مجلة جامعة طرطوس للبحوث والدراسات العلمية _ سلسلة العلوم الهندسية المجلد (6) العدد(12) 2022
Tartous University Journal for Research and Scientific Studies - engineering Sciences Series Vol. (6) No. (12) 2022

تحسين فاعلية مقاييس جودة البرمجيات

د . بسيم صالح برهوم*

(تاريخ الإيداع 2022/8/16 . قُبِل للنشر في 2022/12/21)

□ ملخّص□

في ظل الاستخدام المكثف للتقانات الرقمية ولاسيما في عصر التحول الرقمي الذي يضمن تقديم الخدمات بسهولة وبطريقة تزيد من قدرة المنتجات عـلى المنافسة بفعالية عالية، من هنا تظهر أهمية أن تكون هذه البرمجيات على مستوى عالي من التصميم والبناء وفق مبادىء هندسة البرمجيات المعتمدة والتي تضمن استمرار عمل هذه البرمجيات في مختلف الحالات وهذا يتطلب اختبارها بشكل يضمن استمرارها ويضمن صيانتها و تحسينها وتطويرها بشكل يلبي تطور احتياجات النمو في البيئة والاستخدام والفاعلية ، ولاسيما مع التشابك الكبير بين الأنواع المختلفة للأنظمة الرقمية المستخدمة الذي يفرض تخفيض تعقيد هذه البرمجيات بشكل يسمح بعمليات الصيانة والتطوير اللازمة.

من خلال ما تقدم تظهر أهمية هذا البحث الذي يقوم بتقييم بعض مقاييس اختبار البرمجيات ، ويقترح تحسين في البنية الحسابية لواحد من أهم هذه المقاييس وهو قياس طول البرنامج اعتمادا على صيغة جديدة تشمل طول البرنامج الناتج عن هالستد Halstead's Metric وطول البرنامج الناتج عن Lines Of Code (LOC)، ومن ثم القيام بحساب قيم بقية المقاييس الأخرى اعتمادا على نتائج التعديل المقترحة ومقارنة النتائج مع النتائج التجريبية (طول وزمن كتابة نفس البرنامج) التي حصلنا عليها من جلسة الإختبار مع مجموعة من المبرمجين الأكفاء وبشكل منفصل وحساب قيم المقاييس الأخرى اعتمادا على هذه النتائج التجريبية

الكلمات المفتاحية : software metric ، software quality ،اختبار البرمجيات ، جودة البرمجيات ، مقاييس هالستد ، حساب طول البرنامج ، التعقيد الحلقي Cyclomatic complexity .

* عضو هيئة تدريس في قسم البرمجيات ونظم المعلومات – كلية الهندسة المعلوماتية – جامعة تشرين –سوريا.

مجلة جامعة طرطوس للبحوث والدراسات العلمية _ سلسلة العلوم الهندسية المجلد (6) العدد(12) 2022
Tartous University Journal for Research and Scientific Studies - engineering Sciences Series Vol. (6) No. (12) 2022

Improving the effectiveness of software quality measures

Dr. Baseem Saleh Barhoum*

(Received 16/8/ 2022 . Accepted 12/12/ 2022)

□ ABSTRACT

In the light of the extensive use of digital technologies, especially in the era of digital transformation, which ensures the provision of services easily and in a way that increases the effectiveness of products in competition with high efficiency, so the importance of this software being at a high level of design and construction according to the principles of approved software engineering, which ensures the continuity of the work of these Software in various cases, and this requires testing it in a way that ensures its continuity its maintenance, , its improvement and its development in a way that meets the development needs of growth in the environment, using and effectiveness, especially with the great intertwining between the different types of used digital systems, which forces the complexity of these software to be reduced in a way that allows the necessary maintenance and development operations.

Through the foregoing, the importance of this research, which evaluates some software testing standards, and presents a proposed improvement in the computational structure of one of the most important of these measures, is the measurement of program length depending on a new formula that includes the program length produced by Halstead's Metric and the program length which resulte from Lines Of Code (LOC), and then calculating the values of the rest of the other measures based on the proposed modification results and comparing the results with the experimental results for the length and time of the same program that we obtained from the test session with a group of qualified programmers separately and calculating the values of other measures based on these experimental results

Keywords: software metric, software quality, software testing, software quality, Halsted metrics, program length calculation, cyclomatic complexity.

* Lecturer – Department of Software and Information Systems – Faculty of Informatics engineering – Tishreen University – Syria.

1. مقدمة

تنطوي ظاهرة الجودة على عدد من المتغيرات التي تعتمد على السلوك البشري ولا يمكن السيطرة عليها بسهولة، ولكن قد تقوم المقاييس بقياس وتقدير هذا النوع من المتغيرات التي تحتاج إلى مصادقة نظرية، ومصادقة تجريبية، وهذا يحتاج إلى عدد كبير من إجرائيات الاختبار.

"لا يمكننا تحسين ما لا يمكننا قياسه" يساعدنا اختبار المقاييس على فعل الشيء نفسه تمامًا.."

تعمل مقاييس اختبار البرمجيات على تحسين كفاءة وفعالية عملية اختبار البرمجيات عموما، وتمثل مؤشر كمي لمدى أو أبعاد أو حجم بعض سمات عملية أو منتج ما. مثل قياس إجمالي عدد العيوب، حيث يمكننا قياس بعض خصائص البرامج أو العمليات بدقة، ويمكننا من خلال الكود الثابت قياس وتقدير عدد الأخطاء المتبقية في الكود وقياس التكلفة والمدة و الإنتاجية وقابلية الصيانة، ويمكننا من خلال التنفيذ تقدير أوقات الفشل المستقبلية.

