مجلة جامعة تشرين للبحوث والدراسات العلمية _ سلسلة العلوم الهندسية المجلد (40) العدد (4) 2018
Tishreen University Journal for Research and Scientific Studies - Engineering Sciences Series Vol. (40) No. (4) 2018

تقييم وتحسين جودة البرمجيات باستخدام منهجية CRISP-DM

د. علي سليمان *

(تاريخ الإيداع 11 / 7 / 2018. قُبِل للنشر في 12/ 8 / 2018)

□ ملخّص □

تعتبر هندسة البرمجيات مفهوماً هاماً لتطوير النظم المعلوماتية، وهي أكثر من عملية ترميز أو تشفير للبرامج، حيث تتضمن الجودة والجدولة والاقتصاديات والمعرفة لتطبيق المبادئ. ذلك أن البرمجيات عرضه للأخطاء في التصنيع والتشغيل وتحتاج لعمليات التقييم مثلها مثل أي منتج آخر.

في هذه الورقة البحثية نقدم تطويراً للنموذج الشلالي التقليدي لهندسة البرمجيات باستخدام منهجية *CRISP-DM* المصممة أساساً لهندسة نظم استخلاص البيانات بهدف تحسين جودة البرمجيات، حيث تم دراسة تأثير إدخال هذه المنهجية من ناحية تقليل عدد الأخطاء الموجودة في البرمجيات، وتأثيرها على اكتشاف الأخطاء في مرحلة مبكرة من دورة حياة البرمجية.

أظهرت النتائج أن استخدام المنهجية المذكورة ساعد علي تقليل عدد الأخطاء الموجودة في البرمجية بعد انتهاء عملية التطوير، كما ساعد في اكتشاف نسبة أكبر من الأخطاء في المراحل المبكرة من دورة حياة البرمجية، وهو ما يساهم في رفع جودة البرمجيات، مقابل عدد منخفض نسبياً من الزيادة في أسطر الكود نتيجة تطبيق المنهجية المقترحة.

الكلمات المفتاحية: هندسة البرمجيات – جودة البرمجيات – النموذج الشلالي – منهجية CRISP–DM.

* أستاذ مساعد – قسم الهندسة الطبية – كلية الهندسة الميكانيكية والكهربائية– جامعة تشرين – اللاذقية – سورية

مجلة جامعة تشرين للبحوث والدراسات العلمية _ سلسلة العلوم الهندسية المجلد (40) العدد (4) 2018
Tishreen University Journal for Research and Scientific Studies - Engineering Sciences Series Vol. (40) No. (4) 2018

Evaluate and improve software quality using the CRISP-DM methodology

Dr. Ali Suleiman*

(Received 11 / 7 / 2018. Accepted 12 / 8 / 2018)

□ ABSTRACT □

Software engineering is an important concept for the development of information systems, which is more than coding or writing of programs, that it includes quality, scheduling, economics and knowledge of the application of principles. Software is subject to errors in manufacturing and operation and needs to be evaluated like any other product.

In this research paper, we present a development of the traditional programmatic model of software engineering using the CRISP-DM methodology, which is primarily used in data extraction engineering systems, that aims to improve software quality. We study the effect of introducing this methodology in terms of reducing the number of errors discovered in software and its impact on error detection at an early stage of the software life cycle.

The results showed that the use of this methodology helped to reduce the errors discovered in the software after the development process. It also helped to detect a greater percentage of errors in the early stages of the software life cycle, which helps to increase the quality of the software, compared to a relatively low number of lines. Code because of applying the proposed methodology.

Keywords: Software Engineering – Software Quality – Water flow Model – CRISP-DM Methodology.

*Associate Professor, Department of Biomedical Engineering, Faculty of Mechanical and Electrical Engineering, Tishreen University, Lattakia, Syria.

مقدمة

في عام 2014 أجرى المعهد القومي الأمريكي للمعايير والتكنولوجيا دراسة توقعت أن أخطاء البرمجيات قد تكلف الاقتصاد الأمريكي 60 مليار دولار سنوياً، أو ما يقرب من 6 في الألف من إجمالي الناتج القومي[1]، وأصبحت متابعة أخطاء البرمجيات بالطريقة التقليدية وهي كتابة سطور من لغة البرمجة ثم اختبارها على جهاز الحاسب ومعالجة أي أخطاء تنتج أثناء تجربة البرنامج، أصبحت أقل فاعلية بكثير في ظل التعقيدات الحالية في تداخلات البرمجيات. ومما زاد الأمر صعوبة، أن نظم المعلومات القائمة على البرمجيات أصبحت متداخلة ومتصلة معاً بطريقة غير مسبوقة، كما نجد أن تطبيقاتها تتصرف بتعقيدات كبيرة.

مشكلة البحث

يقضي المبرمجون أوقاتهم في تصحيح ومعالجة أخطاء البرامج القائمة أكثر من الوقت الذي يقضونه في كتابة برامج جديدة، وطبقاً لإحصائيات المعهد القومي الأمريكي للمعايير والتكنولوجيا فإن 80% من تكلفة تطوير البرمجيات لمشروع نظم معلومات تكون في تحديد ومعالجة عيوب البرامج.[2]

إن تطوير برمجيات مستقرة يتطلب برنامج ضمان جودة مهيكل من البدايات المبكرة لتصميم البرمجيات، ولأن المبرمج / المطور يميل لصالح المنتج وأيضاً يكون تحت ضغط تطوير المنتج ضمن المهل الزمنية المحددة، فإن المستهلك غالبا هو المختبر الحقيقي للبرمجيات الجديدة. لذلك واختصاراً للوقت والجهد وضمان للجودة، فإن تطوير طرق أفضل لتأمين جودة البرمجيات يصبح أمراً ضرورياً ومهماً.

