تقويم جدوى استثمار المكتبات التي توفرها لغات البرمجة

Evaluation of Programming Language's Libraries Investment

نادرة جميل علي

جامعة البصرة / كلية التربية/ قسم الحاسب

Abstract

Reusing functions and classes in object oriented programming languages considered as the first real experience for software reuse concept. Libraries investments considered as the most important feature to enhance programmer productivity achieving good quality software product because of it's positive effect on the software performance.

In addition of saving lots of efforts and time because you don't need to start from scratch rather you can reuse the existing, well defined, carefully tested, well documented, portable, widely available components which increases reliability, modifiability and the consequences of the ease to manage, modify and maintain. These units need not to be recompiled again, this supports the success to achieve the object of software production project and the object of managing it. High quality production make time and budget management easier and reduces after sell services cost.

This paper investigate merits of OOP that invest the standard libraries against conventional OOP written in the same language but not investing the existing libraries on quantitative basis. Measurements was made by using some of the well known software metrics [1-2] comparing the results for object oriented programs that use libraries with conventional programs doing the same thing but does not use the exist libraries, the results for the software metrics I get is much better for programs that use libraries. I consider this paper as an invitation to use the existing libraries to achieve effective software industry.

ملخص البحث

إن إعادة استخدام الدوال والفئات في لغات البرمجة الكيانية تعتبر أول تجربة حقيقية لمفاهيم إعادة الأستخدام (reuse) . استثمار مكتبات اللغات يعتبر من أهم الخواص لتعزيز الأنتاج البرمجي ولتحقيق نوعية جيدة من الأنتاج بسبب تأثيرها الأيجابي على أداء البرمجيات بالأضافة لتوفير الكثير من الوقت والجهد وذلك لأن المبرمج لايبدأ من الصفر وإنما يستخدم وحدات برمجية جاهزة معرفة معرفة بصورة جيدة، وتم اختبارها بعناية وجيدة التوثيق وقابلة للنقل ومتوفرةبصورة واسعة. حيث تزيد من الموثوقية وقابلية التعديل، سهولة الأدارة، والتعديل والصيانة. هذه الوحدات لا تحتاج إلى إعادة ترجمة وهذا يدعم النجاح في تحفيز هدف مشروع انتاج البرمجيات والهدف من إدارتها. الأنتاج ذو النوعية العالية تجعل إدارة الوقت والموارد اسهل وتقلل من تكاليف خدمات مابعد البيع.

هذه الورقة البحثية تتحقق بمحاسن البرمجة الكيانية من خلال برامج تستثمر المكتبات القياسية مقارنة ببرامج مكتوبة بنفس اللغة ولا تستثمر المكتبات المتوفرة على اسس كمية.

تم عمل قياسات باستخدام بعض مقاييس البرمجيات المعروفة [2 – 1] وتم مقارنة النتائج للبرامج الكيانية التي تستثمر المكتبات مع برامج تحل نفس المشاكل ولا تستثمر المكتبات المتوفرة، وكانت النتائج التي التي حصلت عليها لمقاييس البرمجيات أفضل بكثير للبرامج التي تستثمر المكتبات وتعتبر هذه الورقة كدعوة لأستخدام المكتبات المتوفرة لتحقيق صناعة برمجية مؤثرة.

1. المقدمة

إن انتعاش صناعة البرمجيات والنمو الهائل الذي حققته في العشرين سنة الأخيرة وخاصة في الدول المتقدمة مقارنة بالصناعات الأخرى، جعل منها مشروعا طموحا للدول النامية ومنها دول وطننا العربي. وهذا التحدي أفرز واقع جديد وإنقلاب عنيف في المفاهيم والآليات والأدوات التي تحتاج لها هذه الصناعة في تحسين القدرة التنافسية في سوق البرمجيات للمنتجين ورفع مساهمتها في الدخل القومي للدول المنتجة. ومن المفاهيم الجوهرية التي تبنتها نماذج الإنتاج الصناعي للبرمجيات مبدأ إعادة الإستخدام (Software Reuse) لما يحققه من تحسين في سمات جودة المنتج البرمجي وتقليص كلف إنتاجه وتسهيل خدمات ما بعد البيع وإدامة المنتج. ولعل تبني مفهوم التوجه الكياني والبرمجة الشاملة (Generic Programming) وتطوير آليات وأدوات لتنفيذها ما هو إلا إفرازات لتفعيل مبدأ إعادة الإستخدام. وهكذا لاحظنا التطور الكبير في لغات البرمجة وفلسفتها. فمكتبات لغات البرمجة التي طورت لدعم مسيرة الإنتاج الصناعي للبرمجيات تميزت بسعة حجمها وتوفيرها لعشرات الآلاف من المكونات التي يمكن إستدعائها وتكاملها في بناء برمجيات تحقق المتطلبات الصناعية.

