عليه في هذا المجال.

2.2 هندسة العمل:

2.2.1 مرحلة التحليل Analysis Phase :

احد أنواع النماذج التي توضح بشكل رسومي عملية تحليل المتطلبات للعمل المطلوب هو مخطط حالات الاستخدام (Use-Case Diagram)، وهو (كما ذُكر سابقا) احد المخططات المستخدمة في لغة التصميم الموحدة.[11] لذا وباستخدام تعاريف مخطط حالات الاستخدام فأن النظام المقدم في هذا العمل يخطط بالشكل الموضح بالمخطط (2). يقترح التحليل في هذا المخطط أن من متطلبات النظام كون الإدخال عبارة عن مخطط UML تُجرى عليه عدة عمليات تتضمن:

1- تحويله إلى وثيقة تصميم XMI.
2- استخدام معرب وثيقة التصميم (XMI) لتحويلها إلى وثيقة خالية من المعلومات غير المرغوب بها (Meta XMI) ، مثل معلومات الهيكلية العامة للتصميم من الألوان وأحجام الخطوط وما إلى ذلك.
3- تحويل الوثيقة إلى بيانات مُهيكلة (XMI Data Structure).
4- استخلاص عمليات القدح.
5- استخلاص الانتقالات بين حالات النظام.
6- توليد سلاسل من الانتقالات.
7- اختبار جودة هذه السلاسل.
8- تكوين حالات الاختبار بالاعتماد على السلاسل ذات الجودة العالية.



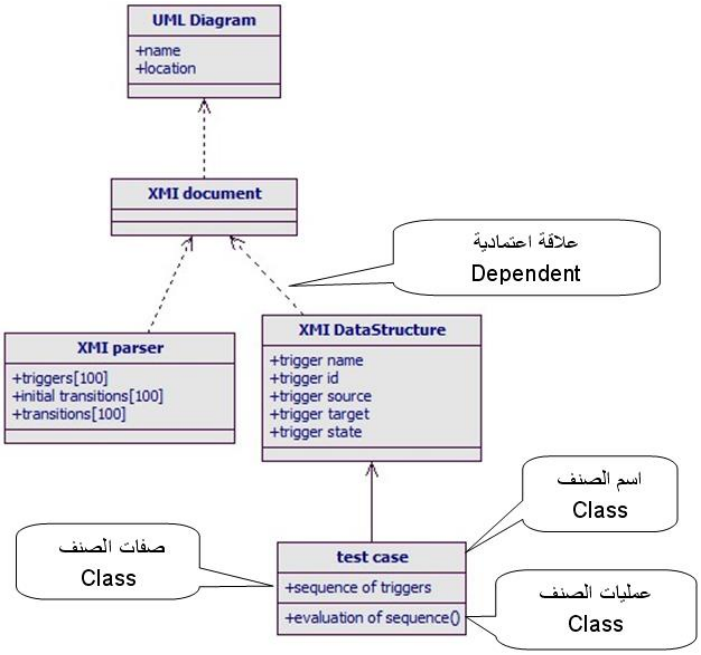
مخطط (2): تحليل نظام تكوي حالات الاختبار

2.2.2 مرحلة التصميم Design Phase :

باستخدام مخطط الأصناف (Class Diagram) في لغة التصميم الموحدة ينتُج المخطط (3). كما هو ملاحظ من المخطط فان النظام قد تم تقسيمه إلى عدة أجزاء والتي مثل كلا منها بصنف (Class) معين.

2.2.3 مرحلة كتابة البرنامج للنظام Coding Phase :

تمت كتابة البرنامج باستخدام لغة (Visual C++ 2008) وبدون استخدام أي مكتبات جاهزة للتعامل مع XMI أو مع UML لعدم توفر أي مكتبة وافية ممكن الاستفادة منها بشكل كامل للتعامل مع وثائق Xmi وUML لذا كان من الأفضل برمجة كل عملية ضرورية لأجل العمل. قد تمت البرمجة بشكل متقن بحيث أن النتائج تظهر بأحسن ما يمكن و كان المقياس هو جودة حالات الاختبار التي تم تكوينها.



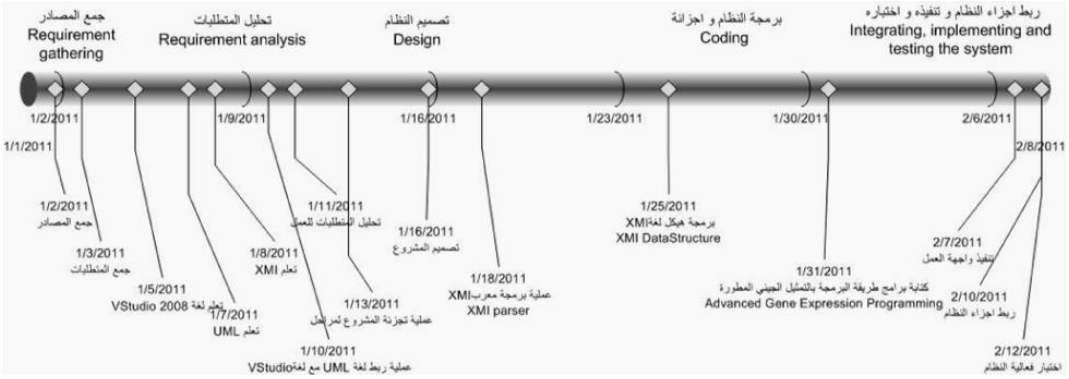
مخطط (3). تصميم نظام تكوين حالات الاختبار

2.2.4 مرحلة الاختبار Testing Phase :

النظام تم اختباره لأكثر من مثال واحد وكانت النتائج تتفاوت بين مثال وآخر إلا أنها لم تقل أبدا عن نسبة جودة 87% مقاسة بحجم التغطية التي توفرها السلاسل المختارة للمثال.