أهمية البحث وأهدافه:

أهمية البحث

بحسب التقرير المقدم من جمعية صناعة البرمجيات والمعلومات للحكومة الأمريكية في عام 2014، فإن الطلب الوطني لمهندسي البرمجيات يزداد بنسبة 12% سنوياً، بينما عدد المهندسين المتاح يزداد نسبة 4% سنوياً فقط. حتى مع هذه النسبة 4% في إنتاجية المبرمجين الفردية، فإنه هناك زيادة 4% في الفجوة بين ما هو متاح وما هو مطلوب[3].

يوجد طريقتين لتضيق هذه الفجوة: زيادة الإنتاجية أو تخفيض الزمن الذي يتم صرفه على الصيانة، والذي يعد رقم واحد في فعاليات البرمجيات. بشكل تقريبي، تصرف من 60 إلى 80% من كلفة البرمجيات و90% من وقت المبرمجين على صيانة البرامج الموجودة. بشكل واضح، في المجال الصناعي تقدر تكلفة كتابة سطر واحد من الكود (عالمياً) حوالي 4$ إلى 6$، بينما صيانته عبر الزمن يمكن أن تكلف بحدود 400$.[4] ، وأيضاً فإن الوقت المصروف على صيانة البرامج القديمة يقلل من الزمن المتاح لخلق وإنشاء برامج جديدة. وبالتالي فإن الطريقة المقترحة في البحث للتحقق من جودة البرمجيات سوف تحسن من إنتاجية المبرمجين وتقلل الوقت المصروف على صيانة البرمجيات.

أهداف البحث

يهدف البحث إلى إدخال منهجية *Crisp-DM* (*CRoss Industry Standard Process for Data Mining*) إلى منهجية تطوير البرمجيات باستخدام النموذج الشلالي التقليدي وتقييم أثرها، على تحسين جودة البرمجيات وذلك من خلال التخلص من الأخطاء والثغرات البرمجية وجعل إمكانية اكتشاف هذه الأخطاء في مرحلة مبكرة قدر الامكان.

طرائق البحث ومواده

تمّ اتباع المنهج التجريبي *Experimental Method* للتحقق من فرضيات البحث، حيث تمّ تثبيت المتغيرات المستقلة، ودراسة القيم التي تمّ الحصول عليها للمتغيرات التابعة، كما تمّ اتباع المنهج الوصفي *Descriptive Method* من خلال استخدام التحليل الإحصائي، وذلك للتحقق من التحسينات التي تم إدخالها إلى النموذج المدروس.

بيئة وعينة البحث

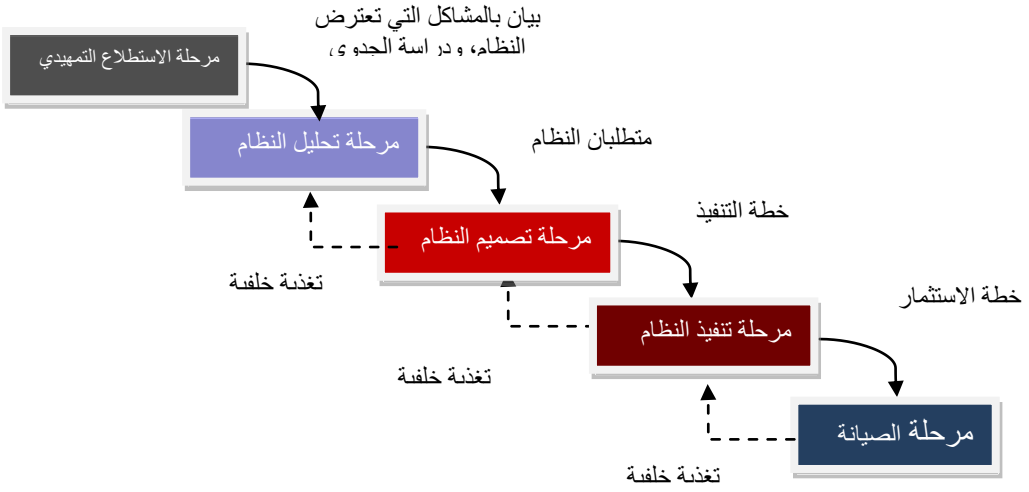
تم تجربة المنهجية الجديدة على مجموعة من الوحدات البرمجية المطلوب تنفيذها من قبل المبرمجين باستخدام منصة العمل الحر للمبرمجين *Freelance AppStartUp* من خلال تكليف مبرمجين بتحليل وتصميم وتنفيذ وظائف فرعية مختلفة، وتمت المقارنة بين الطريقة التقليدية والطريقة المطورة من خلال مطابقة التحسين المدخل على الوحدات ذات عدد المهام البرمجية المتطابقة (التوابع).

النموذج الشلالي

إن النموذج الأساسي المستخدم في البحث هو النموذج الشلالي والذي يتكون من خمسة مراحل متتالية مبينة في الشكل(1)، وهي:

- مرحلة الاستطلاع التمهيدي
- مرحلة تحليل النظام
- مرحلة تصميم النظام
- مرحلة تنفيذ النظام
- مرحلة الصيانة

يتميز النموذج الشلالي بالسهولة، فضلاً عن أن مخرجات كل مرحلة معروفة ومحددة تماماً، كما أن هذا النموذج يفترض انتهاء المرحلة السابقة قبل الانتقال إلى المرحلة الحالية[5]، وبالتالي نضمن الاطلاع بأفضل شكل على معطيات العمل خلال تقدمه، وهو ما يلائم عمليات الجدولة الزمنية الدقيقة، وعلى الرغم من وجود عدد من العيوب لهذا النظام، مثل التأخير في تسليم المنتج النهائي والحاجة للإحاطة بشكل تام بالمتطلبات، فإنه يتوافق مع المشاريع الاستراتيجية التي تتطلب دراسة دقيقة لكل مرحلة تشمل دراسة المدخلات والمخرجات ومدى تلبيتها للمطلوب قبل الانتقال إلى المرحلة التالية.

