ТЕХНОЛОГІЇ РОЗРОБКИ ТА СУПРОВОДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

УДК 004.413

Костів М.М., Сидорова Н.М.
Національний авіаційний університет

СТВОРЕННЯ БІБЛІОТЕКИ ЕФЕКТИВНОГО ПРОГРАМУВАННЯ

У статті розглянута задача створення бібліотеки ефективних методів для мови програмування C# з використанням об'єктно-орієнтованої парадигми, наведені результати експериментів з вибору найшвидших методів програмування відповідно задач в доменах.

В статье рассмотрена задача разработки библиотеки эффективных методов для языка программирования C# с использованием объектно-ориентированной парадигмы, представлены результаты экспериментов по выбору самых быстрых методов программирования относительно задач в доменах.

In the article the task of creation of library with effective methods for programming language C# with using object-oriented paradigm is considered, the results of experimental research for choosing the fastest methods of programming relatively to the tasks in the domains are represented.учасниками форсайт -проекту.

Ключові слова: *стиль ефективного програмування, реверсивна інженерія, конструкція мови програмування, домен.*

Постановка проблеми

Програми, які працюють в режимі реального часу критичні до синхронізації отримання, обробки і передачі даних за прийнятні інтервали часу. Подібні програми вимагають, як правило, оптимізації по навантаженню процесора і синхронізації з системними службами операційної системи. Оскільки, на платформі .NET з використанням мови програмування C# розробники створюють драйвери або сервіси для роботи з операційною системою, наприклад з Windows 7, то неефективний програмний код в кращому випадку уповільнить роботу всієї операційної системи [1].

Розробники надають користувачу програмне забезпечення разом з системними вимогами (до частоти процесора, об'єму оперативної пам'яті, тощо) для можливості його використання на даному апаратному забезпеченні. З підвищенням ефективності програмного забезпечення можливо знизити мінімальні вимоги до апаратного забезпечення та втрати і створити зелену програму [2, 3].

Сучасне програмне забезпечення має більше функціональних можливостей і складніше за прості застосування з текстовим інтерфейсом, тому потребує більше ресурсів для своєї роботи [4].

Найпростіший спосіб зменшити час виконання програми полягає в тому, щоб збільшити обчислювальну потужність комп'ютера за рахунок використання більш продуктивного процесора або розширення об'єму пам'яті. Проблема продуктивності легко буде вирішена з вдосконаленням апаратної частини. Але програміст повинен враховувати, що у користувача може не бути останніх моделей процесора і великого об'єму пам'яті. Тому, не потрібно вирішувати проблему продуктивності програмного забезпечення тільки за допомогою нового обладнання.

Підвищення продуктивності програмного забезпечення можна виконати за наступними напрямами:

- ретельна обробка алгоритму програми;
- використання можливостей мови високого рівня для застосування швидших конструкцій і функцій мови.

Для оптимізації програм існують спеціальні інструменти - профайлери (Visual Studio Team System Profiler, Intel VTune Performance Analyzer), які аналізують готову програму і виявляють місця в коді, де падає продуктивність. На основі отриманих даних розробник самостійно шукає варіанти заміни повільного коду.

В даній статі буде описано розроблену бібліотеку з готовими ефективними методами, які можуть бути використані при створенні програмного забезпечення для підвищення швидкості його виконання. Таким чином розробник зможе ще на етапі написання тексту

програми підвищити її продуктивність шляхом використання методів бібліотеки [5].