تشجيع استثمار المكتبة في إعداد وتنفيذ تصاميم النظم البرمجية ووحداتها ينسجم وتحقيق أهداف صناعة البرمجيات بزيادة إنتاجية الفرد وتوفير الجهد وحسن إستغلال الموارد المتاحة وتسهيل عملية إدارة المشروع الإنتاجي وتسهيل خدمات الإدامة لما بعد البيع. إن هذه المكتبات تتمتع بكفاءة وموثوقية عالية جدا وقد تم اختبارها لمجاميع مختلفة من البيانات التي تمثل معظم الحالات التي يفترض أن تعمل عليها. ورغم أهمية ذلك نجد المناهج الدراسية تخلو من التوجه للإستغلال الكامل للمكتبات التي توفرها لغات البرمجة.

في هذه الورقة تم إجراء تقييم ومقارنة لبرامج مكتوبة بلغة C++ وبدون استثمار للمكتبات مع برامج تؤدي نفس الغرض وباستثمار كبير لمكتبة STL القياسية والمكتبات الأخرى.

لذلك يمكن اعتبار هذا الجهد المتواضع بمثابة دعوة للإهتمام باستثمار المكتبات القياسية لما توفره من جهد ووقت كبيرين.

2. لمحة تاريخية لتطور المكتبات التي توفرها لغات البرمجة:

من السهل فهم وإدارة البرامج القصيرة ولكن عندما يزيد حجم البرنامج عن بضعة آلاف من السطور يصبح غير منظم مما تصعب إدارته، بالإضافة إلى إن عملية إعادة الترجمة ستكون مكلفة جدا عند عمل أي تعديل على البرنامج، لذلك لجأ مصمموا البرامج في اللغات المهيكلة إلى تجزئة البرنامج إلى أجزاء صغيرة [3] دوال (functions) و إجراءات procedures). لذلك تمكن مصممو البرامج من كتابة مجموعة من مكتبات البرمجيات والتي هي عبارة عن مجموعة من هذه الدوال والإجراءات لحل مشاكل حسابية ، إحصائية وغيرها مثل مكتبة NAG الشهيرة و

Mathlab وغيرها كثير والتي تعتبر بمثابة مكتبات خارجية بالإضافة إلى المكتبات التابعة للغة. عندما يحتاج المبرمج أحد هذه الدالات يستدعيها في برنامجه وذلك بذكر اسم الدالة متبوعا بقائمة العوامل الخاصة بها بين قوسين لتقوم بمهمتها وتعيد النتيجة بدون معرفة المستخدم لتفاصيل طريقة العمل [4]. في لغة C مثلا يجب أن يتضمن البرنامج الملف الرأسي <cmath> ليتمكن من استخدام إحدى الدوال الرياضية القياسية مثل ( )sqrt، ( )sin أو ( )cos وغيرها كثير والمتوفرة على شكل دالات في هذا الملف. هذه الطريقة لتجزئة البرنامج جعلت إدارته أسهل وجعلته سهل الفهم وكذلك سهلت إعادة استخدام أجزاء منه دون الحاجة إلى إعادة كتابة الكود مرة أخرى (reuse). ولكن مشكلة هذه اللغات هي عدم اهتمامها بالبيانات Data ولهذا فإن مستخدمي هذه المكتبات كانوا يواجهون بعض المتاعب لأن الدوال لا تعمل مع نوع البيانات المتوفر لديهم مما يضطرهم إلى طلب الكود الأصلي وتعديله بما يتلائم وبياناتهم وإعادة ترجمته وهذا يتطلب جهدا ووقتا كبيرين بالإضافة إلى فقدان السرية بالنسبة للشركة المصنعة.