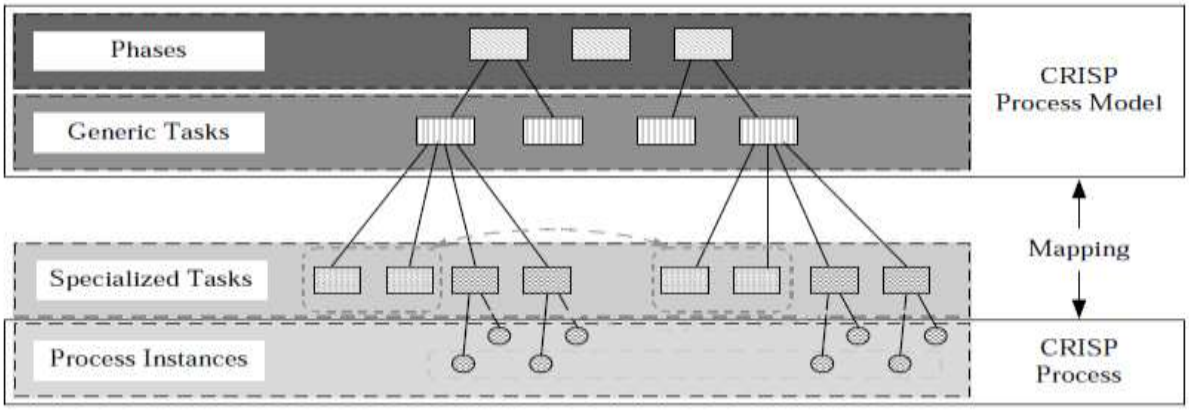


الشكل (1) مراحل النموذج الشلالي.

طريقة CRISP – DM:

هي عبارة عن منهجية لاستخلاص المعرفة من قواعد البيانات تم وضعها من قبل ائتلاف *CRISP – DM*، يقاد تطوير هذا المعيار من قبل *SPSS, Daimler-Chrysler and NCR* يتم تمويله من قبل الاتحاد الأوروبي لتطوير منهجية استخلاص البيانات بناء على التجارب العملية من مختلف أنحاء العالم.

تم توصيف منهجية استخلاص البيانات *CRISP-DM* على شكل نموذج لعملية هرمية *hierarchical process* تتكون من مجموعة من المهمات ومحددة في أربعة مستويات من التجريد (من العام إلى الخاص): الأطوار *phases*، المهمة العامة *generic task*، المهمة الخاصة *specialized task*، نموذج العملية *process instance*، والشكل (2) يوضح البنية الهرمية لمنهجية *CRISP-DM* [6].



الشكل (2) البنية الهرمية لمنهجية *CRISP – DM*.

إذاً، في المستوى الأول(مستوى القمة) يتم تنظيم عملية استخلاص البيانات في عدد من الأطوار، وكل طور يتكون من عدة مهام عامة من المستوى الثاني، بما يكفي لتغطية جميع الحالات المحتملة في استخلاص البيانات.

ويجب أن تكون مهمات عامة كاملة ومستقرة *complete and stable* قدر الإمكان، وتعني "كاملة" أنها تغطي كامل عملية استخلاص البيانات وكل احتمالات تطبيقات استخلاص البيانات، و"المستقرة" تعني أن النموذج يجب أن يكون صالحاً لإمكانية التطوير التي لم يتم لحظها في الوقت الحالي.

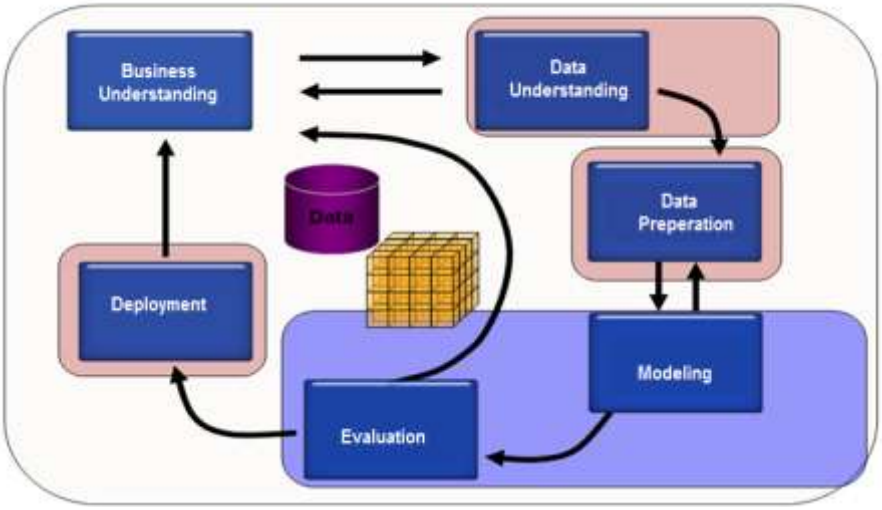
المستوى الثالث هو مستوى المهام الخاصة، وهو المستوى الذي يشرح كيفية إنجاز الأعمال في المهمات العامة لبعض الحالات الخاصة.

على سبيل المثال، نجد في المستوى الثاني مهمة عامة تدعى باسم "تنظيف البيانات" *data cleaning*، سيقوم المستوى الثالث بالاهتمام بالفروقات في تطبيق هذه المهمة في الحالات المختلفة مثل: تقنية تنظيف القيم العددية مقابل تنظيف القيم الفئوية. بينما يتضمن المستوى الرابع (نموذج العملية) تسجيل للعمليات والقرارات والنتائج لعملية استخلاص البيانات.

ترتيب نموذج العملية يكون حسب المهام المعرفة في المستويات الأعلى لإجرائية استخلاص البيانات، ولكنها تمثل ما يحدث فعلا على الواقع بدلا مما هو مفترض أن يحدث بشكل عام.