2.3 الجدول الزمني للعمل :

وضع جدول الخطة الزمنية التي تم إتباعها منذ لحظة بدء العمل ولحد انتهائه. يلاحظ في المخطط (4) تقسيم مراحل العمل على حسب مبدأ مراحل تطوير المشروع في هندسة البرمجيات.

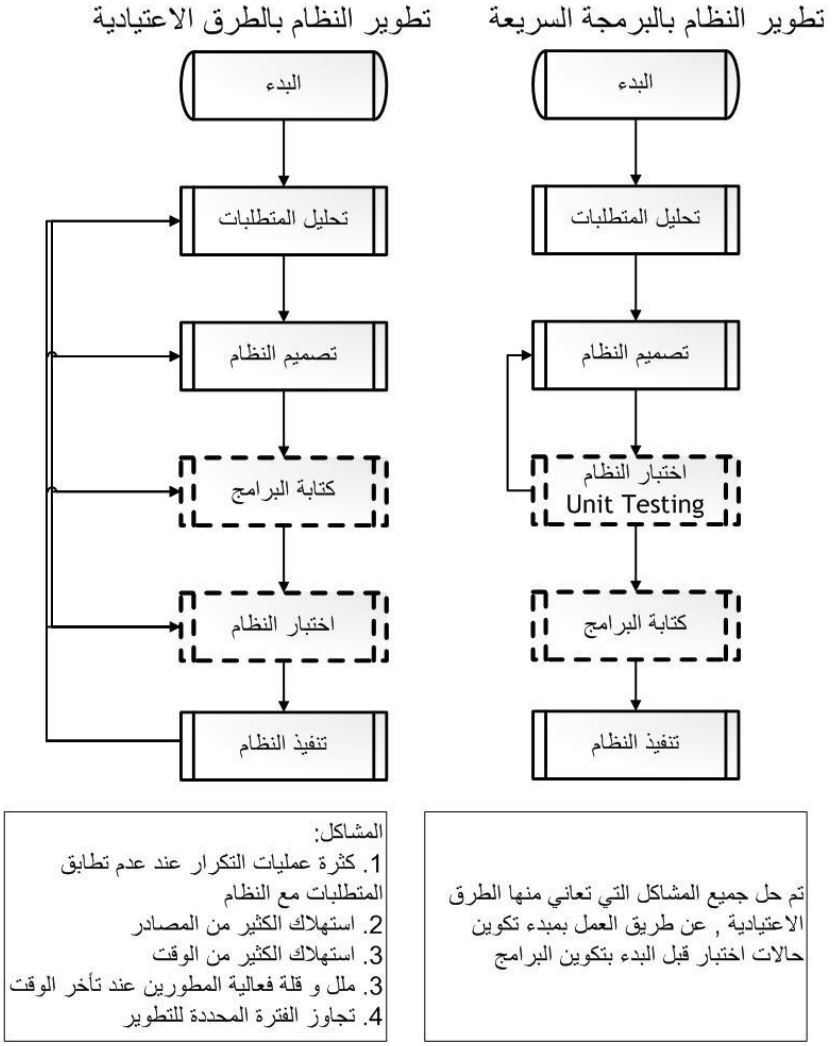


مخطط (4). الجدول الزمني للعمل

1. البرمجة القصوى Extreme Programming(XP) :

تُعرف البرمجة القصوى على أنها طريقة جديدة لتطوير البرامج تُمكن المطورين من إنتاج برامج ذات جودة عالية وبسرعة فائقة, حيث تُعرف الجودة هنا على أنها تطابق البرنامج مع متطلبات النظام المطلوب تكوينه. تختلف هذه الطريقة عن الطرق الاعتيادية في تطوير البرامج في أنها تصب اهتمامها على مرحلة الاختبار Testing. تعد البرنامج في الطرق الاعتيادية بعد عملية التصميم ثم يبنى الاختبار لهذا البرنامج، أما في حالة البرمجة القصوى فيتم تكوين وحدات اختبار (Unit Testing) للبرنامج قبل كتابته، أي تهيئ وحدات الاختبار بعد عملية التصميم ، ثم يتم بناء البرنامج الذي يستطيع تخطي هذه الاختبارات. المخطط (5) يوضح الاختلاف بين تطوير النظام باستخدام البرمجة القصوى وتطوير النظام بالطريقة الاعتيادية. يمكن تقسيم آلية عمل البرمجة القصوى إلى المجاميع الأربعة التالية [6, 8]:

1. الاستماع إلى الزبون وإلى المبرمجين الآخرين.
2. التعاون مع الزبون لتطوير خواص للتطبيق (Application Specification) وحالات الاختبار.
3. البرمجة باستخدام ثنائي برمجي.
4. اختبار البرنامج باستخدام حالات الاختبار.



المخطط (5): الفرق بين التطوير باستخدام البرمجة القصوى والتطوير باستخدام الطرق الاعتيادية

2. اختبار البرامج Software Testing:

أحد أهم عوامل ضمان جودة البرامج هي عملية اختبار البرامج والتي تضمن أن البرنامج الناتج من مرحلة التطوير يتلاءم مع المتطلبات المرجوة من تصميمه، هذا النوع من الاختبارات يسمى باختبار الصندوق الأسود (Black Box Testing) الذي يتم فيه اختبار تطابق البرنامج مع المتطلبات المعرفة لدى بداية مراحل التطوير من ناحية صحة العمل والتي تسمى أيضا بالمتطلبات الوظيفية (Functional Requirement). [6, 9] يقبل البرنامج على أنه ناجح في الاختبار أذا كانت النتائج التي تم الحصول عليها من عملية الاختبار تساوي النتائج المتوقع الحصول عليها، وعلى العكس فأن البرنامج يفشل في حالة كون النتائج من عملية الاختبار مغايرة للنتائج المطلوبة. [5, 7]

