УДК 004.04 **Информационные технологии**

Горячкин Борис Сергеевич, *кандидат технических наук, доцент; Московский государственныйтехнический университет им. Н.Э. Баумана*

Бакланов Никита Владимирович, *магистрант, Московский государственный технический университет им. Н.Э.Баумана*

Попов Максим Алексеевич, *магистрант, Московский государственный технический университет им. Н.Э.Баумана*

ПОНЯТНЫЙ КОД

Аннотация: Написать программу на C# далеко не всегда просто, более того ее трудно написать так, чтобы программист, который будет заниматься ее поддержкой и доработкой, смог оперативно разобраться в коде. Большое значение в вопросе восприятия программного кода человеком играет соблюдение эргономических стандартов и критериев. Однако, наше время требует новых подходов и решений к написанию кода. В данной статье был введен термин: «понятный код», а также созданы новые критерии написания кода. Проведен анализ критериев понятного кода с помощью тестов, в которых участвовали 100 программистов языка C#.

Ключевые слова: Понятный код, эргономичность, критерии, C#, юзабилити, стандарты, семантические, тестирование.

Abstract: It is not always easy to write a program in C#, moreover, it is difficult to write it so that the programmer who will be engaged in its support and revision can quickly understand the code. Compliance with ergonomic standards and criteria is of great importance in the issue of human perception of program code. However, our time requires new approaches and solutions to writing code. In this article, the term "understandable code" was introduced, and new criteria for writing code were created. An analysis of the criteria for understandable code was carried out using tests in which

100 C# programmers participated.

Keywords: Clear code, ergonomics, criteria, C#, usability, standards, semantic, testing.

Введение

В работе программиста эргономика играет важную роль. Рынок программного обеспечения постоянно требует совершенствования и качественной поддержки существующего программного кода, а также создания новых программ. Над этим вопросом трудится множество разработчиков, использующих в своей работе самые различные языки программирования. Программистам очень часто приходится вязнуть в чужом беспорядочном коде. Замедление в разработке может быть очень большим. Каждое изменение, вносимое в код, в котором у программиста нет полного понимания, может нарушать работу кода в одном или нескольких местах. Для каждого дополнения или модификации системы необходимо понимать все тонкости кода. По мере накопления плохого кода производительность компании начинает снижаться, приближаясь к нулю. В ходе снижения производительности начальство подключает к проекту новых работников в надежде повысить производительность. Но новые программисты также ничего не понимают в архитектуре системы. Они не знают, какие изменения соответствуют намерениям проектировщика, а какие им противоречат. Также новые программисты находятся под большим давлением со стороны начальства, поэтому в спешке они работают все небрежнее, от чего производительность компании только продолжает падать. В итоге разработка системы затухает, а вскоре прекращается, так как накопление ошибок становится огромным, а понимание кода становится невозможным. Компании приходится с нуля делать новую систему с таким же функционалом, что может отбросить компанию на десятки лет назад.

Для увеличения производительности компаний и эффективности работы с ПО существуют стандарты разработки программного обеспечения. Однако,

наше время требует новых подходов и решений к написанию кода, который сможет поддерживаться много лет. Языком программирования при разработке критериев и тестировании был выбран C#.

Эргономичность программного обеспечения

В области разработки программного обеспечения эргономика играет очень важную роль и призвана сделать программный код понятным, компактным, комфортным для восприятия не только для автора, но и для стороннего разработчика. Данные аспекты крайне важны в современном мире, так как программные средства очень динамично развиваются, появляются новые программы, созданные как «с нуля», так и в результате переработки и модернизации существующих.

Построение эргономически понятного программного кода является очень важным этапом, поскольку именно здесь происходит понятие пригодности использования и поддержания программного кода, дальнейшее развитие процесса разработки. Под пригодностью использования (usability) понимается свойство продукции, при наличии которого продукция может применяться в определенных условиях использования для достижения установленных целей с необходимой результативностью, эффективностью и удовлетворенностью. В стандарте ИСО 9241-12 существуют эргономические требования для программного кода в рамках подхода usability:

1. Требование к пригодности использования разработанного программного кода. Это требование гарантирует возможность применения разработанного программного продукта для достижения поставленных целей с необходимой эффективностью.

2. Требование к условиям использования программного кода. Это требование гарантирует достижение поставленных целей с использованием заданного программного и аппаратного обеспечения в условиях физической и социальной среды, в которых используется программное обеспечение.