في لغات البرمجة الكيانية التوجه (Object Oriented) مثل C++ و جافا تتكون مكتباتها من وحدات أخرى غير الدوال والإجراءات، وهي أنواع جديدة يعرفها المبرمج (user defined types) تسمى classes صفوف وهو عبارة عن كبسولة تحتوي على البيانات (data member) وكذلك الدوال (member function)التي تعمل على هذه البيانات. لقد وضعت هذه الصفوف في ملفات عديدة على شكل مجموعات ويستطيع المستخدم أن يصل إليها ليعيد استخدامها وذلك بتضمين الملف الذي يحتوي على الصف المطلوب في برنامجه حيث هناك آلية خاصة بكل لغة، #include<filename> بلغة C++، و import package_name بلغة جافا، في بداية البرنامج بالإضافة إلى خلق كيان object من نوع الصف المطلوب والذي سيتم استدعاء الدوال المطلوبة عن طريقه. إن آلية إعادة الاستخدام (Reuse) في اللغات الكيانيه أعطت للبرنامج مرونة كبيرة

حيث يستطيع المستخدم أن يعيد استخدام الصفوف من المكتبات كما هي دون إجراء أي تعديل عليها أو إجراء أي تعديلات مطلوبة دون الإطلاع على الكود الأصلي للصف أو تعديله وهذا يزيد من عنصر إخفاء المعلومات information hiding.

3. دور استثمار مكتبات لغات البرمجة في جودة المنتج البرمجي وتحسين القدرة التنافسية:

لا شك إن مكونات المكتبات التي توفرها لغات البرمجة تمثل وحدات برمجية قياسية دولياً [3] (International Standards) مما يحقق مبدأ القياسية (Standardization) للمنتج البرمجي والقياسية هي أحد أهم سمات الجودة للمنتج البرمجي الصناعي لما يوفره من تسهيل لعملية الإنتاج وخدمات ما بعد البيع وخفض في كلف اختبار المنتج (والتي تعتبر أكبر كلف في العملية الإنتاجية قد تصل بموجب العديد من الدراسات إلى أكثر من 40% من الكلف الإجمالية لإنتاج أي منتج صناعي برمجي) [5]. إضافة لما تقدم فإن تشجيع استخدام الوحدات البرمجية التي توفرها مكتبات لغات البرمجة يحقق مبدأ المتانة (consistency) للمنتج البرمجي ويساهم في رفع درجة تماسكه (Cohesive) واستقلالية مكوناته مما يحسن إمكانية استغلال اجزاء منه في بناء منتجات برمجية اخرى وما في ذلك من أهمية في دعم صناعة البرمجيات وتحسين القدرة التنافسية في سوق البرمجيات. ومما تقدم يتضح أن مراعات رفع درجة استغلال المكتبة في تصميم وحدات المنتج البرمجي يساعد على الأقل فيما يلي:

1) رفع مستوى الإنتاجية من خلال استخدام هذه الوحدات البرمجية الجاهزة التي توفرها لغات البرمجة.
2) تحسين جودة المنتج و موثوقيته من خلال كون هذه الوحدات مستخدمة في تطبيقات كثيرة سابقة ومبرهنة الجودة مسبقاً.
3) قياسية المنتج بسبب كون هذه المكونات قياسية وشائعة الاستخدام وجزء لا يتجزأ من لغة البرمجة المستخدمة.

4) خفض كلف الإنتاج إلى 80% أحيانا وذلك بعدم البدء من الصفر بل إعادة استخدام جهود من سبقونا وتطويرها بالاتجاه المطلوب.
5) خفض الكلف وزيادة الإنتاجية يدعم سهولة تحقيق أهداف إدارة مشروع الإنتاج البرمجي من خلال ترشيد إدارة ميزانية المشروع ومدة الإنتاج.

4. المقاييس المقترحة:

انطلاقاً من كون المقياس بمفهومه مؤشر كمي دال لخاصية من خواص ما، يتم تقويمه، يتسم بالوضوح والدلالة، سهل الحساب، ويخضع لميزان واضح (Scale) تقترح الدراسة المقياس التالي: –

المقياس الأول: ينتج عنه مؤشر كمي يخضع لميزان النسبة (Ratio Scale) ارتفاع قيمته تدل على ارتفاع مستوى استغلال المكتبة بإعتباره مؤشر جودة وإنخفاضه يوضح تدنيها. القيمة صفر للمقياس تعني عدم استغلال المكتبة والقيمة 100% تعني أن الوحدة البرمجية تستند بالكامل على مكونات المكتبة. الصيغة المقترحة سهلة الحساب وبذلك تحققت شروط المقياس. بالآضافة الى المقاييس اللأولية

5. التجربة :

تعتمد مصداقية القياسات والنتائج على نماذج البرامج المستخدمة. وقد تم توخي الحذر والدقة في اختيار البرامج[6] حيث تم تطبيق المقاييس على العديد من البرامج خلال تدريس مادة البرمجة الكيانية لطلبة علوم الحاسوب بعدة جامعات وقد تم اختيار برنامجين كنموذج, كتب كل منهما بطريقتين، الأولى بدون استثمار المكتبات المتاحة والأخرى باستثمار احد المكتبات القياسية وقد تم تطبيق المقاييس الشائعة في هندسة البرمجيات [2-1] والمدرجة أدناه على البرنامج الأول ومقار نة النتائج لكلا الطريقتين والمبينة في جدول (1). كذلك تم تطبيق المقاييس على البرنامج الثاني ومقارنة النتائج لكلا الطريقتين جدول (2).