تستخلص المتطلبات في مرحلة التصميم من قبل المطور ثم تحول إلى خواص برمجية يكون الزبون بحاجة لها. تؤخذ هذه المتطلبات في مرحلة التصميم لكي تحول إلى وثائق تصميم لاستخدامها في مرحلة تكوين البرنامج. [10] تعتبر هذه الوثائق الناتجة من مرحلة التصميم كدليل للمطور لإنتاج الحالات الخاصة بالاختبار ولغرض الحصول على النتائج المتوقعة من اختبارات الصندوق الأسود. هناك نوعية ثانية من الاختبارات وهي ما تسمى باختبار الصندوق الأبيض (White Box Testing) والذي يُعنى بالعمليات التي يقوم بها البرنامج، أي أنه اختبار لكل المسارات الممكنة في البرنامج بعد تكوينه. تهتم هذه النوعية من الاختبارات بعملية الضمان لجودة البرنامج المكتوب بأي لغة برمجية مستخدمة.[7]

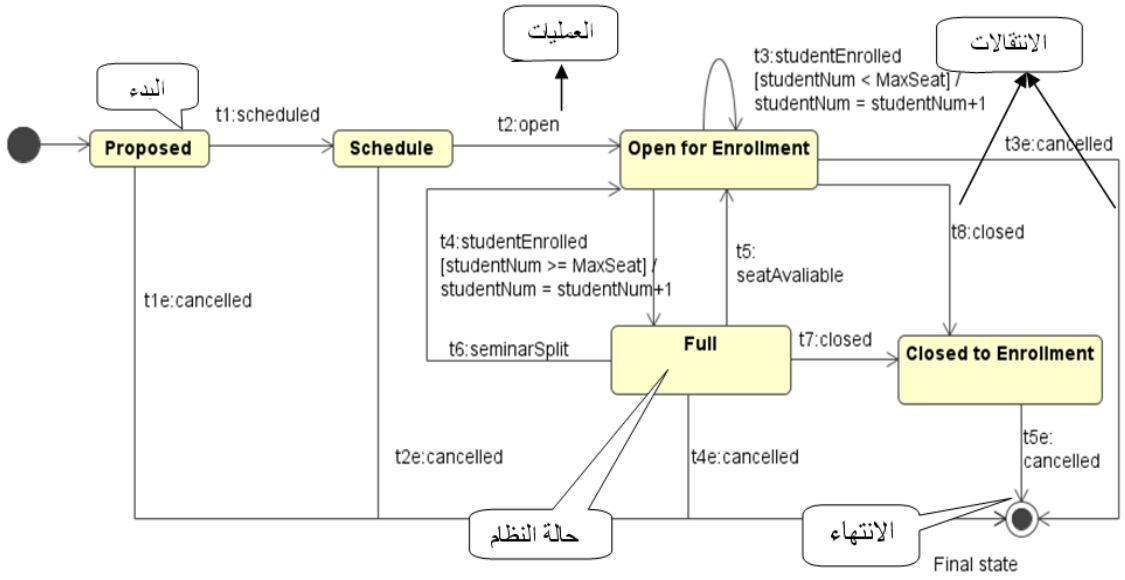
إن حالات الاختبار المستخدمة في هذه النوعية من الاختبارات تعمل على تنفيذ كافة المسارات الممكنة في البرنامج تحت الاختبار. النوعية الثالثة من الاختبارات المستخدمة هي اختبار الصندوق الرمادي (Grey Box Testing) والذي يعتبر دمج للنوعيتين السابقتين من الاختبارات، إذ يقوم بأخذ وثائق خواص البرنامج كإدخال كما هو الحال في اختبار الصندوق الأسود إلا أن هذه الوثائق لا تصف الصفات المطلوبة من النظام فقط بل إنها تصف أيضا حالة النظام طريقة تصرف النظام إزاء حالة معينة وبهذه الصفة فأنه يشابه اختبار الصندوق الأبيض من ناحية استخدامه حالة النظام لأجل تكوين حالات الاختبار كما هو الحال في الاختبار المطبق في هذا البحث. [5]

3. لغة التصميم الموحدة (Unified Modeling Language):

إن لغة التصميم الموحدة (UML) هي عبارة عن لغة تستخدم لتكوين تصاميم بنماذج متعددة للمتطلبات المستخلصة من مرحلة التحليل، هذه التصاميم تكون عبارة عن تمثيل رسومي (Graphical Representation) للمتطلبات وخواص النظام المطلوبة [5].

استخدمت في هذا البحث النماذج التي تتعامل مع حالة النظام لتكوين حالات اختبار التي تعرف عن مدى صحة تصميم وتحليل النظام، وكذلك (State Chart Diagrams) والذي يوضح في المثال بالمخطط (6). يوضح المثال نظام القبول المركزي، وكما هو ملاحظ فإن النظام مكون من حالة بدء ، ويمكن وجود أكثر من حالة بدء واحدة كما في الأنظمة التي تعمل بأكثر من عملية واحدة وبأكثر من مجال في آن واحد (Parallel Systems)، للانتقال بين حالة وأخرى من حالات النظام (الممثلة بالمستطيلات) ولأجل تنفيذ هذه الانتقالات فان كل انتقال يتم تفعيله باستخدام واحدة من العمليات التي تتم في النظام. تؤدي هذه العمليات إلى تنفيذ أحد

الانتقالات لتغير حالة النظام من حالة إلى أخرى. مثال على هذه الانتقالات هو (t4:studentEnrolles[studentNumber>=MaxSeat]/studentNum=studentNum+1).