3. Требование эффективности программного кода. Это требование гарантирует достижение поставленных целей с использованием минимального

количества ресурсов.

4. Требование доступности программного кода. Это требование гарантирует комфортное восприятие программного кода при выполнении операций его редактирования, рефакторинга, тестирования и сопровождения.

5. Требование к удобству интерфейса программного продукта. Это требование гарантирует естественную и комфортную работу с программным обеспечением. При удобном интерфейсе программного продукта не должно возникать вопросов о последовательности выполняемых действий, об ограничениях на вводимые данные, поиске элементов управления программным продуктом.

Также в стандарте ИСО 9241-12 приведены следующие характеристики представляемой информации:

• Четкость – информационное содержание передается быстро и точно;

• Распознаваемость – отображаемая информация может быть точно распознана;

• Лаконичность – предоставляется только та информацию, которая необходима для выполнения задачи;

• Постоянство – одинаковая информация представлена одинаковым образом во всем программном коде, согласно ожиданиям разработчиков;

• Обнаруживаемость – внимание разработчика направлено на требуемую информацию;

• Разоборчивость – программный код легко прочесть;

• Понятность – значение программного кода понятно, недвусмысленно, интерпретируемо и узнаваемо.

Понятие «удобство ПО» - размыто и понимается каждым по-своему, все характеристики, приведенные в международных стандартах, требуют ужесточения под конкретный проект или предметную область. Поэтому необходимо ввести общие критерии написания программного кода, которые будут применимы к любым проектам. Для этого необходимо ввести новый термин: «понятный код».

Понятный код — это код, в котором с первого взгляда видно то, что он делает, для чего он написан, с какими данными работает, какая у него структура. Такой код читается, как книга даже сторонним разработчиком. В нем нет нагромождений, непонятных цифр и букв.

Критерии понятного кода

Основным критерием понятного кода являются семантические ***имена переменных***. Имена переменных используются во всех частях программного кода, поэтому необходимо подходить к их выбору обдуманно. Они должны означать то, что они представляют в жизни или в самой системе. Имена должны быть оригинальными и эксклюзивными для каждой переменной для того, чтобы не создавать путаницы при прочтении кода. Например, при создании программы необходимо создать переменную, в которой будет храниться количество товаров. Чаще всего программисты при подсчете количества используют переменную «k», а, если в программе будет несколько переменных с количеством добавят номера «k1», «k2». При чтении программы сразу возникнет путаница, так как не понятно в какой переменной хранится количество товаров, а в какой количество людей. Программисту придется потратить какое-то количество времени для того, чтобы понять значение каждой переменной. Поэтому необходимо называть переменные полными именами, например: для переменной количества товаров подойдет: «kolichestvoTovarov», а для количества покупателей: «kolvoPokupateley».

Еще одной проблемой с именами переменных, с которой сталкиваются программисты, является поиск нужной переменной в коде. Часто с этим возникают проблемы так, как многие переменные могут иметь одинаковые названия и отличаться на один символ, например «number» и «num». При поиске второй переменной возникнут проблемы потому, что она совпадает с началом другой переменной. А если речь идет о больших программных продуктах, то поиск переменной может растянуться на несколько часов. Для избежания данной проблемы необходимо дать оригинальное имя для каждой переменной, не совпадающей с началом, серединой или концом другой переменной. В данном

примере переменные можно назвать: «numberId» и «numberUser».

Следующим критерием понятного кода является семантические ***имена функций***. По примеру переменных имена должны означать то, что они выполняют в системе или программе. Функция – это какое-то действие. Название должно «говорить», что происходит в данной функции, и с чем это происходит. Поэтому желательно в названии функции использовать глаголы такие, как: do, find, open и т.д. Например функцию которая удаляет товары можно назвать «delete()», но если в коде есть другие функции для удаления, тогда необходимо уточнить: «deleteTovar()». Для лучшей читабельности следует давать полные и содержательные имена для функций, как в примере с уточнением. Ведь для того, чтобы код читался как книга, в нем должны быть не только существительные (переменные), но и глаголы (функции). По примеру имен переменных, для удобного поиска, имена функций не должны совпадать с началом, серединой или концом другой функции или переменной.

Данные правила написания имен необходимы для быстрого чтения и понимания кода для программиста, который видит этот код впервые. Несмотря на большее количество символов в названиях переменных, программа, написанная по такому правилу, будет легче поддерживаться и обслуживаться.