إن وصف الأطوار والمهام على شكل خطوات منفصلة يتم بترتيب محدد مكون من خطوات أمثلية من الأحداث، ولكن في الواقع العملي، فإنه من الممكن إنجاز العديد من المهام في تراتيب مختلفة، وسيكون من الضروري في معظم الأحيان العودة المتكررة إلى المهام السابقة وتكرار بعض الخطوات، وهذا النموذج من العمليات المبين في الشكل (3)

لا يحاول أن يسلك جميع المسارات الممكنة خلال عملية استخلاص البيانات، لأن هذا سوف يتطلب نموذجاً معقداً جداً للإجرائية[7].



الشكل (3) أطوار النموذج *CRISP –DM*.

تم اختيار النموذج *CRISP – DM* لاستخلاص المعرفة في البحث للأسباب التالية:

- النموذج شامل ويمكن مكاملة النماذج الأخرى ضمنه، وذلك لتغطية وجود نقص في بعض النواحي في توصيف الاجرائيات لهذا النموذج.
- النموذج مدعوم من قبل عدد من الشركات الرئيسية في مجال تطوير النظم وعدد من شركات البرمجيات الخاصة بتطوير أدوات استخلاص البيانات.
- النموذج موصف بشكل مفصل لكافة المراحل.
- النموذج يغطي مرحلة فهم العمل وفهم البيانات وهو أمر ضروري من أجل الاحاطة بالنظام ومدخلاته ومخرجاته بالشكل الأمثل.

تلبية الحاجة لبرامج SQA Meeting the need for SQA programs:

إن استخدام معايير التحقق من جودة البرمجيات *SQA* (Software Quality Assurance) يمكن أن يوفر الزمن اللازم لصيانة البرمجيات من خلال كشف وإلغاء احتمال حدوث مشاكل قبل مغادرة المنتج لمختبر التطوير. بالإضافة لذلك، يوفر وضع برنامج *SQA* في مكانية المناسب ضمن حلقة تطوير المنتج البرمجي الوقت المصروف لاحقاً في إجرائية صيانة البرامج التي تكون مكلفة ومستهلكة للوقت[8].

بشكل عام، لم يكن هناك فصل في وظيفة *SQA* في أقسام تطوير البرمجيات، وكان من الشائع بالنسبة للمبرمجين إنجاز تقييم جودة البرمجيات *QA* (Quality Assurance) الخاص بهم وتطوير أدوات الفحص الخاصة بهم. من الممكن، إن أي قسم تطوير يملك الكثير من العقود، سوف يقلل من أهمية وميزانية المصادر اللازمة من اجل الفحص ومن أجل *QA*.

بالإضافة لذلك، يعاني المبرمجين/المطورين من التغير في بيئة العمل، بسبب اعتيادهم على البرامج وكيف يجب أن تعمل، لذلك يكون من الصعب استنباط خيارات صحيحة لاكتشاف الأعطال غير المتوقعة. حتى المبرمجين الجيدين

جداً ليس بالضرورة أن يعلموا كيفية إجراء الفحص. بالنتيجة، يصبح المستهلك المستخدم غالبا المنتقد الأول والمكتشف الأول لأخطاء البرمجية، نظراً لكون البرمجيات تحت الحمل في البيئة الحقيقية.

حالما يتم توزيع البرنامج للعموم، يصبح من الصعب، إن لم يكن من الصعب، الحصول على مدة تمكن من تجاوز الاخطاء المكتشفة. لتجنب هذه الحالة، فإن مجموعات *SQA* يجب أن تكون المستخدم الأول، بعيداً عن كونها جزءاً من فريق التطوير. يسمح هذا بإنهاء تحيز المبرمج/الفاحص ويمنع التجاوزات الحرجة لمعايير الفحص بسبب التآلف مع المنتج. [9]

أكثر من ذلك، تحدد *SQA* معايير لعمل البرمجيات، وهي مجموعة من المعايير الشاملة تؤكد أن المنتج البرمجي سيكون قادراً على اجتياز تجارب فحص الزمن مع الزبائن الفعليين. لذلك فإن الاختبارات المستنبطة من قبل مجموعة *SQA* يجب أن تضع شروط صارمة أكثر من تلك التي يمكن أن يصادفها المستخدمين.

من بديهيات صناعة البرمجيات إن الكشف المبكر للتغيرات البرمجية هو أسهل لتجاوزها، وأقل تكلفة من الصيانة اللاحقة. لذلك من أول مهام مجموعة *SQA* توضيح أهمية الفحص المبكر لفريق التطوير.[10] مجموعة *SQA* يجب أن تعزز الإدراك لحالات الفحص وأن تتيح لأطقم الفحص الأدوات المناسبة (برامج الفحص المفردة تستخدم لفحص توابع جزئية أو وسطاء معنية لمنتج، بينما مجموعة برامج الفحص التي تفحص البرمجية بشكل كامل). مجموعة *SQA* يجب أن تنشئ قاعدة البيانات لحالات الفحص التي يمكن من خلالها قياس جودة المنتج. يجب أيضا أن تعطي مجموعة من أطقم الفحص والأدوات التي يمكن أن تستخدم من أجل التحليل الدلالي للمنتج البرمجي من خلال دورة حياته.

مناقشة النتائج

تم تطبيق المنهجية المطورة من خلال المراحل التالية والتي تم فيها الدمج بين النموذج الشلالي التقليدي والمفاهيم الخاصة بمنهجية *CRISP-DM*:

- الطور الأول: فهم النظام وتحليل احتياجات العمل ***Business Understanding***:

في هذه المرحلة تم التركيز على فهم أهداف المشروع والمتطلبات من وجهة نظر الأعمال، وتصميم خطة أولية لتحقيق هذه الأهداف، و وضع الجدول الزمني لهذه المرحلة. كما تم تحديد مصدر البيانات والمعلومات التي تم الحصول عليها من المستخدمين، مع أخذ بعين الاعتبار المتطلبات الأمنية ، طريقة تقديم المعلومات لهم، بالإضافة إلى تحديد متطلبات النظام ***check system requirements*** وتحديد العوامل والعناصر التي تؤثر على أداء وفعالية النظام.