المخطط (6): مثال مخطط حالة نظام القبول المركزي [2]

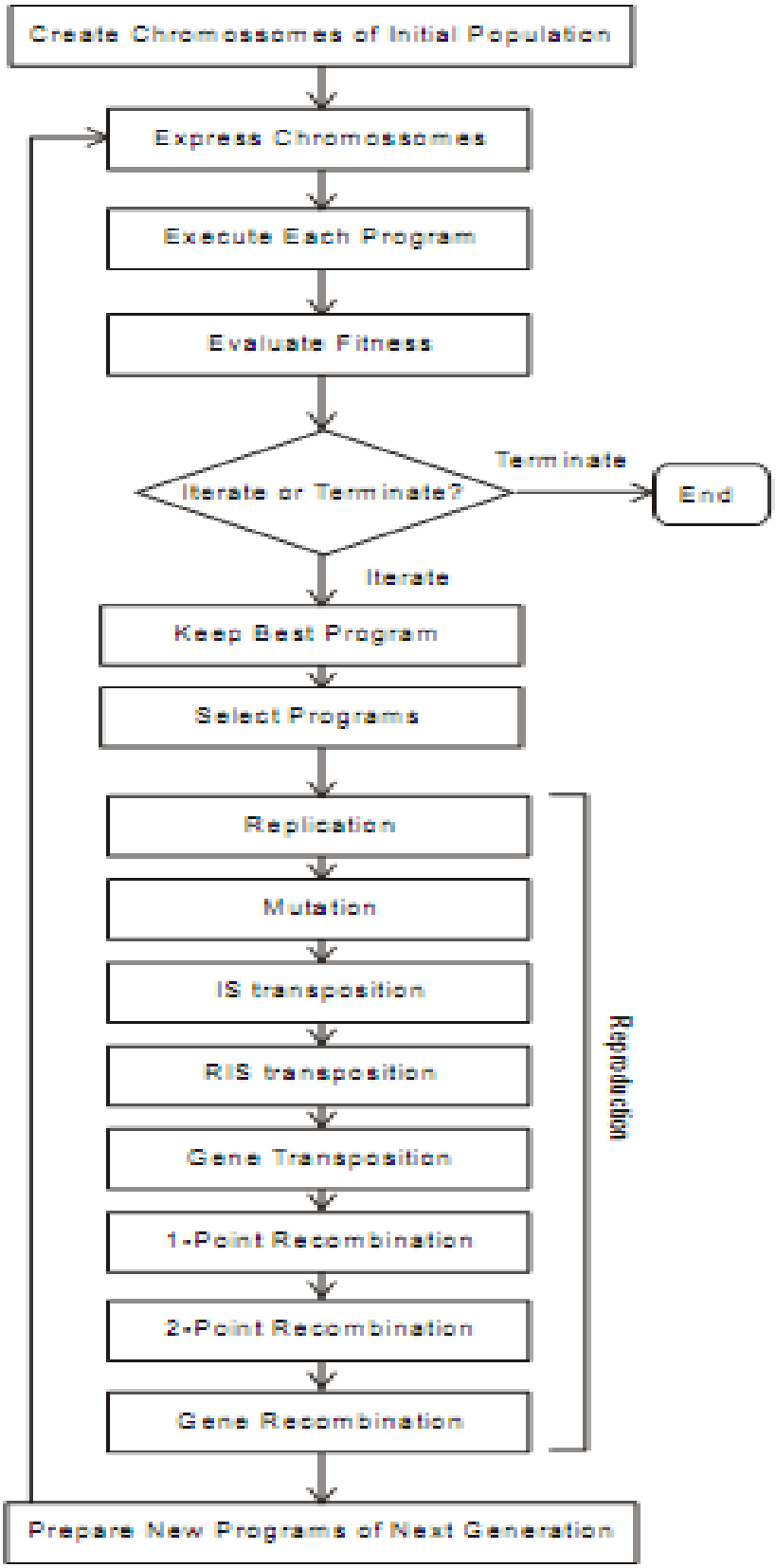
نظام القبول المركزي هو النظام المتبع في قبول الطلاب في الكليات والأقسام, حيث يبدأ بحالة تقديم الطلبات proposed و ينتقل منها إلى الجدولة الخاصة بالقبول, حال فتح القبول ينتقل بعدها إلى حالة التقسيم إلى الكليات والأقسام open for enrollment حيث يقرر عندها إن كان الطلب سيدخل للكلية أو القسم إن كان هناك مقاعد فارغة ويضم الطلب للكلية أو القسم أو إن الطلب سيرفض في حالة عدم وجود مقاعد فارغة وينتهي عندها النظام[2]. وكما هو موضح في هذا المثال، فإن كل عملية قدح تتكون من أربعة أجزاء: أسم الانتقال ممثل بـ (t4)، جزء أسم العملية وهو (studentEnrolles)، الشرط المطلوب لتنفيذ هذه العملية والواقع ضمن الأقواس الوسطية [studentNumber>=MaxSeat] والجزء الأخير وهو ما سيتم عمله في حالة تنفيذ العملية وهو (studentNum=studentNum+1). [3]

4. البرمجة بالتمثيل الجيني المطور (Advanced Gene Expression):