قمنا في هذا البحث بتحليل بعض مقاييس البرامج الموجودة والتحقق منها عمليا، و تقديم اقتراح لتحسين عملية حساب طول البرنامج، واستثمار ذلك في تحسين قيم مقاييس **Halstead** .

2. مشكلة البحث

تعتبر مسألة اختبار البرمجيات Software Testing ، وضمان صيانتها Maintenance ، وإعادة استخدامها Reuse من أهم المسائل التي تواجه عمليات تصميم وبناء الأنظمة البرمجية والتي يخصص لها أعلى الميزانيات ولاسيما في المستويات الحرجة منها ، وتعتمد هذه الاختبارات على مقاييس محددة مثل طول البرنامج ، والتعقيد الحلقي Cycloramic complexity للبرنامج ، والزمن اللازم لتحويل خوارزمية ما إلى كود برمجي قابل للتنفيذ، والجهد اللازم لذلك، تتلخص مشكلة البحث الأساسية في حجم التناقضات في قيم هذه القياسات والمرتبطة بشكل أساسي بطريقة حساب طول الكود البرمجي، حيث قمنا في هذا البحث باقتراح طريقة جديدة لحساب طول البرنامج ، ومن ثم تحسبن قيم هذه القياسات وصولا إلى نتائج متقاربة مع نتائج التجارب العملية التي قمنا بها.

3. أهداف البحث

الهدف العام من تتبع وتحليل مقاييس البرامج هو تحديد جودة المنتج Product quality، أو العملية الحالية، وتحسينها ، والتنبؤ بالجودة بمجرد اكتمال مشروع تطوير البرمجيات، وعلى مستوى أكثر دقة. يهدف هذا البحث بشكل أساسي إلى تحسين قيم مقاييس هالستيد من خلال:

1- دراسة المقاييس المختلفة لتحليل سمات ومعايير البرامج، ثم تحليل مقاييس هالستيد .

2- اقتراح تعديل على آلية حساب طول البرنامج، وإعادة حساب قيم المقاييس الأخرى اعتمادا على ذلك.

3- القيام بتجربة عملية لقياس زمن وطول برنامج محدد من خلال مجموعة من المبرمجين وبلغات مختلفة.

4- إجراء تحليل مقارن لقيم المقاييس اعتماداً على المنهجية الكلاسيكية المعروفة في حساب طول البرنامج، وبين قيم المقاييس اعتماداً على المنهجية المقترحة، وبين النتائج العملية التجريبية، وصولاً إلى تحسين قيم مقاييس هالستيد.

4. سيناريو العمل :

الخطوة الأولى: سنبدأ بدراسة مختصرة لكل من مقاييس Thomas McCabe مثل(Lines of Code (LOC و Cyclomatic Complexity (CC) ، و مقاييس Halstead ولاسيما Halstead Volume (HV) للبرنامج والتي تعتمد على عدة قياسات تعتمد بدورها على طول البرنامج N [1].

الخطوة الثانية: حساب قيم أهم المقاييس المستخدمة في اختبار البرمجيات وفق طول Halstead لبرنامج ما، ووفق طول McCabe وذلك من أجل كود برمجي بسيط، ومن ثم مقارنة النتائج واستنتاج الاختلافات والتناقضات الكبيرة بين قيم تلك المقاييس.

الخطوة الثالثة: إجراء تجربة عملية لحساب كل من الزمن الحقيقي اللازم لكتابة نفس البرنامج المحدد في الخطوة الثانية، وحساب طوله، وبعدة لغات برمجية (C++ , Java , C# , Python)، ومن ثم حساب قيم بقية المقاييس الأخرى اعتماداً على النتائج التجريبية .

الخطوة الرابعة: مقارنة قيم المقاييس الناتجة عن القييم التجريبية (طول البرنامج بلغة C++، والزمن اللازم) مع قيم المقاييس الناتجة عن طول هالستيد N ، وطول ماكابي LOC.

الخطوة الخامسة: إجراء تعديل على طريقة حساب طول البرنامج ومن ثم حساب قيم المقاييس الأخرى اعتماداً على الطول الناتج عن التعديل، ومقارنة النتائج مع كل من قيم مقاييس هالستد الناتجة عن القيم التجريبية ، وقيم مقاييس هالستد الناتجة قبل تطبيق التعديل المقترح.

5. مقاييس جودة البرمجيات

تُعبر مقاييس جودة البرامج عن وحدات كمية تستخدم للإشارة إلى خصائص البرنامج (قابلية الصيانة، والتعقيد، وقابلية الاستخدام، والمرونة، والمهام الوظيفية، والموثوقية، والكفاءة، وقابلية النقل، ....) ، و يتم تقسيم هذه المقاييس إلى :

1. مقاييس المنتج Product metrics :

تصف هذه المقاييس خصائص البرنامج مثل: حجم البرنامج، تعقيد البرنامج، مستوى الجودة، ومستوى الصعوبة ،[1][2].