أدناه مجموعة من المقاييس الأولية المستخدمة في هندسة البرمجيات والتي تم

استخدامها في هذه الورقة[5]. لا نريد هنا الخوض في مناقشة هذه المقاييس وسنكتفي باستخدامها:

n1 : عدد المؤثرات المميزة

n2 : عدد العوامل المميزة

N1 : العدد الكلي لتكرار المؤثر

N2 : العدد الكلي لتكرار العامل

n = n1 + n2 : مجموعة مفردات البرنامج

N = N1 + N2 : طول البرنامج

EN : الطول التخميني للبرنامج النقي

حيث

$EN = n1*\log_2 n1 + n2*\log_2 n2$

V : حجم البرنامج وهو عدد البتات المطلوبة، حيث $V = N * \text{Log}_2\ n$

L : مستوى التجريد في البرنامج حيث المستوى التخميني:

$L = (2/n1)*(n2/N2)$

D2 : مقياس الصعوبة ويعطى بالعلاقة:

1/L

LL : مستوى اللغة حيث $LL = L^{2}*V$

E : المجهود حيث : $E = V / L$

IC : المحتويات الذكية حيث:

$IC = L * V$

6 . تنفيذ المقياس ومناقشة النتائج:

الجدول (1) يبين النتائج الخاصة بالبرنامج الأول أ و ب لبعض مقاييس هندسة البرمجيات الموضحة سابقا. يقوم البرنامج ببعض العمليات على سلسلة نصية. البرنامج الأول_أ مكتوب بدون استثمار المكتبات القياسية والأول_ب يؤدي نفس الغرض تماما ولكن باستخدام المكتبة القياسية STL:

يبين الجدول(1) قيم المقاييس للبرنامج الأول(أ) بدون استثمار المكتبة والبرنامج الأول(ب) باستثمار المكتبة.

جدول (1)

	القيم بدون استثمار	القيم باستثمار

المقاييس	المكتبة	المكتبة
n1	29	18
n2	58	19
N1	703	87
N2	379	56
n	87	37
N	1082	143
EN	480.6443	155.7693
V	6971.265	744.9518
L	0.010554	0.037698
D2	94.75	26.52632
LL	0.776521	1.058703
E	660527.3	19760.83
IC	73.57535	28.0835

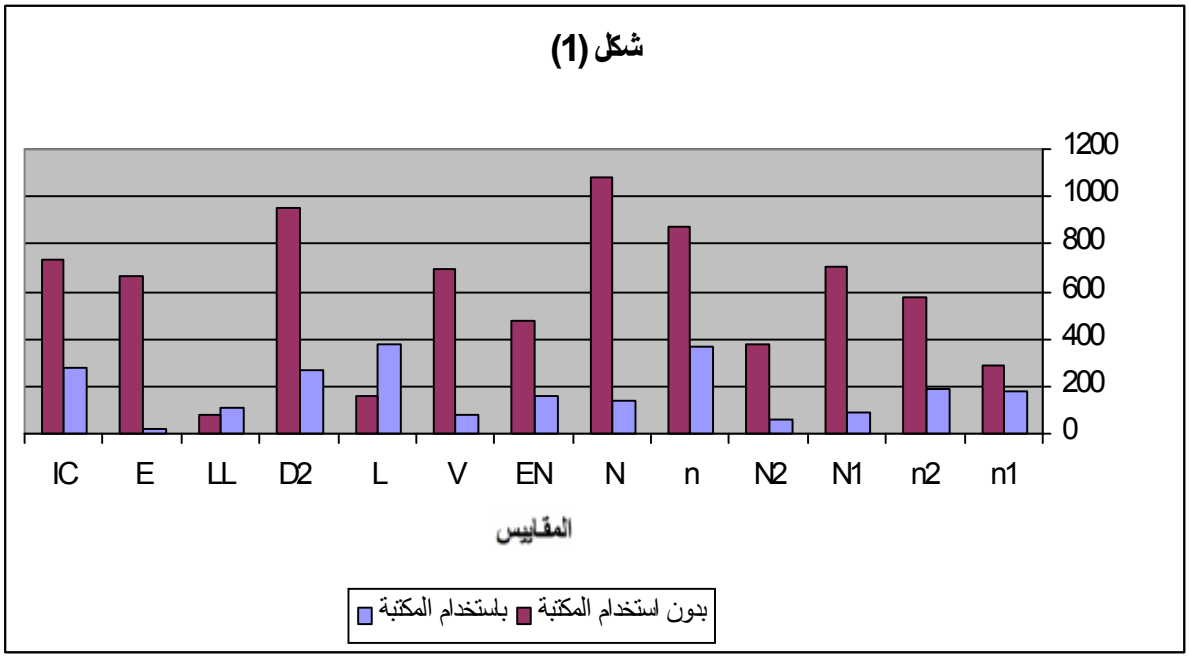
أما بالنسبة لمستوى استثمار المكتبة في البرنامج الأول(أ) فهو : 1.886792.