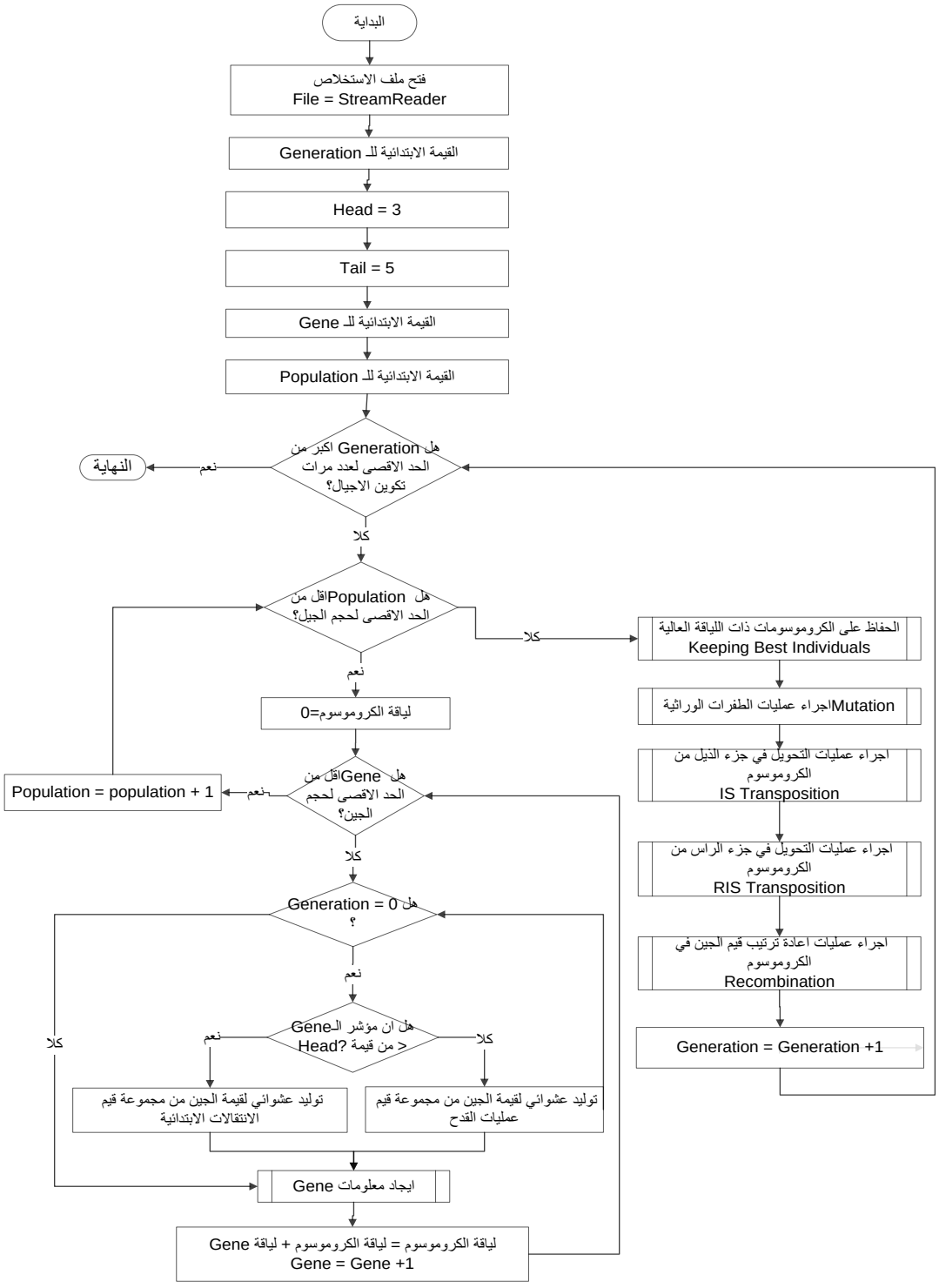
البرمجة بالتمثيل الجيني هي عبارة عن خوارزمية تطويرية تعتمد مبدأ توارث الأجيال, وهي تطوير للبرمجة الجينية. في البرمجة بالتمثيل الجيني, فان الأفراد يتكونون من جمل خطية ثابتة الحجم linear strings of fixed size. كل جيل يتكون من عدة كروموسومات كل واحدة منها تتكون من جين واحد أو أكثر. هذه الجينات مقسمة إلى جزأين, رأس وذيل. يتكون الرأس من عدد من العمليات الرياضية أو المنطقية أو البرمجية فيما يتكون الذيل من عدد من المتغيرات التي ستقام عليها العمليات المذكورة في جزء الرأس.

في البرمجة بالتمثيل الجيني, هناك العديد من العمليات الجينية المشابهة تقريبا لتلك الموجودة في البرمجة الجينية. هذه العمليات تتضمن الاختيار, التبادل, الطفرة, تحويل الرأس, تحويل الذيل, تحويل الجين, تحويل الجذر, إعادة الترتيب المفردة والمزدوجة, وأخيرا عملية إعادة الترتيب الخاصة بالجين [4]. كما هو موضح في المخطط الانسيابي (8).

أما عن مخطط خوارزمية البرمجة بالتمثيل الجيني الاعتيادية فتم توضيحها في المخطط الانسيابي (7).



مخطط (7). المخطط الانسيابي للبرمجة بالتمثيل الجيني بدون عملية التطوير [3].

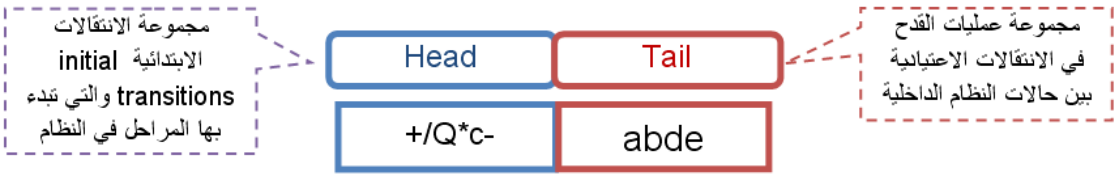


مخطط (8): مخطط البرمجة بالتمثيل الجيني التي تم تطويرها

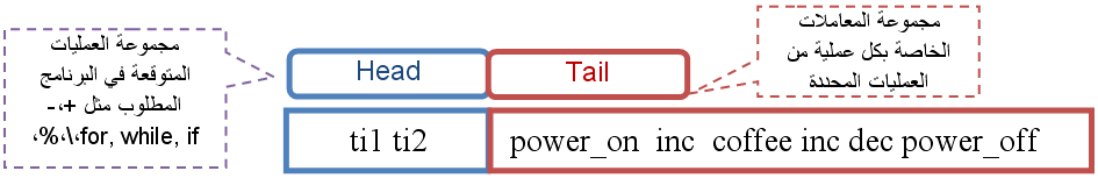
طُور هذا المبدأ ليلائم تطبيقه في مجال تكوين مجموعة من حالات الاختبار عالية الجودة، وشمل هذا التطوير عملية تمثيل الكروموسومات، الدالة المستخدمة لبيان فعالية كل كروموسوم وجودتهFitness Function، بالإضافة إلى ناتج الكروموسومات [3]، شمل هذا التغيير:

1. طريقة تمثيل الكروموسومات في الطريقة الأصلية كان تمثيل الكروموسومات عبارة عن جزأين, الأول يتضمن العمليات و هو جزء الرأس والثاني يتضمن المعاملات أو المتغيرات المستخدمة في العلاقة والتي ستتم عليها تلك العمليات والتي توضع في جزء الذيل كما هو موضح بالشكل (1). أما في الطريقة المطورة هنا فأن تمثيل الكروموسوم أصبح يحتوي على الانتقالات الأولية في جزء الرأس والعمليات التي سيقوم بها الانتقال و المكونة في جزء الذيل وكما هو مبين بالشكل (2).
2. أما التطوير الحاصل في دالة اللياقة Fitness Function فأن طبيعة النظام المراد تطويره هنا قد جعلت هذه الدالة تحسب من خلال عدد الانتقالات التي تم فعليا الحصول عليها من جراء تنفيذ الكروموسوم على التصميم ، وبذلك فكلما زادت قيمة دالة اللياقة كلما كان الكروموسوم أفضل. أي أن دالة اللياقة تحسب على أنها (Fitness Function = Number of Triggered Transitions).
3. التطوير الأخير كان في طبيعة الكروموسومات الناتجة من هذه العملية، ففي الحالات الطبيعية كانت طريقة التمثيل الجيني تستخدم لتكوين برامج تحاكي معطيات معينة، في حين أنها طورت لتعطي سلسلة من الاستدعاءات للعمليات المستخدمة في النظام لغرض إجراء عملية المقارنة بين النتائج المطلوبة والنتائج الفعلية للنظام المطور لغرض إيجاد أفضل جودة للنظام ولحالات الاختبار المكونة.

إن الغرض من هذا التصميم هو حل مشكلة المسارات التي تنفذ في الوقت نفسه أو المسارات المتوازية (Parallel Paths) والمسارات الدائرية (Loop Paths). إذ أن كثرة عدد الانتقالات الابتدائية (Initial Transitions) في تمثيل الكروموسوم فتحت المجال لأكثر من مسار لان ينفذ في ذات الوقت بدلا من الاحتفاظ بمسار واحد في الاختبار وترك باقي المسارات الأخرى.



الشكل (1). تمثيل الكروموسوم في طريقة البرمجة بالتمثيل الجيني الاعتيادية



الشكل (2). تمثيل الكروموسوم في طريقة البرمجة بالتمثيل الجيني المطورة

أما عن المسارات الدائرية فقد تم تطوير خوارزمية خاصة للتعرف عليها وتفادي تنفيذها أكثر من مرة واحدة بدلا من أن يتم تفاديها أو في حالات أخرى الوقوع في دوامة هذه المسارات الدائرية. تقوم الخوارزمية المقترحة لحل هذه المشكلة بتشخيص المسارات الدائرية التي تتساوى فيها مصدر وتبحث عن حل ثاني للخروج من هذا المسار الدائري بشكل صحيح وكما هو موضح في المخطط الانسيابي (9).

5. النتائج والاستنتاجات Results and Conclusions:

قدمت في هذا البحث طريقة لتكوين حالات اختبار باستخدام نماذج لغة التصميم الموحدة UML. كُونت هذه الحالات بصورة تلقائية باستخدام مبدأ مطور للبرمجة بالتمثيل الجيني. وقد تم تقييم جودة الاختبارات الناتجة من خلال قياس عدد عمليات القدح التي تم فعلا تنفيذها في هذه الاختبارات، وتضمن العمل أيضا حل مشاكل المسارات التي تنفذ في نفس الوقت والمسارات الدائرية في التصميم كما هو موضح في المخطط (9). أجريت برمجة النظام على أن يستلم وثيقة من مرحلة التصميم وهي (XMI Document) والناتج يكون عبارة عن مجموعة من المسارات التي سيتم اختبارها فيما بعد عملية تكوين البرنامج. كتبت البرامج باستخدام لغة (Visual C++ 2008) واعتمدت النسخة (XMI Version 0.1).

بعد اعتماد عدد تنفيذات (Runs = 20) تنفيذ لتفادي الحصول على النتائج عن طريق الحظ لكون النتائج تعتمد على إخراجات الدالة العشوائية. نُفذ النظام المقترح على :