تساهم هذه المرحلة في وضع الإطار الدقيق لمجال البيانات التي سوف تكون مدخلات النظام البرمجي.

- الطور الثاني: فهم البيانات ***Data Understanding***:

تبدأ مرحلة فهم البيانات بتجميع البيانات الأولية، ويلي ذلك تحديد الأنشطة التي تؤدي إلى التأقلم مع البيانات بهدف الوصول إلى المشاكل الخاصة بنوعية (جودة) البيانات واكتشاف الانطباعات الأولية عن تلك البيانات أو لاكتشاف المجموعات الفرعية التي تساعد في تشكيل الفرضيات الخاصة بالمعلومات المخفية.

تم في هذه المرحلة التعرف على طبيعة البيانات وبالتالي تم تحديد العمليات التي يمكن القيام بها على البيانات سواء لغرض استكشاف البيانات أو تلخيص البيانات أو تحديد آلية تمريرها إلى النظام البرمجي، يبين الجدول (1) مجموعة من العمليات التي تم تطبيقها على البيانات.

كما تم في هذه المرحلة تحديد الدقة *granularity* التي ستخزن بها هذه البيانات، فهل ستخزن المعلومات عن أنواع الطرق بشكل عام، أم ستخزن معلومات الطرق بشكل مفصل على سبيل المثال، وهذا يتطلب التعرف على الهيئة الحالية لتخزين البيانات والشكل المتوقع والمطلوب في المستقبل.

الجدول (1) أنواع السمات

نوع السمة		الوصف	أمثلة	العمليات
نوعية (نوعية)	أسمي	إن قيم السمة الاسمية تكون مجرد أسماء مختلفة، أي أن القيم الاسمية تعطي معلومات تكفي فقط لتميز كائن عن أخر $(\neq,=)$.	الرموز البريدية، أرقام تعريف الموظفين، لون العيون، الجنس.	المنوال، الانتروبية، التوافق، الارتباط، اختبار X^2.
	ترتيبي	تقدم قيم السمة الترتيبية معلومات كافية لترتيب الكائنات $(>،<)$.	قساوة المعادن. الدرجات، أرقام الشوارع.	الوسط، النسب المئوية، ارتباط الرتب *rank correlation* اختبارات التكرار *(run test)* اختبارات الإشارة.
رقمية (كمية)	مجال	من أجل سمات المجال، تكون الفروقات بين القيم ذات دلالة أي أن هناك وحدة قياس$(-،+)$.	تواريخ التقويم، الحرارة بالدرجات المئوية أو فهرنهايت.	المتوسط الانحراف المعياري، ارتباط *pearson*، اختباري T و F .
	نسبي	من أجل متحولات النسبة تكون كلا الفروقات والنسب ذات دلالة $(*،/)$.	الحرارة بالكلفن*(Kelvin)* المقادير المالية،العمر، الكتلة، الطول، التيار الكهربائي.	المتوسط الهندسي، المتوسط التوافقي، تباين النسبة المئوية.

- الطور الثالث: تحضير وتحميل بيانات اختبارية إلى قاعدة البيانات ***Data Preparation***:

تم في هذه المرحلة إنشاء مجموعة البيانات (*dataset*) النهائية (البيانات التي سوف تغذي أداة النمذجة) والتي تم أخذها من البيانات الخام الأولية، كما تم تحقيق تنظيف البيانات *data cleaning* والتحقق من تكامل البيانات ووضع البيانات بالصيغة التي يمكن التعامل معها من قبل النظام.

- الطور الرابع: النمذجة ***Modeling***:

ويشمل اختبار النماذج الخاصة باستخلاص البيانات واختبار بارامتراتها وتقييم النموذج المختار. في هذه المرحلة يتم اختيار تقنيات نمذجة مختلفة وتطبيقها ويتم ضبط بارامتراتها وفقاً للقيم المثالية. وبشكل عام هناك عدة تقنيات يمكن استخدامها لنفس النوع من مسألة استخلاص البيانات، البعض من هذه التقنيات لها متطلبات معينة من شكل البيانات، وبالتالي من الضروري وجود خطوة تتمثل في العودة إلى مرحلة تحضير البيانات في معظم الحالات.

- الطور الخامس: التقييم ***Evaluation***:

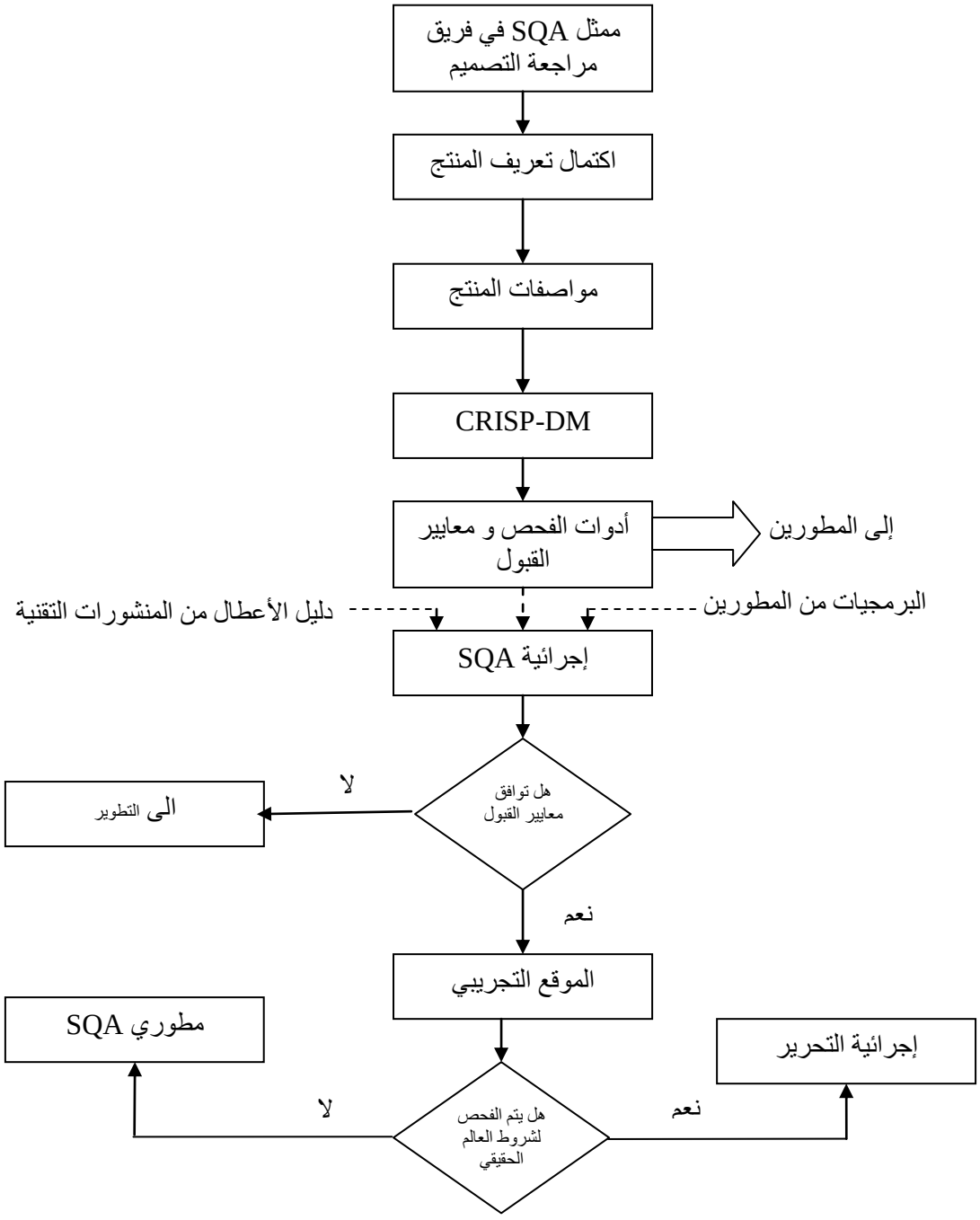
وفيه تم تقييم النتائج مع عملية مراجعة شاملة لكامل العملية واعتماد النماذج التي يبدو أنها ذات أداء عالي من وجهة نظر تحليل البيانات، ومراجعة الخطوات المنفذة أثناء إنشائه وذلك قبل الانتقال إلى التطوير النهائي للنموذج، وذلك للتأكد من أنه يحقق أهداف المشروع، وايضاً التأكد من أنه قد تم أخذ النقاط ذات العلاقة بالأعمال بعين الاعتبار بشكل كافي.

- الطور السادس: البناء النهائي ***Deployment***:

تم فيه تحديد خطة التطوير وخطة الصيانة وتعريف التقارير *define reports* حيث أن عملية وصول المستخدمين للبيانات تكون من خلال التقارير ولذلك يجب تحديد الخطوط العامة للتقارير واختبارها، إذ أنه وبشكل عام، لا تعتبر خطوة إنشاء النموذج هي المرحلة النهائية للمشروع حتى لو كان الهدف من النموذج هو زيادة معارفنا حول البيانات، لأن هذه المعلومات سوف تحتاج إلى تنظيم وتمثيل بطريقة ما بحيث يستطيع المستخدم الاستفادة منها، وحسب الحالة، فقد تكون مرحلة التطوير بسيطة، تتمثل في إنشاء تقرير، أو معقدة كما هو الحال في تحقيق عملية موثقة بشكل كامل لاستخلاص البيانات، ففي الكثير من الحالات سيقوم المستخدم وليس محلل البيانات بإنجاز خطوات التطوير، ولكن عندما يقوم المحلل بتنفيذ خطوات التطوير سيكون من المهم بالنسبة للزبون أن يفهم أولاً ماهي الأعمال التي يجب القيام بها لكي تتم الاستفادة من النماذج التي تم إنشاءها.

يبين الشكل (4)، الموقع المقترح لمنهجية *CRISP-DM* ضمن إجرائية التطوير. يتطلب تطوير البرمجيات المناسبة اشتراك مجموعات متعددة، تتضمن مهندسي برمجيات، تسويق، منشورات تقنية، مهندسي مكونات صلبة، و*SQA*. هذه المجموعات تشكل فريق مراجعة البرمجيات. مسؤوليات فريق *SQA* تتضمن أن نكون مدركين لحالة البرمجيات وزيادة أدراك المطورين لمتطلبات *SQA*.

في تعريف المنتج، ينصح فريق ضمان جودة المنتج *SQA* فريق التطوير بالحلول المقترحة للمشاكل المحتملة التي يمكن أن تحدث خلال الفحص وتعرف شروط القبول، وتتعلم آلية عمل المنتج بشكل يترافق مع التقدم نحو الفحص النهائي خلال مرحلة التطوير. حالما يحقق المنتج شروط *SQA* فإنه ينتقل إلى موقع التجريب، وإذا عبر المنتج البرمجي جميع الاختبارات، فإنه يتم نقله إلى موقع التطبيق.