2. مقاييس العملية Process metrics أومقاييس التحكم المرتبطة بشكل عام بعملية البرمجيات

يعبّر عن العملية أو الإجرائية Process في سياق هندسة البرمجيات Software Engineering على أنها مجموعة من النشاطات activities والطرق التي يتم تنفيذها من أجل تطوير المنتج البرمجي، وتتمتع الاجرائية بإمكانية تكرارها في مشاريع أخرى، ويتم تقييمها من خلال مقاييس الكلفة والجودة، و تُستخدم لتحسين كفاءة العمليات خلال دورة حياة تطوير البرمجيات(Software Development Life Cycle (SDLC، ويتم وصف مقاييس عملية قابلية الصيانة الأكثر شهرة في تحديد جودة البرنامج ، والتي تمثل مجموعة من السمات التي تؤثر على الجهد المطلوب لإجراء تعديلات محددة ويمكن تعريف قابلية الصيانة على أنها سهولة تعديل نظام أو مكون برمجي لتصحيح الأخطاء أو تحسين الأداء أو السمات الأخرى أو التكيف مع البيئة المتغيرة. بالنتيجة، إن صيانة البرامج تعبر عن الدرجة التي يمكن بها إجراء التغييرات والتعديلات على نظام البرنامج، وترتبط ارتباطًا وثيقًا بسهولة أداء صيانة النظام [2].

يعتبر مؤشر الصيانة Maintenance Index (MI) واحد من أهم الطرق المستخدمة لتحديد قابلية الصيانة للبرامج. ويتكون هذا المؤشر `(MI) من مزيج من المقاييس مثل Lines of Code (LOC) ، Cyclomatic Complexity (CC) ، و Halstead Volume (HV) [4] [3].

3. مقاييس المشروع Project metrics :

تصف خصائص المشروع وتنفيذه مثل: عدد المطورين، نمط التوظيف على مدى دورة حياة البرنامج والتكلفة [8].

5-1 مقياس مكابي

يُعتبر طول الكود Lines of Code (LOC) أو KLOC من أوائل المقاييس الرئيسية التي استُخدمت لقياس الإنتاجية البرمجية والجهد اللازم للتعبير عن الخوارزمية بشكل كود تنفيذي، حيث يحدد مقياس مكابي تعقيد الكود بناءً على عدد مسارات التحكم، و تم إنشاؤه بواسطة الرسم البياني Graph المُعبر عن الكود ويستند إلى العلاقات الرياضية بين عدد المتغيرات، ونوع عبارات اللغة البرمجية المستخدمة. وفقًا لـ [2]

في الوقت الحاضر ، يتم تنفيذ مقياس LOC في العديد من أدوات القياس المستخدمة ، والتي يمكن أن تستخدم لحساب المقاييس الأخرى مثل التعقيد الحلقي Cyclomatic Complexity الذي يُشتق ويُحسب من الرسم البياني للبرنامج المكون من مجموعة من العقد ، ومجموعة من الاضلاع التي تصل بين هذه العقد فمثلا البيان G = (V, E) يتكون من مجموعة منتهية من العقد V ومجموعة من الاضلاع تحوي أزواج من العقد المرتبة أو غير المرتبة على الشكل : $E=(e_1 , e_2 , \ldots, e_k)$, $V=(n_1 , n_2 , \ldots, n_m)$، نسمي تتالي أزواج متجاورة من الاضلاع مسار، و إذا كان المسار يحوي على الاقل عقدة جديدة أو ضلع جديد من عقدة البداية إلى عقدة النهاية نسميه مسار مستقل Independent path ، اتجاه الاضلاع أو ما يسمى مسارات البيانات الموجهة paths in a directed graph يحدد طبيعة البيان (موجه أو غير موجه) ، كما يقدم عدد هذه الأضلاع والعقد في البيان الموجه المعلومات الأساسية اللازمة لحساب التعقيد الحلقي ، وبالتالي تحديد عدد حالات الاختبار Test Cases الأساسية المطلوب تنفيذها على الكود البرمجي المعبر عنه ببيان موجه وذلك من خلال صيغة حساب التعقيد الحلقي (CC) الآتية : $V(G) = e - v + 2p$ ، حيث تمثل e عدد الأضلاع ، و V عدد العقد ، و P عدد الأجزاء في البيان ، وقيمة التعقيد الحلقي التي نحصل عليها تعبر عن عدد حالات الاختبار الأساسية ، إضافة إلى ذلك فإن قيمة التعقيد الحلقي تؤشر إلى إمكانية إعادة كتابة الكود البرمجي بشكل قياسي مما يؤدي إلى تخفيض قيمة التعقيد الحلقي وذلك من خلال المحافظة على عمق محدد من التداخلات في الحلقات البرمجية (عمق أقل من 5)، ومن خلال استبدال مجموعة من الشروط المتداخلة (عبارات if) بالصيغة المناسبة من select، وهذا ينعكس على عدد حالات الاختبار Test Cases ، وعلى قابلية الكود للصيانة والتطوير [3][2].

أسطر الكود (LOC) Lines of Code: هو مقياس برمجي يستخدم لقياس حجم برنامج الكمبيوتر عن طريق حساب عدد الأسطر في البرنامج والتي تحوي الكود المصدري الفعلي، و يستخدم لتقدير مقدار الصيانة المطلوبة ويمكن استخدامه لحساب عدد من قيم المقاييس الاخرى للبرامج [6][11].