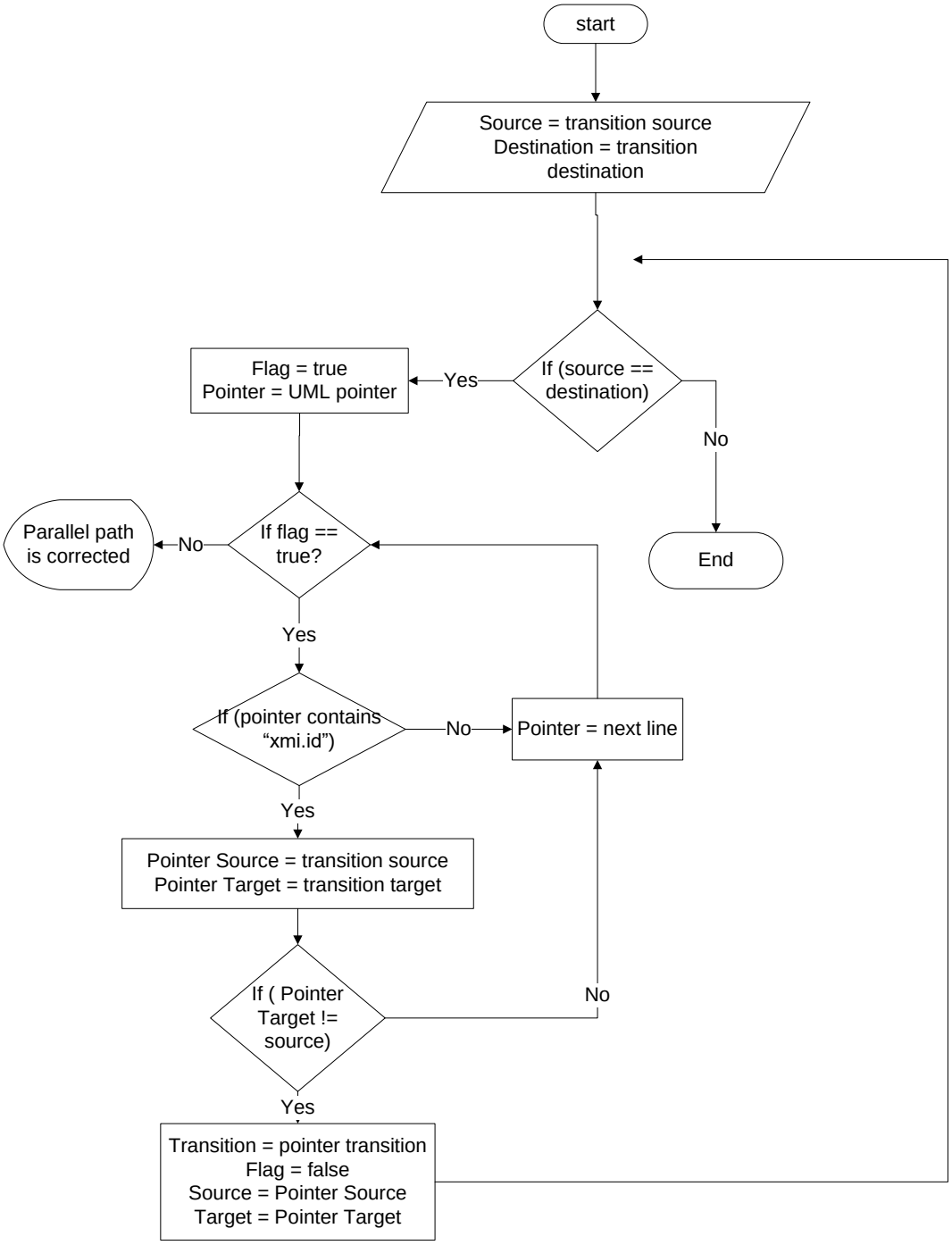
1- عدد حجم أجيال 10 ، 20 ، 30 ، 40

2- نسبة عملية التبادل المستخدمة كان 0.4 ، 0.5 ، 0.6 ، 0.7

3- نسبة عملية الطفرات الوراثية فكانت 0.01 ، 0.1 ، 0.05

4- نسبة عمليات فكانت 0.2 ، 0.4 ، 0.25

تبين النتائج الموثقة بالجدول (1) النسب المئوية لعدد التنفيذات التي تم الحصول فيها على النتائج المطلوبة ممثلة بقيمة دالة لياقة عالية والمتمثلة بنسبة تنفيذات للعمليات في السلسلة المتولدة والتي تتجاوز 80% من حجم السلسلة. وكما هو ملاحظ من الجدول فان أفضل النتائج تم الحصول عليها عندما كان حجم الجيل مكون من 30 كروموسوم. وقد تم الحصول على النتائج المبينة من خلال حساب النسبة المئوية لعدد التنفيذات التي تم فيها الحصول على سلاسل حالات اختبار ذات دالة لياقة تنفذ فعليا أكثر من 70% من عمليات القدح المكونة للسلسلة.



مخطط (9). خوارزمية حل المسارات المتوازية والدائرية

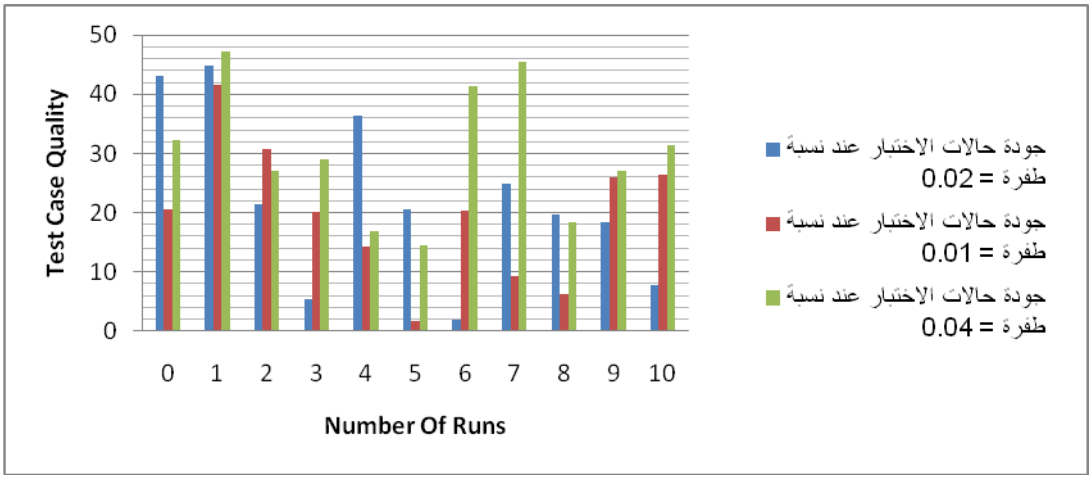
جدول (1). نسب النجاح لعلاقة حجم الجيل مع نسبة التبادل