Розв'язання проблеми

Бібліотека ефективного програмування, схема побудови якої представлена на рис.1, створена для мови програмування C# за об'єктно-орієнтованою парадигмою на базі наукового методу, який розроблено для стилю ефективного програмування мови PHP [2], і основана на наступних принципах:

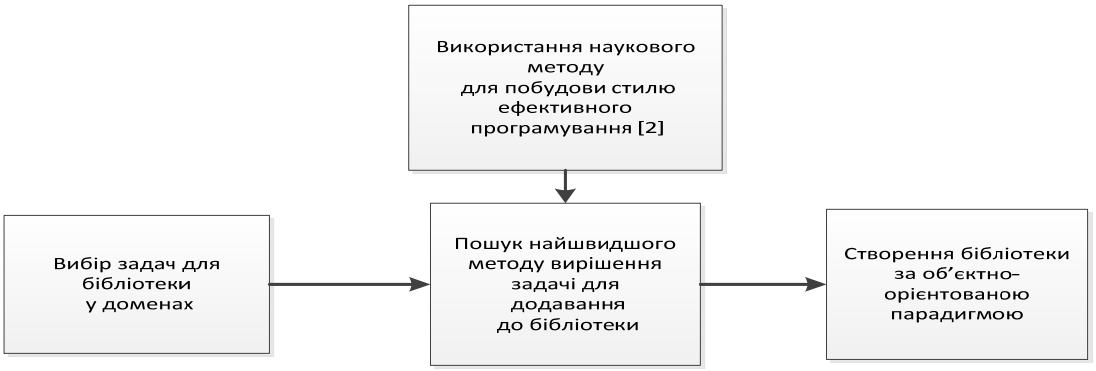

Рис. 1 Схема побудова бібліотеки ефективного програмування

- методи класу мають бути обґрунтованими з точки зору швидкості виконання;
- кожен метод, в загальному, може мати декілька рішень, але в даній бібліотеці повинні міститися лише найшвидші їх варіанти.

Бібліотека складається з ефективних методів для роботи з різними доменами, які описують алгоритми вирішення найпопулярніших задач у певному домені. Для створення бібліотеки необхідно проаналізувати декілька алгоритмів вирішення задачі в домені і обрати найбільш ефективний, який буде реалізовано в бібліотеці шляхом побудови і перевірки гіпотез за науковим методом ефективного стилю програмування [2].

Для перевірки гіпотез потрібно використати асемблерний код, отриманий завдяки інструменту Disassembly [6], що входить до складу середовища розробки Visual Studio 2012.

Для використання методів бібліотеки необхідно визначити критерій за яким програміст зможе зупинити свій вибір на необхідному методі з бібліотеки (вхідні дані). Після аналізу змісту бібліотеки ефективного програмування програміст може вибрати і використати методи бібліотеки (вихідні дані) (рис. 2).

Рис.2 Архітектура бібліотеки ефективного програмування

Оскільки, бібліотека створена за об'єктно-орієнтованою парадигмою з використанням статичних методів, то для застосування методів необхідно вибрати клас бібліотеки і назву необхідного методу (клас бібліотеки – EffectiveCodeLib) (рис. 3).

```
    string[] testStrs = new string[]
{     "Effective",     "Programming",
"Library"}
    lib.ConcatStringBuilder(testStrs
)
```

Рис. 3 Використання методів бібліотеки

Розглядаючи задачу конкатенації рядків з масиву в мові програмування C# з домену «Обробка рядків» необхідно зазначити, що ця задача має два варіанти рішення. Можна використати оператор конкатенації "+" (рис. 4) або клас *StringBuilder* (рис. 5).

Кодом для аналізу є два методи, названі відповідно ConcatPlusOperator та ConcatStringBuilder.

```
    static                          string
ConcatPlusOperator(string[] items){
      string strRet = string.Empty;
      foreach (string item in items) {
                  strRet += item;
       }
    return strRet;
    }
```

Рис. 4 Використання оператора конкатенації "+"

```
    static                          string
ConcatStringBuilder(string[] items) {
    var builder = new StringBuilder();
       foreach (string item in items) {
                  builder.Append(item);
    }
    return builder.ToString();
    }
```

Рис. 5 Використання класу StringBuilder

Для перевірки швидкості виконання кожного методу варто використати клас *Stopwatch*, за допомогою якого час виконання можна виміряти наступним чином:

System.Diagnostics.Stopwatch swatch = new System.Diagnostics.Stopwatch();
swatch.Start(); // початок
// код для аналізу
swatch.Stop(); // кінець
Console.WriteLine(swatch.Elapsed); // виводимо час виконання коду в консоль

Після запуску обох методів, отриманий асемблерний код даних методів зображено на рис. 6 і рис. 7 відповідно, де розглянуті операції конкатенації в обох випадках.

```
0000007b  mov         ecx,dword ptr [ebp
40h]
0000007e  mov         edx,dword ptr [ebp
44h] 00000081  call        6ADF43F8
00000086  mov         dword ptr [ebp
58h],eax
00000089  mov         eax,dword ptr [ebp
58h]
0000008c  mov         dword ptr [ebp
40h],eax
```

Рис. 6. Асемблерний код конкатенації в методі ConcatPlusOperator

```
     0000008d   mov              ecx,dword  ptr
[ebp-40h]
     00000090   mov              edx,dword  ptr
[ebp-44h]
     00000093    cmp                dword  ptr
[ecx],ecx
     00000095  call         6AE09F78
     0000009a   mov            dword ptr [ebp-
5Ch],eax
```

Рис. 7. Асемблерний код конкатенації в методі ConcatStringBuilder

Після аналізу отриманого коду можна сформувати гіпотезу про те, що час виконання методу ConcatPlusOperator буде більшим, оскільки асемблерний код для методу ConcatStringBuilder містить 5 операцій, а для ConcatPlusOperator – 6. Для того, щоб перевірити цю гіпотезу на практиці, потрібно виконати тестовий код для різної кількості елементів масиву рядків, які мають бути об'єднані у один рядок тексту. Відобразивши результати експерименту у вигляді графіку залежності часу виконання методів від розміру масиву (рис. 8) і проаналізувавши його, можна стверджувати, що висунута гіпотеза, щодо швидшого часу виконання методу ConcatStringBuilder, правильна. Тобто для об'єднання елементів масиву краще використовувати клас ConcatStringBuilder.

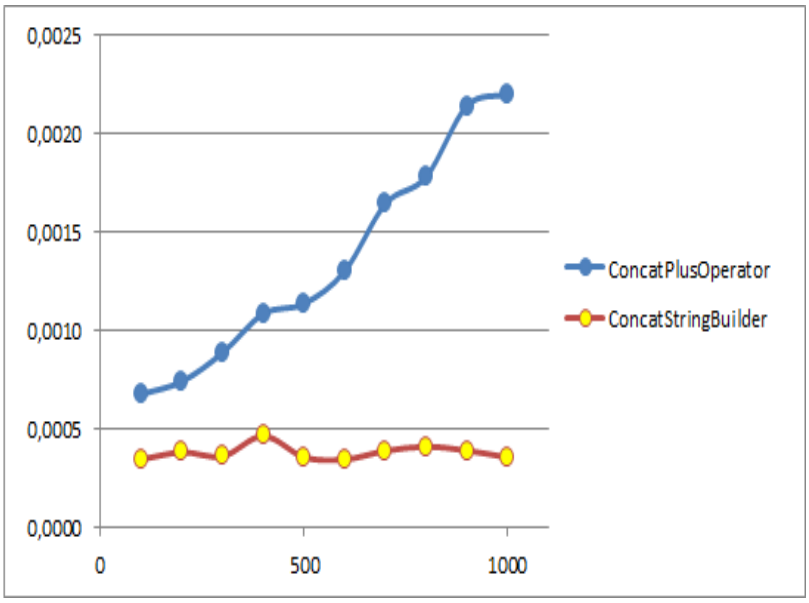

Рис. 8 Графік залежності часу виконання методів ConcatPlusOperator і ConcatStringBuilder від кількості елементів у масиві

Для перевірки об'єму використаної пам'яті потрібно застосувати метод Console.WriteLine(GC.GetTotalMemory(true)).