5-2 التعقيد الحلقي Cyclomatic Complexity (CC) :

في عام1976 أوجد Thomas McCabe مفهوم التعقيد الحلقي الذي يقيس عدد المسارات المستقلة خطيًا من خلال بيان البرنامج Control Flow (CF)، ويعتبر تعقيد مكابي أحد أكثر مقاييس البرامج المقبولة والمستخدمة على نطاق واسع، والذي يعمل بشكل مستقل عن تنسيق اللغة البرمجية المستخدمة [4][3] .

ويمكن تطبيق مقياس التعقيد السيكلومي في مجالات عدة من أهمها:

1–تحليل مخاطر تطوير الكود، والذي يتفحص الأكواد قيد التطوير لتقييم المخاطر الكامنة أو تراكم المخاطر.

2–تغيير تحليل المخاطر في الصيانة، حيث يؤشر تعقيد الكود إلى زيادة المخاطر في الصيانة [5].

3–تخطيط الاختبار، حيث أظهر التحليل الرياضي أن التعقيد السيكلومي يعطي العدد الدقيق للاختبارات الأساسية اللازمة لاختبار نقاط القرار في البرنامج.

3-5 مقاييس هالستد Halstead Metrics

مقاييس برمجية قدمها موريس هوارد هالستيد في عام 1977 كجزء من بحث حول إنشاء علم تجريبي لتطوير البرمجيات، حيث لاحظ هالستيد أن مقاييس البرنامج يجب أن تعكس التنفيذ أو التعبير عن الخوارزميات بلغات مختلفة، ولكن يجب أن تكون مستقلة عن تنفيذ هذه الخوارزميات على منصة معينة [1].

و يتكون برنامج الكمبيوتر وفق رؤيته من سلسلة من الرموز المميزة التي يمكن تصنيفها على الشكل الآتي :

- **عوامل التشغيل Operators:** أي رمز أو كلمة مفتاحية في البرنامج تحدد عملاً حسابياً، معظم علامات الترقيم. ويتضمن أيضًا بعض الكلمات الأساسية غير القياسية الخاصة بالمترجم
- **المعاملات operands** : هي جزء من تعليمات الكمبيوتر التي تحدد البيانات التي سيتم معالجتها (variables and constants) ، و تعتمد مقاييس Halstead على تفسير شيفرة كود البرنامج على أنها سلسلة من الرموز وتصنيف كل رمز ليكون operators عاملًا أو operands معاملًا ، وتتم الإشارة لعدد العوامل **operators** الفريدة بـ n1 ، ولعدد المعاملات **Operands** الفريدة بـ n2 ، وللعدد الإجمالي للعوامل بـ N1 ، و للعدد الإجمالي للمعاملات بـ N2 ، و يتم اشتقاق مقاييس هالستيد الأخرى من هذه الكميات الأربع على الشكل الآتي:
- **طول البرنامج (N)** : هو مجموع العدد الإجمالي للعوامل operators والمعاملات operands في البرنامج، أي أن

$$N = N1 + N2$$

- **حجم المفردات (n):** هو مجموع عدد العوامل والمعاملات الفريدة: unique operators and unique operands

$$n = n1 + n2$$

- **حجم البرنامج (V):** هو محتويات البرنامج من المعلومات ، ويمثل حجم المعلومات (بالبتات) اللازمة لتحديد البرنامج، و يتم حسابه من خلال ضرب طول البرنامج باللوغاريتم الثنائي لحجم المفردات (n) :

$$V = N \times \log_2(n)$$

- **مستوى الصعوبة (D):** يتناسب مستوى الصعوبة أو مؤشر الخطأ (D) للبرنامج مع عدد عوامل التشغيل الفريدة في البرنامج، و أيضًا مع النسبة بين العدد الإجمالي للمعاملات وعدد المعاملات الفريد unique operands

$$D = (n1 / 2) \times (N2 / n2)$$

1- **مستوى البرنامج (L)** : يعبر عن مستوى التجريد، حيث يكون البرنامج ذو المستوى المنخفض أكثر عرضة للأخطاء من البرامج عالية المستوى، ويعطى بالعلاقة :

$$L = 1 / D$$

2- **جهد التنفيذ (E):** يفسر على أنه مقدار من التمييز العقلي المطلوب لتنفيذ البرنامج، ويتناسب الجهد المبذول (E) لتنفيذ أو فهم برنامج ما مع حجم و مستوى صعوبة البرنامج باستخدام العلاقة التالية :

$$E = V \times D$$

و يُترجم مقياس الجهد هذا إلى الوقت اللازم لكتابة البرنامج.

3- **وقت التنفيذ (T):** يتناسب وقت تنفيذ أو فهم البرنامج (T) مع الجهد المبذول ، و يمكن استخدام التجارب لمعايرة هذه الكمية بالثانية :

$$T = E / 18 \text{ s}$$

4- **عدد الثغرات المكتشفة (B):** يرتبط عدد الأخطاء (B) التي يتم كشفها بالتعقيد الكلي للبرنامج. ويحسب من العلاقة $B = (E)^{2/3} / 3000$ ، و هو تقدير لعدد الأخطاء في التنفيذ [11][13].