مستوى استثمار المكتبة للبرنامج الأول (ب) هو: 93.47826 .

يبين الشكل(1) نسبة كل مقياس للبرنامج الأول بدون استثمار المكتبة مقارنة بنسبته باستثمار المكتبة.

الجدول (2) يبين النتائج الخاصة بالبرنامج الثاني أ و ب لنفس المقاييس المستخدمة في

شكل (1)



في الجدول(1).

جدول (2)

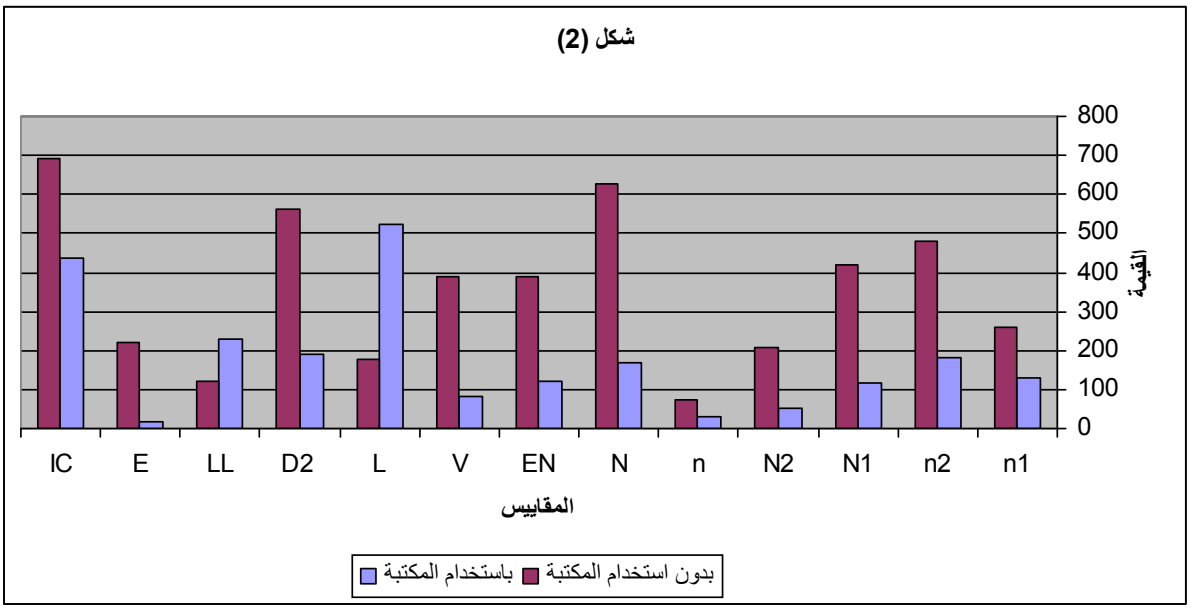
المقاييس	القيم بدون استثمار المكتبة	القيم باستثمار المكتبة
n1	26	13
n2	48	18
N1	41.8	11.6
N2	20.8	5.3
N	74	31
N	62.6	16.9
EN	39.02896	12.31644
V	38.87118	8.372592
L	17.751	52.25
D2	56.33333	19.13889
LL	12.24889	22.85738
E	218.9743	16.02421
IC	69.00209	43.74646

حيث يقوم البرنامج ببعض العمليات على قائمة أرقام. البرنامج الثاني_أ مكتوب بدون استثمار المكتبات القياسية والثاني_ب يؤدي نفس الغرض تماما ولكن باستخدام الصف القياسي الجاهز List.