Відобразивши результати експерименту у вигляді графіку залежності використання пам'яті від кількості елементів у масиві при застосуванні методів ConcatPlusOperator і ConcatStringBuilder (рис. 9), можна зробити висновок, що розмір зайнятої оперативної пам'яті зростає у залежності від кількості елементів масиву, але є незмінним в залежності від використаного методу.

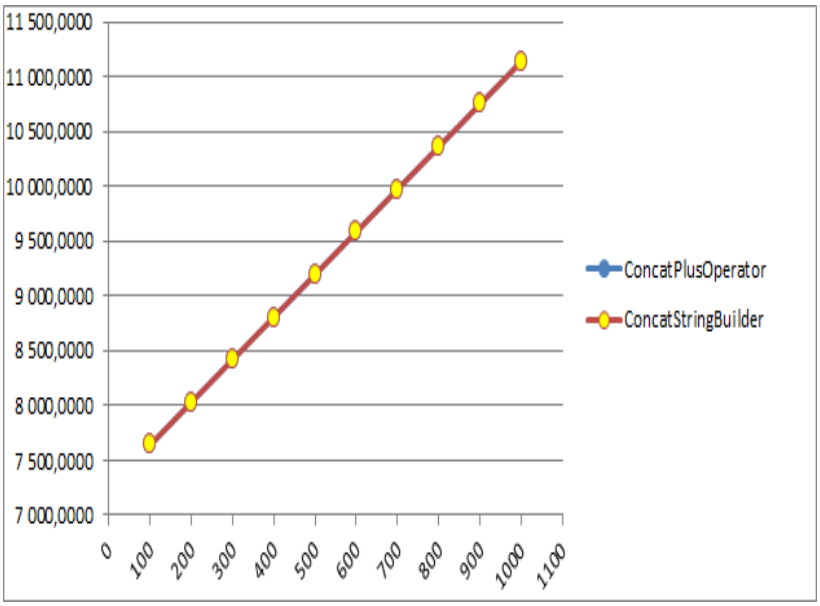

Рис. 9 Графік залежності використання пам'яті від кількості елементів у масиві при застосуванні методів ConcatPlusOperator і ConcatStringBuilder.

Розглянемо ефективний метод обернення рядка у створеній бібліотеці з домену «Обробка рядків». Для бібліотеки був обраний метод із використанням LINQ, що дозволило записати реверсування в один рядок коду (рис. 10).

```
static string ReverseLINQStr(string
input) {
return new
string(input.ToCharArray().
Reverse().ToArray());
}
```

Рис. 10 Метод реверсування рядка з використанням LINQ

Інший алгоритм вирішення задачі реверсування рядка полягає в проході по всім символам рядка і записі символів в інший вихідний рядок, який виступає контейнером.

```
static string
ReversePassStr(string input) {
string output = "";
for (int i = input.Length - 1;
i >= 0; i--)

{
    output += input[i];
}
}
```