أظهرت التجارب أنه عند البرمجة باستخدام C أو C++، يحتوي الملف المصدر على أخطاء أكثر مما يقترحه المقياس B. وينمو عدد العيوب بسرعة أكبر من قيمة B، ويتم التقريب إلى القيم الصحيحة

5- **معادلة الطول N:** تمثل طول برنامج منظم بشكل جيد، وتدل فقط على عدد المشغلين والمعاملات الفريدة

ويمكننا حساب الطول المتوقع للبرنامج من العلاقة الآتية:

$\tilde{N} = n1 * \log2(n1) + n2 * \log2(n2)$ ، ويسمى طول البرنامج المقدر المحسوب

يمكننا استخدام هذه المقاييس من أجل قياس الجودة الشاملة للبرامج، و التنبؤ بمعدل الخطأ، و توقع جهود الصيانة اللازمة، ويمكن استخدام هذه المقاييس مع أي لغة برمجية [9]، ولكن يمثل ذلك ثغرة عندما تكون أهداف القياس مرتبطة بلغة البرنامج.

نلاحظ أن مقاييس هالستيد تختلف عن مقاييس مكابي، لأن مقياس مكابي يحدد مدى تعقيد الكود بناءً على عدد مسارات التحكم ، ويتم إنشاؤه بواسطة الكود ويستند إلى العلاقات الرياضية بين عدد المتغيرات، ونوع عبارات لغة البرمجة [5] [1].

6- حساب المقاييس وفق طول Halstead، ووفق طول McCabe.

لنقوم الآن بحساب قيم أهم مقاييس الاختبارات المستخدمة اعتماداً على طول البرنامج المحسوب بصيغة هالستيد ، ومن خلال طول ماكابي ، وذلك لنفس البرنامج الذي سنعتمده في التجربة والمبين الشكل رقم (1) الآتي:

```
1    // writen in -2021
2   #include <iostream>
3   using namespace std ;
4   int main()
5    {
6   int n ;
7      // Enter Number of elements in arry
8    cout << "enter poditive number : " ;
9    cin >> n ;
10  int a[n] ;
11    // Enter  the n   numbers
12  for ( int i=0 ; i<n ;i++ )
13    cin >> a[i] ;
14  int smallest= a[0] ;
15
16  for( int i=1 ; i<n ; i++ )
17    if ( a[i]< smallest )
18      smallest= a[i] ;
19     // Print of the smallest number
20
21    cout << " smallest is : ";
22    cout << smallest ;
23    return 0 ;
24  }
```

الشكل رقم (1) الكود البرمجي

بشكل عام يعتمد تعقيد هالستيد Halstead Software Complexity على قيم كل من
n1 , n2 , N1 , N2 ، والتي يُعبر عنها بتابع تعقيد هالستيد الاتي :

HC = f (n1,n2, N1 , N2) [7] [10]

نبين بداية عدد unique operators للبرنامج الوارد في الشكل رقم (1) في الجدول رقم (1) الآتي:

الجدول رقم (1)

main	int	cout	cin	for	if	return	( )	{ }	[ ]	=	<	<<	>>	++	;	std	using	namespace	**N1**
1	6	3	2	2	1	1	4	1	5	4	3	3	2	2	15	1	1	1	**58**

ونبين عدد المعاملات **unique operands** للبرنامج السابق في الجدول رقم (2) الآتي :

الجدول رقم (2)

n	a	i	smallest	0	1	" "	N2
5	5	9	4	3	1	2	29

فيكون طول البرنامج وفق صيغة هالستيد كما هو موضح بالجدول رقم (3) الآتي:

الجدول رقم (3)

	Unique	**Total**
operators	n1=19	N1=58
operands	n2=7	N2=29
Length	**N= N1 + N2 = 87**	

نحسب قيم المقاييس المعتمدة في البحث (الحجم Volume، و الصعوبة Difficulty ، و الجهد Effort ، و الزمن Time ، و عدد الثغرات المتوقع Bugs):

1- وفق طول هالستيد :

$$Volume: V = N \times \log_2 n = 87 \times \log_2 26 = 87 \times \frac{\ln 26}{\ln 2} = 87 \times 4.7$$

$= 408.9$ حجم البرنامج

$$Difficulty: D = \frac{n1}{2} \times \frac{N2}{n2} = \frac{19}{2} \times \frac{29}{7} = 39.35$$ درجة الصعوبة

$$Effort: E = D \times V = 39.35 \times 408.9 = 16090.215$$ الجهد اللازم

$$Time: T = \frac{E}{18} = \frac{16090.215}{18} = 893.900$$ الزمن المستغرق

$$Estimated\ Number\ of\ Bugs: B = \frac{V}{3000} = \frac{408.9}{3000} = 0.136$$

2- وفق طول ماكابي:

نقوم بحساب قييم هذه المقاييس اعتمادا على طول ماكابي الذي يأخذ بالحسبان فقط الأسطر التي تحوي كود فعلي ويهمل الأسطر البيضاء و أسطر التعليقات [12] [11]، وذلك بنفس الطريقة المتبعة مع مقاييس هالستيد، ولكن بالاعتماد على طريقة ماكابي في حساب طول البرنامج نحصل على النتائج الموضحة في الجدول رقم (4) الآتي ، مع مقارنة بين قيم المقاييس بناءً على كل من طول هالستد ، وطول ماكابي:

الجدول رقم (4)

	Length	Time	Volume	Difficulty	Effort	Bugs
قيم المقاييس اعتماداً على طول هالستيد	87	893.900	408.9	39.35	16090.215	0.136
قيم المقاييس اعتمادا على طول ماكابي	18	41.0988	18.8	39.35	739.78	0.0062