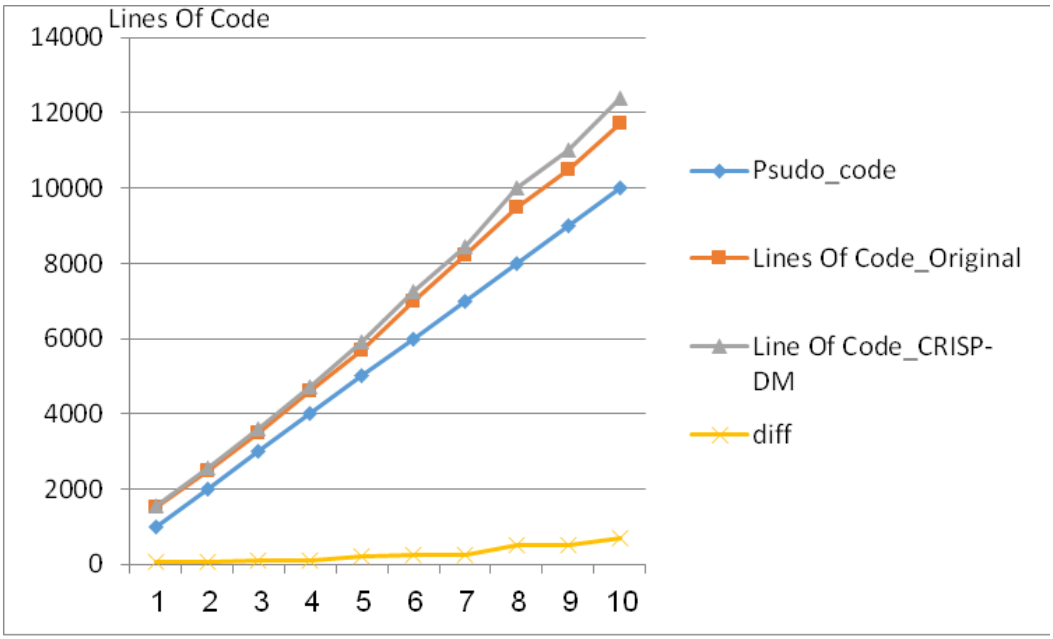


الشكل (4) الموقع المقترح لمنهجية *CRISP-DM* ضمن إجرائية التطوير.

يبين الجدول (2) عدد الأسطر التي يضيفها تطوير البرمجية باستخدام النموذج الشلالي قبل وبعد إدخال التطوير المقترح في البحث باستخدام *CRISP-DM*، وتبين النتائج الموجودة في الجدول والتي تم رسمها في الشكل (5) أن عدد الأسطر منخفض نسبياً مقارنة مع العدد الإجمالي لأسطر البرمجة، وتم التجريب من أجل عشر وحدات برمجية منفصلة.

الجدول (2) عدد الأسطر المضافة للكود باستخدام النموذج الشلالي قبل وبعد إدخال منهجية *CRISP-DM*.

	Psudo_code	Lines Of Code_Original	Line Of Code_CRISP-DM	diff
1	1000	1500	1550	50
2	2000	2500	2550	50
3	3000	3500	3600	100
4	4000	4600	4700	100
5	5000	5700	5900	200
6	6000	7000	7250	250
7	7000	8200	8450	250
8	8000	9500	10000	500
9	9000	10500	11000	500
10	10000	11700	12400	700

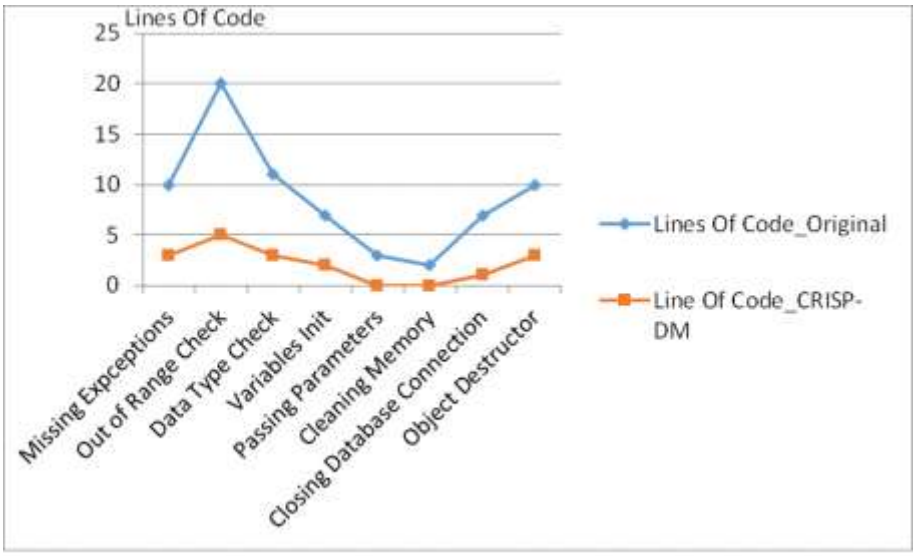


الشكل (5) مقارنة عدد الأسطر التي يضيفها تطبيق المنهجية المطورة إلى الكود الأصلي للبرمجية.

تم تقييم الأخطاء التي تم اكتشافها حسب نوع الخطأ لدى تطوير البرمجيات باستخدام النموذج الشلالي الأساسي والمطور باستخدام *CRISP-DM* وبينت النتائج التي تم عرضها في الجدول (3) والمبينة في الشكل (6) انخفاض عدد الأخطاء باستخدام المنهجية المطورة من أجل جميع أنواع الأخطاء التي تم تدقيقها في البحث.

الجدول (3) عدد الأخطاء التي تم اكتشافها باستخدام النموذج الشلالي قبل وبعد إدخال منهجية CRISP-DM حسب نوع الخطأ.