Рис. 11 Метод реверсування рядка ReversePassStr

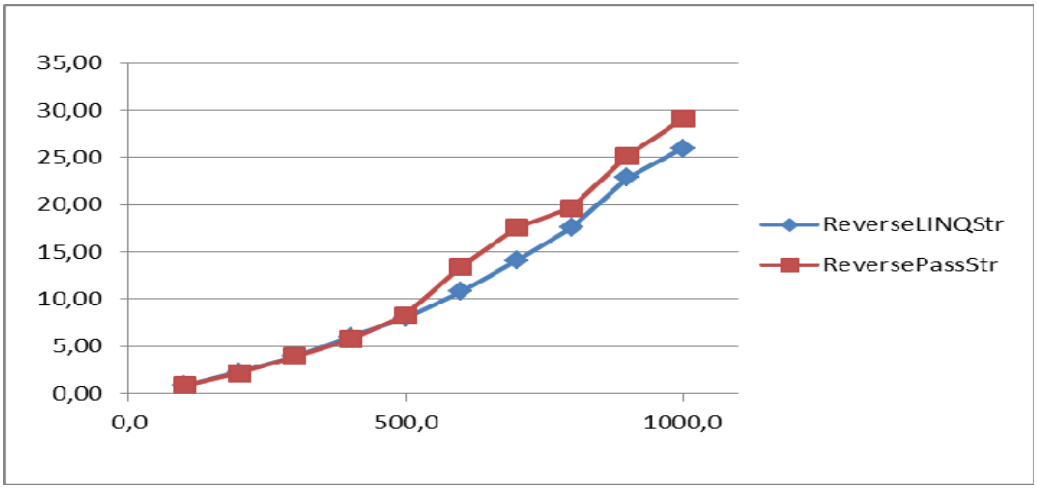

Рис. 12 Графік залежності часу виконання методів ReverseLINQStr і ReversePassStr від кількості елементів в рядку

На рис. 12 зображено графік залежності часу виконання методів ReverseLINQStr і ReversePassStr від кількості елементів в рядку.

Таким чином, до створеної бібліотеки увійшли найбільш ефективні методи вирішення задач відносно доменів.

У програмі всі методи статичні, тому що при роботі із статичними методами можна підвищити ефективність роботи за критерієм пам’яті.

При створенні 3х об’єктів бібліотеки і виклику не статичного методу було використано 94540 байтів. Програміст може створювати будь-яку кількість об’єктів для роботи з не статичними методами і використовувати додаткову пам'ять, але в результаті дослідження було встановлено, що без створення об’єкту і при виклику статичних методів через клас було використано 94500 байтів пам’яті. Дані отримані за допомогою системного методу GC.GetTotalMemory(true).

Висновки

При створенні бібліотеки застосовано науковий підхід для формування стилю ефективного програмування [2].

Розроблена бібліотека допомагає створювати оптимальні програми при використанні програмістом готових і перевірених на ефективність методів.

До бібліотеки методи додавались шляхом реалізації найшвидших алгоритмів для вирішення задач. Кожна реалізація була перевірена на швидкість і об’єм оперативної пам’яті, яка необхідна для її виконання.

Подібна бібліотека може бути створена для інших мов програмування.

Список використаних джерел

1. Магда.Ю.С. Ассемблер. Разработка и оптимизация Windows-приложений. – С.-Пб.: БХВ-Петербург, 2003. – 544с.

2. Сидоров М.О., Костів М.М.. Метод створення ефективного стилю програмування // Інженерія програмного забезпечення – 2013. – № 3 – 4 (15 – 16) – С. 17 – 24.

3. Сидоров Н.А. Экология программного обеспечения // Інженерія програмного забезпечення. – 2010. – №1. – С.53 – 61.

4. Костів М.М., Крамар Ю.М., Інструмент для створення ефективного стилю програмування // Інженерія програмного забезпечення. – 2014. – № 1 (17) – С. 28 – 31.

5. Крамар Ю. М. Стили программирования в конструировании программного обеспечения // Інженерія програмного забезпечення. – 2011. – № 1(5). – С. 46 – 53.

6. How to: Use the Disassembly Window [Электронный ресурс] – Режим доступа: http://msdn.microsoft.com/en-us/library/a3cwf295.aspx

Відомості про авторів:

Костів Мілана Миколаївна – асистент кафедри інженерії програмного забезпечення Інституту комп'ютерних інформаційних технологій Національного авіаційного університету. Наукові інтереси: інженерія програмного забезпечення.

E-mail: milana.kostiv@livenau.net

Сидорова Ніка Миколаївна – аспірантка кафедри інженерії програмного забезпечення Національного авіаційного університету. Наукові інтереси: інженерія програмного забезпечення, освіта.

E-mail: nika.sidorova@livenau.net