بدراسة القييم المسجلة في الجدول (4) ، نلاحظ وجود اختلافات كبيرة بين القيم التي اعتمدنا في حسابها على LOC والقيم التي اعتمدنا في حسابها على هالستد والسبب الأساسي لهذه الاختلافات هو طول البرنامج ، فمثلا من أجل طول هالستيد N=87 نلاحظ الجهد هو E=16090.215 ، والحجم V=409.9 ، و الأخطاء B=0.136 ، بينما من أجل طول ماكابي LOC= 18 ينخفض الجهد إلى E=739.78 ، و الحجم إلى V=18.8 ، والاخطاء إلى B=0.0062 ، مع ثبات درجة الصعوبة في كلتا الحالتين كونها مستقلة عن طول البرنامج وفق مقاييس هالستيد، وبالتالي يمكننا اعتماد مقاييس هالستيد عندما يكون الأمر مرتبط بالجهد والزمن وحجم العمل اللازم لتحويل الخوارزمية إلى برنامج تنفيذي ، وفي حالة الحديث عن التعقيد الحلقي والمسارات المستقلة التي تمثل حالات الاختبار الأساسية اللازمة فإن مقاربة ماكافي هي الأفضل ، ولكن في كلتا الحالتين يمكننا الاعتماد على نتائج الطريقة المقترحة بسبب قرب نتائجها من النتائج التجريبية العملية.

7- الدراسة التجريبية وحساب قيم مقاييس هالستيد:

تلعب اللغة البرمجية المستخدمة دورا هاما في عدد المعاملات المستخدمة، والتي تحدد طول البرنامج وبالتالي في الزمن اللازم لكتابته [13] .

قمنا بإجراء دراسة تجريبية على كتابة كود البرنامج السابق لإيجاد العدد الأصغر من بين مجموع أعداد مدخلة وذلك من خلال مجموعة من المبرمجين وبلغات برمجية مختلفة ومن ثم قمنا باعتماد متوسط الزمن ومتوسط الطول بالنسبة لعدد مرات التجربة مع عدد من المبرمجين، وقمنا بتسجيل النتائج في الجدول رقم (5) الآتي:

الجدول رقم (5)

Language	Time (min)	Program Lines
C++	4.7	24
C#	4.3	18
Java	5.6	17
Python	4.2	8

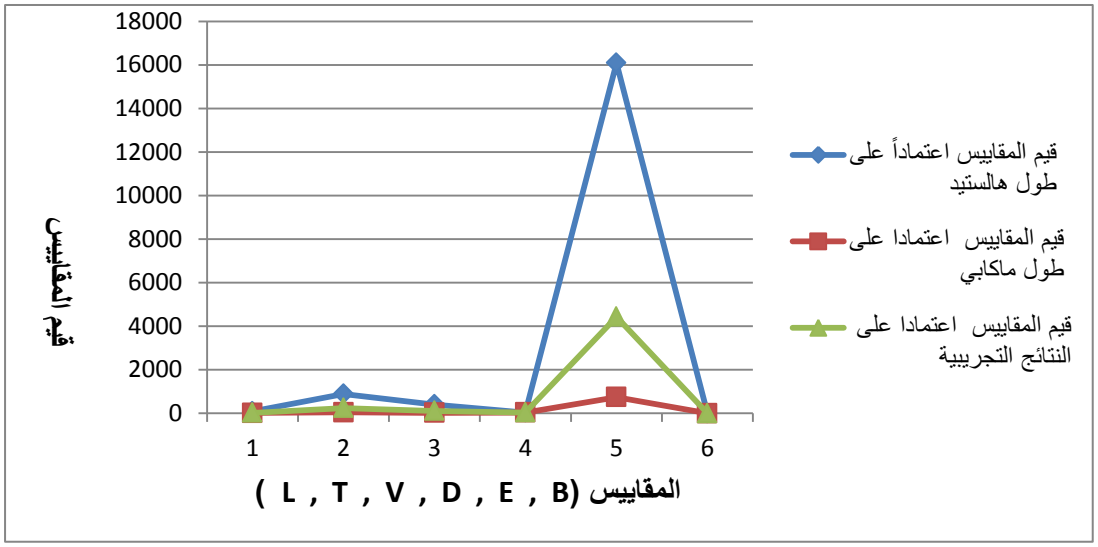
لنقوم الآن بحساب قيم مقاييس هالستيد اعتماداً على النتائج التجريبية بلغة C++ (الطول الحقيقي للبرنامج N=24 ، والزمن الحقيقي T=4.7) ، فنحصل على $V = 112.8$ ، و $D = 39.35$ ، و $E = 4438.68$ ، و $T = \frac{E}{18} = 246.59$، واخيراً $B = \frac{V}{3000} = 0.0376$

نقدم في الجدول رقم (6) الآتي مقارنة لقيم مقاييس هالستد اعتماداً على طول هالستد، وطول ماكابي ، والنتائج التجريبية

الجدول رقم (6)

	Length	Time	Volume	Difficulty	Effort	Bugs
قيم المقاييس اعتماداً على طول هالستيد	87	893.900	408.9	39.35	16090.215	0.136
قيم المقاييس اعتمادا على طول ماكابي	18	41.0988	18.8	39.35	739.78	0.0062
قيم المقاييس اعتمادا على النتائج التجريبية	24	246.59	112.8	39.35	4438.68	0.0376

المخطط في الشكل رقم (2) الآتي يوضح المقارنة بين قيم القياسات اعتمادا على الآليات الثلاث السابقة.