	Errors Number using Original Method	Errors Number using Developed Method
Missing Exceptions	10	3
Out of Range Check	20	5
Data Type Check	11	3
Variables Init	7	2
Passing Parameters	3	0
Cleaning Memory	2	0
Closing Database Connection	7	1
Object Destructor	10	3

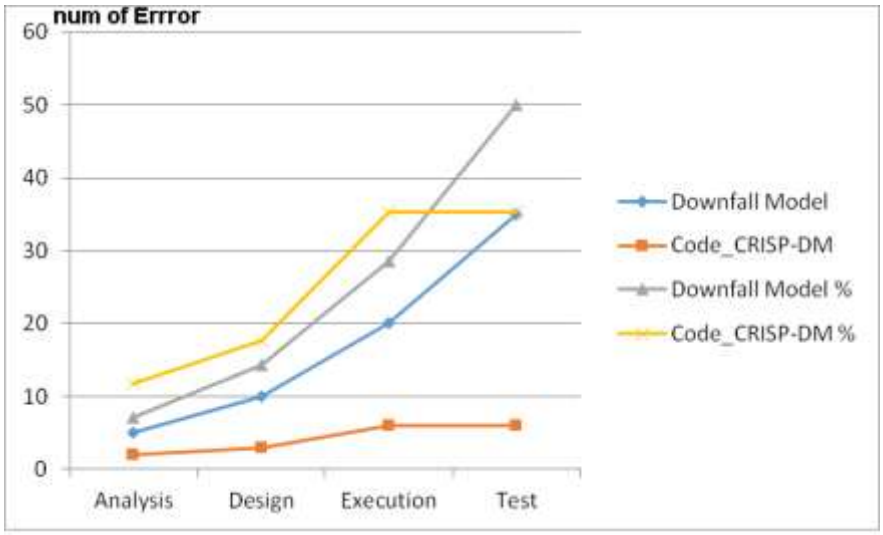


الشكل (6) عدد الأخطاء في المنهجية الأساسية والمطورة حسب نوع الخطأ.

تم دراسة نسبة الأخطاء المكتشفة في البرمجية حسب مرحلة تطوير البرمجية باستخدام النموذج الشلالي الأساسي والمطور، وبينت الأرقام ارتفاع نسبة الأخطاء التي تم اكتشافها وتصحيحها في المراحل المبكرة من تطوير البرمجة كما هو مبين في الجدول (4) والشكل (7).

الجدول (4) النسبة المئوية للأخطاء المكتشفة باستخدام النموذج الشلالي
قبل وبعد إدخال منهجية CRISP-DM في كل مرحلة من مراحل تطوير البرمجية.

	Downfall Model	Code_CRISP-DM	Downfall Model %	Code_CRISP-DM %
Analysis	5	2	7.142857143	11.76470588
Design	10	3	14.28571429	17.64705882
Execution	20	6	28.57142857	35.29411765
Test	35	6	50	35.29411765



الشكل (7)عدد الأخطاء المكتشفة حسب مرحلة دورة حياة البرمجية باستخدام المنهجية الأساسية والمطورة

الاستنتاجات والتوصيات:

الاستنتاجات

- الزيادة في عدد الأسطر البرمجية نتيجة تطبيق منهجية *CRISP-DM* تعتبر منخفضة نسبيا مقارنة مع عدد الأسطر الأساسي للبرمجية، حيث كانت النسبة بحدود *3.6%* وسطياً من إجمالي عدد الأسطر الأساسي للبرمجية.
- انخفاض ملحوظ في عدد الأخطاء التي تم اكتشافها باستخدام منهجية *CRISP – DM* حيث بلغ الانخفاض *25%* زيادة في عدد الأخطاء التي تم اكتشافها مقارنة مع المنهجية الأساسية.
- إرتفاع في نسبة اكتشاف الأخطاء في وقت مبكر من دورة حياة البرمجية باستخدام منهجية *Crisp-DM*، إذ نلاحظ أن نسبة اكتشاف الخطأ في المرحلة الأولى من المنهجية المطورة إلى 60% مقارنة مع الخوارزمية الأساسية.

التوصيات

- استخدام منهجية *CRISP-DM* في تطوير النماذج التقليدية لتطوير البرمجيات بما يحسن من جودة البرمجيات المنتجة.
- إدخال فعاليات تطوير أخرى إلى منهجية *CRISP-DM* مثل منهجية *SEMMA(Sampling - Explore - Modify - Model – Asses).*
- دراسات أثر إدخال تقنية *CRISP-DM* إلى نماذج مختلفة من نماذج تطوير البرمجيات مثل النموذج الحلزوني والنموذج التكراري المتزايد.

المراجع

[1] Awni Hammouri, Mustafa Hammad, Mohammad Alnabhan, Fatima Alsarayrah, Software Bug Prediction using Machine Learning Approach,(IJACSA) International Journal of Advanced Computer Science and Applications,Vol. 9, No. 2, 2018

[2] Nexhati Alija, Justification of Software Maintenance Costs, International Journal of Advanced Research in Computer Science and Software Engineering, Volume 7, Issue 3,March 2017

[3] Robert J. Shapiro of Sonecon, The U.S. Software Industry: An Engine for Economic Growth and Employment, , The U.S. Software Industry: An Engine for Economic Growth and Employment, SIIA White Pape,2014

[4] Sams, Bradley J., Calhoun, Deriving the Cost of Software Maintenance for Software Intensive Systems, The NPS Institutional Archive DSpace Repository ,2011

[5] Ahmed Mateen, Muhammad Azeem Akbar, Mohammad Shafiq, AZ Model for Software Development, International Journal of Computer Applications (0975 –8887)Volume 151 –No.6, October 2016

[6] Olegas NIAKŠU, CRISP Data Mining Methodology Extension for Medical Domain , Baltic J. Modern Computing, Vol. 3 (2015), No. 2, 92-109

[7] Daniel Adomako Asamoah, Ramesh Sharda, Adapting CRISP-DM Process for Social Network Analytics: Application to Healthcare , Twenty-first Americas Conference on Information Systems, Puerto Rico, 2015

[8] Päivi Williams, Future Trends and Development Methods in Software Quality Assurance, Master's Thesis Degree Programme in Information Systems Management 2017

[9] MD.SHAHADAT HOSSAIN,CHALLENGES OF SOFTWARE QUALITY ASSURANCE AND TESTING , International Journal of Software Engineering and Computer Systems (IJSECS) ISSN: 2289-8522, Volume 4 Issue 1, pp. 133-144, February 2018

[10] Tim Menzies, William Nichols, Forrest Shull, Lucas Layman, Are Delayed Issues Harder to Resolve? Revisiting Cost-to-Fix of Defects throughout the Lifecycle, arXiv:1609.04886v1 [cs.SE] 16 Sep 2016