نسبة التبادل	نسبة النجاح			
	حجم جيل 10	حجم جيل 20	حجم جيل 30	حجم جيل 40
0.4	%80	%90	%100	%100
0.5	%70	%100	%100	%90
0.6	%80	%90	%100	%100
0.7	%80	%100	%100	%90

جدول (2). نسب النجاح لعلاقة حجم الجيل مع نسبة الطفرة

نسبة الطفرة	نسبة النجاح			
	حجم جيل 10	حجم جيل 20	حجم جيل 30	حجم جيل 40
0.01	%100	%100	%100	%90
0.02	%100	%100	%100	%100
0.05	%80	%100	%100	%100

إن النتائج التي تم الحصول عليها في الجدول (2) توضح أن أفضل النسب تم الحصول عليها من جراء استخدام نسبة طفرة = 0.02 والتي أعطت نسبة نجاح 100 % لكل حجم جيل مستخدم.



المخطط (10). جودة حالات الاختبار التي تم تكوينها

يبين المخطط (10) جودة حالات الاختبار المتكونة وعلاقتها مع التغير في نسبة الطفرة وفي عدد (تنفيذات =10). يوضح المخطط انه تم الحصول على أجود النتائج وأفضلها منذ ثاني تنفيذ للبرنامج كما أن النتائج تبدأ بالانحدار تدريجيا مع زيادة عدد التنفيذات والسبب يعود في ذلك إلى النتائج الأولية التي تعطيها الدوال العشوائية المستخدمة في البرنامج لأجل اختيار الكروموسومات التي يتم إجراء عملية التطوير عليها واختيار الجين الموجود في الكروموسوم لإجراء الطفرة عليها وكذلك اختيار قيمة الطفرة الجديدة.

8. الاقتراحات والتوصيات المستقبلية :

تم استخدام التمثيل الجيني المطور من اجل إيجاد حالات اختبار ذات جودة عالية وبصورة تلقائية من مخططات حالة النظام (StateMachine Diagram) الموجود في لغة التصميم الموحدة UML، إن حالات الاختبار المكونة هنا هي عبارة عن سلاسل من عمليات القدح الموجودة في المخطط، كذلك تم تطوير مبدأ التمثيل الجيني لأجل ملائمتها مع طبيعة العمل لغرض إيجاد حالات اختبار، من خلال تطوير تمثيل الكروموسوم، دالة اللياقة المستخدمة، وكذلك على طبيعة الإخراجات الناتجة. تم تطوير طرق لحل مشاكل المسارات المتوازية ومشاكل المسارات الدائرية والتي تعتبر من أكثر مشاكل التصميم تعقيدا والتي لم يتم الحصول على حلول لبعض منها لحد هذا اليوم . كذلك تم تطوير أدوات لغرض التعامل مع وثائق التصميم التي تعرف بصعوبتها وصعوبة التعامل معها.

أما عن المقترحات المستقبلية فان بالإمكان استخدام أنواع أخرى من المخططات مثل مخططات الفعاليات (Activity Diagram) أو مخططات التسلسل (Sequence Diagram) أو مخططات التعاون (Collaboration Diagram) من اجل إيجاد حالات الاختبار أفضل. بالإضافة إلى ذلك فانه بالإمكان استخدام طرق الذكاء الاصطناعي الأخرى كالخوارزمية الجينية أو البرمجة الجينية التي تتواءم مع طبيعة هذا العمل مثل خوارزمية النمل أو النحل أو غيرها لإيجاد أفضل حل ممكن.

المصادر

[1] Cheon, Y., Kim, M.Y. and Perumandla, A., (2005), A Complete Automation of Unit Testing for Java Programs. In the proceedings of the 2005 International Conference on Software Engineering Research and Practice (SERP '05. Las Vegas, Nevada, USA.

[2] Doungsa-ard, C., .et al, (2007), Test Data Generation from UML State Machine Diagrams using GAs. In the Second International Conference on Software Engineering Advances, ICSEA 2007. French Riviera, France. (Summited)

[3] Ferreira, C., (2001), Gene Expression Programming: a New Adaptive Algorithm for Solving Problems, in Complex Systems, 13(2), pp: 87-129.

[4] Ferreira, C., (2002), Gene Expression Programming in Problem Solving, in Roy, R., Köppen, M., Ovaska, S., Furuhashi, T., and Hoffmann, F., eds., Soft Computing and Industry – Recent Applications, Springer-Verlag, pp: 635-654.

[5] Kim,H., Kang, S., Baik, J., Ko, I.,(2008), Test Cases Generation from UML Activity Diagrams, ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel / Distributed Computing. pp: 556-561.

[6] Myers, J., (2004), The Art of Software Testing, John Wiley & Sons, 2nd edition, 255p.

[7] Offutt, J., and Abdurazik, A., (2000), Generating Tests from UML Specifications. In 3rd International Conference on the Unified Modeling Language (UML'00), York, UK, October 2000, pp: 383-395.

[8] Pressman, R., (2001), Software Engineering - A Practitioner's Approach, McGraw-Hill series in computer science, 5th edition, pp: 437-507.

[9] Pickin, S., Jard, C., Le Traon, Y., Jéron, T., Jézéquel, J., (2005), System Test Synthesis from UML Models of Distributed Software, This work has been partially supported by the COTE RNTL National Project and the CAFÉ European project. Eureka ∑! 2023 Programme, ITEA project IP 0004, 16 P.

[10] Shabani, K., Ahmadi, M., Keshavarz, S., Babamir, F., Babamir, S., (2010), GA based-Software Test Data Generator, Using Dynamic Repetition Frequency, July 2010, (IJCNS) International Journal of Computer and Network Security, Vol. 2, No. 7, pp: 6-10.

[11] OMG, (2005) MOF 2.0 / XMI Mapping Specification, v2.1.