الشكل رقم (2) يوضح مقارنة بين القياسات الناتجة عن آليات حساب طول البرنامج

بدراسة القيم المسجلة في الجدول (6) نلاحظ الفجوة الكبيرة بين قيم المقاييس المحسوبة اعتماداً على النتائج التجريبية التي تعتمد المقاربتين معا بسبب تركيزها على العنصر البشري ومهارته وعلى اللغة وإمكانياتها مضافا عليهما بيئة العمل، وبين قيم المقاييس المحسوبة وفق طول هالستيد (لأنه يعتمد على اللغة البرمجية فقط)، ووفق طول ماكابي الذي يعتمد على الطريقة التي يكتب بها البرنامج وعدد العقد (نقاط القرار) التي يحويها ، وأن النتائج التجريبية أعطت قيم تقع بالمجمل بين قيم القياسات بالاعتماد على هالستيد (التشاؤمية)، وبين قيم القياسات بالاعتماد على ماكابي (التفاؤلية).

8- التحسين المقترح

طرأت تحسينات على جودة البرمجيات من خلال تحسين التكلفة بأشكالها المختلفة وبالاعتماد على طول البرنامج المحسوب وفق طريقة ماكابي ، أو وفق طريقة هالستيد [13]، ولكن في هذا التحسين المقترح نعتمد على طريقة جديدة في حساب طول البرنامج تقوم على دمج الطريقتين (طول البرنامج اعتمادا على كل من هالستيد ومكابي LOC) على الشكل :

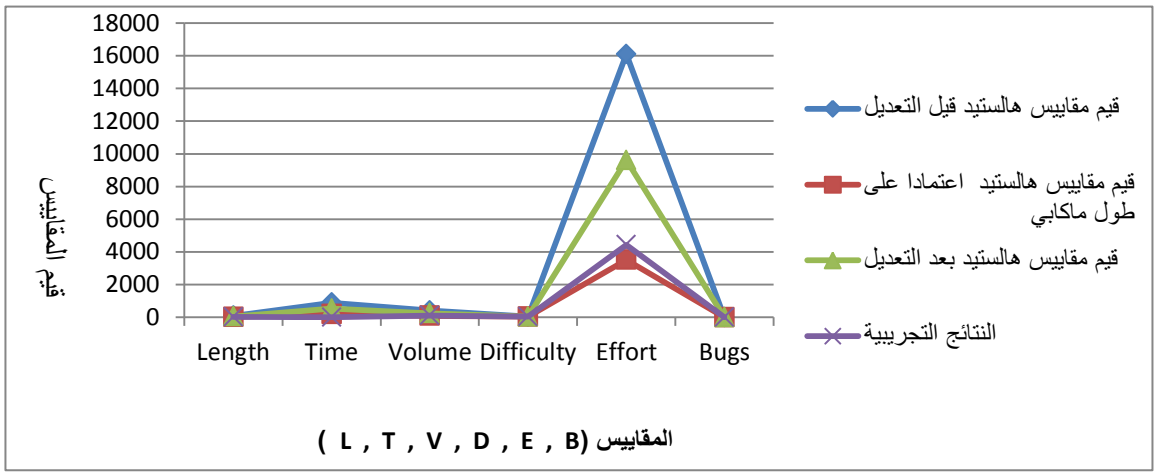
$$N = \lfloor (L_H + LOC) \rfloor = \lfloor (L_H + LOC) / 2 \rfloor = \lfloor (87 + 18) / 2 \rfloor = \lfloor 105/2 \rfloor = 52$$

قمنا بدمج الطريقتين من أجل الاخذ بعين الاعتبار اللغة البرمجية المستخدمة وذلك من خلال مقاربة هالستيد ، ومقاربة الاختبارات الأساسية اللازمة اعتماداً على التعقيد الحلقي من خلال طريقة ماكابي . نقوم الأن بإجراء الحسابات لمقاييس هالستيد اعتمادا على الطول الجديد المقترح $(N=52)^{1}$ ، فنحصل على نتائج جديدة ومحسنة عن قيم القياسات المحسوبة قبل التعديل ، قيم القياسات قبل وبعد التعديل مع القيم المعتمدة على النتائج التجريبية موضحة في الجدول رقم (7) الآتي :

الجدول رقم (7)

Bugs	Effort	Difficulty	Volume	Time	Length	
0.136	16090.215	39.35	408.9	893.900	87	**قيم هالستيد قيل التعديل**
0.0297	**3513.955**	39.35	89.3	**195.219**	18	**قيم مقاييس هالستيد اعتمادا على طول ماكابي**
0.0814	9617.14	39.35	244.4	534.28	52	**قيم هالستيد بعد التعديل**
0.0376	**4438.68**	39.35	112.8	246.59 C++	24	**النتائج التجريبية**

المخطط في الشكل رقم (3) الآتي يوضح المقارنة بين قيم القياسات اعتمادا على الآليات الثلاث السابقة مع الآلية المقترحة.