يبين الشكل(2) نسبة كل مقياس للبرنامج الثاني بدون استثمار المكتبة مقارنة بنسبته باستثمار المكتبة مستوى استثمار المكتبة للبرنامج الثاني (أ) هو : 2.8

وللبرنامج الثاني(ب) فهو : 58

البرنامج الأول (أ) يقوم ببعض العمليات على سلسلة نصية وقد حاولت عدم استثمار المكتبات الجاهزة في اللغة ثم تم عمل نسخة



"البرنامج الأول (ب)" وبنفس اللغة C++ ولكن باستثمار المكتبات المتاحة في اللغة. الجدول 2 يبين نتائج تطبيق نفس المقاييس ولكن على برنامج آخر وهو مبين في الملحق. البرنامج الثاني يقوم ببعض العمليات على قائمة أرقام.

تم كتابة البرنامج بنسختين أيضا النسخة (أ) بدون استثمار المكتبات القياسية والنسخة (ب) باستثمار المكتبات القياسية. النتائج لكل المقاييس المستخدمة موضحة في الجدولين 1 و 2. يتضح من الجدولين بأن النتائج التي

7. الاستنتاج والتوصيات:

لقد تم تطبيق مجموعة من المقاييس الشائعة في هندسة البرمجيات [2-1] (عدد المؤثرات المميزة، عدد العوامل المميزة، العدد الكلي لتكرار المؤثر، العدد الكلي لتكرار العامل، مجموعة مفردات البرنامج، طول البرنامج ... الخ) على البرامج المبينة في الملحق حيث

حصلنا عليها ذات دلالة بما يشير إلى قوة البرامج التي تستثمر المكتبة فعند مقارنة القيم التي حصلنا عليها من البرامج التي لا تستثمر المكتبة مع نظيرتها التي تستثمرها بشكل كبير نلاحظ إن قيم معظم المقاييس في البرامج التي تستثمر المكتبة هي أقل بكثير من نظيرتها في البرامج التي لا تستثمر المكتبة بموجب مقاييس هندسة البرمجيات [2] فالقيم المتدنية لكل من n1 و n2 تعني بقوة تقليل الصعوبة والأخطاء في البرنامج حيث إن استخدام المكتبة قلل من عدد المتغيرات المعرفة والثوابت المستخدمة في تنفيذ الخوارزمية. القيم المنخفضة لهذه المقاييس كان تأثيرها واضحا على قيم كل مفردات البرنامج، طول وحجم البرنامج حيث كانت قيمها اقل بصورة واضحة في حالة استثمار المكتبة. بالنسبة لقياس مستوى اللغة LL فقد حصلنا في تجربتنا على معدل القيمة 0.776521 بعدم استثمار المكتبة و 1.058703 عند استثمارها في البرنامج الأول والقيم 12.24889عند عدم استثمار المكتبة و 22.8573 عند استثمارها في البرنامج الثاني حيث إن القيمة الأعلى تعني مستوى أعلى للغة وبالتالي فهذا يؤثر على قيمة المجهود المبذول لتنفيذ أي خوارزمية في أي لغة برمجة والممثل بالمقياس E في تجربتنا حيث قيمتها في حال استثمار المكتبة اقل بكثير وهو يعني أيضا جهود اقل لفهم البرنامج وصيانته وإعادة استخدامه. إضافة إلى ذلك فإن العلاقة المستخدمة لحساب E تبين إنها تتناسب عكسيا مع مربع LL أي مع مستوى اللغة والتي تعني كفاءة اللغة. أما بالنسبة لمقياس الصعوبة D2 والذي يعتمد على النسبة بين n2 إلى N2 فمن الواضح بأن قيمة D2 التي حصلنا عليها في تجربتنا في حالة استثمار المكتبة هي اقل بكثير من تلك في حالة عدم استثمارها وهذا يبين بوضوح تقليل الصعوبات في البرمجة وعملية حل المسائل الذي سببه استثمار المكتبة.

من هذه التجربة يتضح التأثير القوي لاستخدام المكتبات القياسية في البرامج مقارنة بمثيلاتها التي لا تستثمر المكتبات. وتبين هذه الدراسة أيضا إن مقاييس هندسة البرمجيات يمكن أن تستخدم بصورة متقنة لحساب نوعية اللغات الكيانية وتشكل اساس لمزيد من التحقيقات لحساب تأثيرات السمات الأخرى المختلفة

للبرمجة الكيانية ولغات البرمجة الكيانية. ويمكن توسيع هذه الدراسة بسبب تعدد لغات البرمجة الكيانية لمعرفة أكثرها قوة.

8. المراجع:

[1] M. H. Halstead Elements of Software Science. Elsevier Computer Science Library, New York, 1977.