الشكل رقم (3) يوضح المقارنة بين القياسات الناتجة عن آليات حساب طول البرنامج التقليدية مع الآلية المقترحة

بدراسة النتائج في الجدول رقم (7) ، والشكل رقم (3) نلاحظ أن قيم القياسات المبنية على التعديل الذي قمنا به أقرب إلى القيم المبنية على الدراسة التجريبية، فمثلاً الحجم قبل التعديل كان 408.9 ، وبعد التعديل أصبح 244.4 وتجريبياً 112.8 ، ايضا الزمن قبل التعديل كان 893.900 ثانية وبعد التعديل أصبح 534.28 ، أما عدد الثغرات قبل التعديل 0.136 وبعد التعديل 0.0814 ، ونفس التغيرات المنطقية وجدناها عند مقارنة النتائج بعد التعديل مع النتائج المبنية على طول LOC

[1] $[(x)]$ تعطي أقرب عدد صحيح لـ x

9- النتائج والتوصيات

قمنا في هذا البحث بإجراء عدة تجارب عملية على برامج بسيطة قام بكتابتها مجموعة من المبرمجين ، ثم قمنا بإجراء عدة تجارب على البرنامج المبين في الشكل رقم (1) وبعدة لغات سجلنا النتائج الزمنية اللازمة لكتابة هذه البرامج وكذلك عدد الأسطر التي نحتاجها في كل لغة ثم قمنا بإجراء الحسابات اللازمة لإيجاد قيم مقاييس هالستد اعتماداً على النتائج التجريبية التي حصلنا عليها ، ومن ثم قمنا بإجراء تعديل على طريقة حساب طول البرنامج وحسبنا قياسات هالستيد بناءً على القيم الجديدة الناتجة عن هذا التعديل ، وعلى طول هالستيد ، واعتماداً أيضاً على اختبارات LOC مرة أخرى.

وبمقارنة نتائج قيم القياسات وجدنا ما يلي:

1-يوجد تباعد كبير بين قيم القياسات المحسوبة قبل التعديل وبين قيم القياسات المحسوبة بناءً على النتائج العملية.

2- اختلافات كبيرة بين القيم التي اعتمدنا في حسابها على LOC والقيم التي اعتمدنا في حسابها على هالستد.

3- اختلافات كبيرة بين قيم هالستد وقيم LOC من جهة، وبين القيم الناتجة عن التجارب السابقة الذكر.

4- تم تحسين (تخفيض) قيم قياسات هالستيد بعد التعديل بنسبة تقارب 40%، وتم تحسين (زيادة) قيم قياسات هالستيد المبنية على طول ماكابي(المتفائل) بعد التعديل بنسبة أكثر من الضعف، وهذا جعل قيم القياسات تقترب من القيم المبنية على الدراسة التجريبية والتي تجعلنا نطمح أن تتطبق في البرامج الكبيرة والمكونة من ملايين من الأسطر البرمجية القياسية.

التوصيات

يمكن العمل في المستقبل من أجل برامج أكبر، ومن أجل لغات برمجية أخرى، كما يمكن البحث في تحسين عدد حالات الاختبار اللازمة بناءً على القيم التجريبية التي تنتج، وصولا إلى صياغة حالات اختبار بعدد أقل وفاعلية أكبر تنعكس على جودة المنتج البرمجي.

المراجع

1- Olabiyisi .S., Omidiora E. and Sotonwa. K., Comparative Analysis of Software Complexity of Searching Algorithms Using Code Based Metrics, Ladoke Akintola University of Technology, Nigeria. International Journal of Scientific & Engineering Research, Volume 4, June-2013

2- Neha .S, Sapna. K, Anushree. A, University, Delhi, A Study of Significant Software Metrics, International Journal of Engineering Inventions, Volume 3, July 2014.

3- Gharuan , Mohali ."Comparative analysis of software metrics on the basis of complexity". Chandigarh University, India. International Journal of Computer Science & Engineering Technology (IJCSET) , Vol. 5 , Dec 2014.

4- Prachi. C, Lalit.B , An Effective Implementation of Improved Halstead Metrics for Software Parameters Analysis, International Journal of Computer Science and Mobile Computing, IJCSMC, Vol. 3 , August 2014.

5- Gustavo .V¹, Danyllo .A, Eduardo .F, Alessandro .G , Defining Metric Thresholds for Software Product Lines: A Comparative Study , Department of Computer Science Federal University of Minas Gerais -OPUS Research Group – Software Engineering Lab – Informatics Department Pontifical Catholic University of Rio de Janeiro , Brazil ,2015 .

6- Sandeep. K, Navjot. K, Software Metrics and Metric Tools- A Revie, International Journal on Recent and Innovation Trends in Computing and Communication, Volume: 3, April 2015.

7- Maksim. S, Vyacheslav. Z, Improving Fuzzing Using Software Complexity Metrics, Springer International Publishing Switzerland 2016..

8- Mohammed .A, Rizwan .Q, COMPARATIVE STUDY OF SOFTWARE ESTIMATION TECHNIQUES, International Journal of Software Engineering & Applications (IJSEA), Vol.8, November 2017.

9- Xiaoxing Yang. "Evaluating Software Metrics for Sorting Software Modules in Order of Defect Count". Sun Yat-Sen University, China. International Conference on Software Technologies (ICSOFT) 2019.

10- Salisu ,G, METRIC-BASED FRAMEWORK FOR TESTING & EVALUATION OF SERVICE-ORIENTED SYSTEM, International Journal of Software Engineering & Applications (IJSEA), Vol.10 , May 2019.

11- Jozsef.M, Alexandra .N¸ and Simona .M, Evaluation of Software Product Quality Metrics , Babe¸s-Bolyai University, Romania ,2020

12- D urgesh. R, Introduction to Software Testing, International Journal of Trend in Scientific Research and Development (IJTSRD) , Volume 4 , April 2020.

13- Luca .A, Luca .B, Riccardo. C, Michele .V, Evaluation of Rust code verbosity, understandability and complexity, Peer J Computer Science, Italia, February 2021.