[2] A. Ahmad. "A Methodology for Development of Software Composite Metrics," Journal of Computer Science and Technology, Vol. 7, No. 3, July 1992

[3] . H. M. Deitel, P. J. Deitel C++ How to program Prentice Hall Fifth Edition 2003.

[4] B. Cox, "Message/Object Programming: An Evolutionary Change in Programming Technology," IEEE Software, Vol.1, No., January 1984, pp.50-61.

[5] "A Measurement Based Comparative Evaluation of Effectiveness of OO Versus Conventional Procedural Programming Techniques and Languages", Proceedings of the Ninth Asia-Pacific Software Engineering Conference (APSEC'02) 2002 IEEE .

[6] D.F. Stubb and N.E.Weber, Data-Structures with Abstract Data Types and C. Brooks/Cole Pub. Co.,1989.

9. الملحق

البرنامج الأول_أ :

```
#include<iostream.h>

class String { friend ostream &operator
        <<(ostream & , const String &);

private:
        char *strptr ;
         unsigned int Length;

        void append1( char *  s1, const
                              char * s2)
        {
        while ( *s1 != '\0')
          ++s1;
       for( ; *s1 = * s2 ; s1++, s2++)
                ;
        }
           int length1(const char * s)
        {
           int l;
         for ( l = 0 ; * s != '\0' ; s++)
              ++l;
           return l;
        }
         void copy1(char * s1 , char * s2)
```

```
        {
           while ( *s2 != '\0')
          {
            for( ; *s1 = * s2 ; s1++ ,s2++)
                  ;
          }
        }

        void swap(char * s1, char *s2)
        {
          char * temp;
         temp = s1;
         s1 = s2;
         s2 = temp ;
        }



public :
     String( char * ="");
     String(const String & c)
          {   Length = c.Length;
              strptr = new char[Length +1];
       if(strptr != NULL)
       copy1(strptr, c.strptr);
          }

String append(const String s2)
          {
```

```
        append1(strptr, s2.strptr);
        return *this;

      }
      const String &operator = (const
                          String & R)
      { if(&R != this)
        {
          delete[ ] strptr;
        Length = R.Length;
          Strptr = new char[Length +1];
           if(strptr != NULL)
                copy1(strptr, R.strptr);
      }
      else
         cout<<"itself"<<endl;
          return *this;}

String operator+( const String & R)
{  String T;
   T.Length = Length +R.Length;
   T.strptr= new char[T.Length +1];
    if(strptr != NULL)
    copy1(T.strptr, strptr);
    append1(T.strptr,      R.strptr);
        return T;
       }
```

```
int operator !=( const String & R)
  {return Compare( R) != 0;}

int operator <(const String & R)
 {return Compare( R) < 0;}

int operator >(const String & R)
{return Compare( R)  > 0;}

int operator ==(const String & R)
{return Compare( R)  == 0;}

int operator >=(const String & R)
{return Compare( R)  >= 0;}

int operator <=(const String & R)
{return Compare( R)  <= 0;}


void copy(String s2 )
{
    copy1(strptr, s2.strptr);
}

void Swap(String &R)
{ char * temp;
   temp = strptr;
    strptr = R.strptr;
```

```
            R.strptr= temp;


            }
           int Compare (const String R)
           {   if ( strptr == R.strptr)
              return 0;
            else
             {if (R.strptr < strptr)
                return  1;
                return -1;}
            }
           int length()
           {  int ll;
 ll = length1(strptr);
 return ll;
 }
};
String::String( char * s)
 { Length = length1(s);
   strptr = new char[Length +1];
   if(strptr != NULL)
   copy1(strptr , s)  ;
 }


ostream &operator<<(ostream & out ,
                          const String & s)
 {out<<s.strptr;  return out;}
```

```
void main()
{String     s1 ="Jordan" ,
                s2 = "Irbid",
                s3  = "Sarah";
 String s4("Tower"), s5 =" ";
 cout<< s1<<"   "<<s2<<"   "<<s3<<endl;
 cout<<s3.append( s2)<<endl;
 s1.Swap(s3);
 cout<<s1<<"     "<<s3<<endl;
 cout<< (s3 == s2)<<endl;
 cout<<( s1<= s2)<<endl;
 cout<<( s2>= s1)<<endl;
 cout<<s1.Compare(s2)<<endl;
 cout<<( s3< s2)<<endl;
 cout<<( s3> s1)<<endl;
 cout<<s1.length()<<endl;
 s1.copy(s4);
 cout<<s4<< "      "<<s1;
 s5= s1; cout<<s5<<endl;
 s5= s2 + s3; cout<< s5;

}
```

البرنامج الأول_ب :

```
#include<iostream.h>
using  std::string;

void main( )
{
string  s1 ="Jordan" ,
s2 = "Irbid",
s3 = "Sarah",
s5=” “ ;
char  s4[ ] = "Tower";
cout<< s1<<"   "<<s2<<"   "<<s3<<endl;
s3.append(s2);cout<<s3<<endl;
s1.swap(s3); cout<<s1<<"     "<<s3<<endl;
cout<< (s3 == s2)<<endl;
cout<<( s1<= s2)<<endl;
cout<<( s3< s2)<<endl;
cout<<( s3> s1)<<endl;
cout<<s1.compare(s2)<<endl;
unsigned int l=s1.length();
cout<<l<<endl;
s1.copy(s4,5);
cout<<s1<< "      "<<s4;
s5= s1; cout<<s5<<endl;
```

```
s5= s2 + s3; cout<< s5;


}
```

البرنامج الثاني_أ :

```
#include<iostream.h>
template<class T>

class List
{
   private:
   int size;
   T * vector;
  int length;
```

```
  int currentpos;
public:
  List(int);
  int ListLength();
void RemoveItem( T Key_To_Remove
                              bool& Found);
  int flag(T s,int &loc);
void InsertItem(T Key_To_Remove);
void GetNext(T& item);
void ResetList();
};
template <class T>
List<T>::List(int l)
{
   size =l;length=0;
   vector = new T[size];
}

template <class T>
int List<T>::ListLength()
{
   return length;
}
template <class T>
int List<T>::flag(T s,int & loc)
{
   if(s < vector[loc])
```

```
  return 1;
  else if(s > vector[loc])
 return 2;
 return 3;
}
template <class T>
void List<T>::RemoveItem(T Key_To_Remove,bool& found )
{
int midpoint ;
int first=0;
int last= size-1;
bool SearchMore=(first<=last);
found=false;
while(SearchMore && !found)
{
midpoint=(first+last)/2;
switch(flag(Key_To_Remove,midpoint))
{
case 1:
last=midpoint-1;
SearchMore=(first<=last);
break;
case 2:
first=midpoint+1;
      SearchMore=(first<=last);
break;
case 3:
```

```
found=true;
      Key_To_Remove=vector[midpoint];
break;
}

}
if(found==false)
cout<<"cant find"<<endl;
else
cout<<"find at location"<< midpoint <<endl;
size = size -1;
for (int m = midpoint ; m< size ; m++)
vector [m] = vector[m +1];
for (m = 0 ; m< size ; m++)
cout<<vector[m]<<"  ";

}

template <class T>
void List<T>::InsertItem(TKey_To_Remove)
{
bool SearchMore ;
int location=0;
if(location==length)
{
  vector[location]=Key_To_Remove;
  length++;
}
```

```
else
{
  SearchMore = true;
  while(SearchMore)
{
 switch( flag(Key_To_Remove,location))
{
case 1:
          SearchMore=false;
          break;
case 2:
      location++;
      SearchMore=(location<length);
      break;

case 3:
      SearchMore=false;
      break;
}
}
if(Key_To_Remove!=vector[location])
{
for(int i=length;i>location;i--)
vector[i]=vector[i-1];
vector[location]=Key_To_Remove;
length++;
}
```

```
else
{cout<<"the
searchkey"<<Key_To_Remove<<"is
                        repeated"<<endl;
size = size-1;}


}
}
template <class T>
void List<T>::ResetList()
{
currentpos=-1;
}
template <class T>
void List<T>::GetNext(T& item)
{
currentpos++;
item=vector[currentpos];
}
void main()
{ int  N; cin>> N;
  List<int> list(N);
  int  item;
  cout<<"we will enter  "<< N<<" items
                                    "<<endl;
  for (int i=0; i<N ; i++)
{
```

```
  cin>>item;
   list.InsertItem(item);
}
 list.ResetList();
 cout<<"\nEnter the item to be removed
                                    "<<endl;
 int ritm;cin>>ritm;
 bool f=false;
 list.RemoveItem(ritm,f);

}
```

البرنامج الثاني_ب

```
#include<iostream.h>
 #include<list.h>
 void main()
 {
    int size ;cin>> size;
```

```
    int *vec= new int[size];
    for(int k =0;k<size; k++)
     cin>> vec[k];
     list<int> l1(vec,vec+size);
    cout<<l1.empty( );
    l1.sort();l1.unique();
    const int  k1 = 9;
    l1.remove( k1);
    int s = l1.size();
    for(int j=0; j<s; j++)
 { cout<<l1.front()<<"  ";
   l1.pop_front();
  }
}
```