Следующий критерий ***разрядность функций***. Под разрядностью функции понимается количество входящих переменных. В самом лучшем случае функции должны быть нульарными, т.е. не иметь входных параметров (например: «FindSumma()»), однако не всегда получается получить такую функцию. Данная разрядность функций имеет наибольшую понятность при чтении так, как нет никаких лишних слов, только название, которое говорит, что происходит. Нульарные функции компилируются с самой высокой скоростью.

Следующий разряд функции – унарный, т.е. на вход подается один параметр (например: «FindSummaTovarov(double[]massivTovarov)»). Данный разряд функции читается легко и понятно так, как видно действие и с чем оно происходит. Унарные функции компилируются медленнее чем нульарные, однако также быстро.

Бинарный разряд функции – по аналогии с предыдущими разрядами, на вход подается 2 параметра (например: «FindSredKolvoTovarov(double SummaTovarov, int KolvoTovarov)»). Бинарные функции также допускаются, но при компиляции программы занимают больше времени, чем предыдущие разряды. Бинарные функции понять сходу сложнее, чем унарные, однако бывают случаи, когда необходимо использовать бинарную функцию, например, функция вызова точки в декартовом пространстве FindPoint(0,0). Так как у точки есть две координаты, нецелесообразно менять разрядность функции, поэтому в данном примере необходимо использовать бинарную функцию.

Тернарные функции допускаются в редких исключениях. Тернарные функции самые сложные для понимания, их не рекомендуется использовать при написании кода, за редкими исключениями. Компиляция таких функций занимает наибольшее количество времени.

Последний критерий – ***количество действий в функции***. Часто программисты сталкиваются с работой с чужим кодом с сотнями и тысячами переменных и функций. Порой чтобы разобраться в работе одной функции потребуется день или даже неделя. Из-за того, что программисты записывают все действия в одну глобальную функцию, так как это ускоряет написание кода. Однако несмотря на то, что скорость написания кода и увеличивается, в то же время уменьшается понятность кода, соответственно уменьшается жизненный цикл программы. Поэтому необходимо разбивать глобальные функции на маленькие подфункции, выполняющие конкретные небольшие действия, например (подсчет суммы, количества и т.д.) Количество действий в подфункциях не должно превышать двух или трех. При чтении таких функций у стороннего программиста будет складываться понимание того, что происходит в программе. Программисту пошагово будет видна последовательность выполнения каждой подфункции. Если будут использованы критерии, описанные выше, то код будет читаться как книга. При компиляции функций с множеством действий затрачивается гораздо больше времени, нежели чем при компиляции небольших подфункций в одно – два действия.

Теперь можно составить формулу написания понятного кода:

Понятный код = {Имена переменных, Имена функций, Разряд функций, Количество действий в функции}.

Методика оценки критериев понятного кода

Методикой оценки критериев было выбрано тестирование. Были созданы два онлайн теста. В первом тесте при написании программ и функций были использованы критерии понятного кода. Во втором тесте критерии использованы не были. Для решения каждого задания в тесте необходимо было выбрать: что выведет программа либо понять, что это за функция и, что она делает.

Программы в обоих тестах были написаны на языке C#. В тестировании принимало участие 100 программистов, знакомых с языком C#. Прохождение тестов проходило по ссылке. Время прохождения и количество правильных ответов выводилось в общую статистику после каждого прохождения теста.

Критерием оценки результатов тестов было выбрано время прохождения теста, а не количество правильных ответов. Так как главной задачей является быстрое понимание кода. При этом большинство программистов, участвующих в тестировании, выполнили все задания.

Пример программы с использованием критериев понятного кода:

```
using System;
public class Program
{
   public static void Main()
   {
    double[] massiv = new double[] { 1, 2, 3, 5 };
    double SredneeMassiva=FindSredneeMassiva(massiv);
    Console.Write(SredneeMassiva);
   }
    private static double FindSredneeMassiva(double[] massiv)
    {
        int KolichestvoElementov=FindRazmerMassiva(massiv);
        double SummaElementov=FindSummaMassiva(massiv);
        double
SredneeMassiva=SummaElementov/KolichestvoElementov;
        return SredneeMassiva;
    }
     private static double FindSummaMassiva(double[] massiv)
     {
         double SummaMassiva=0;
         foreach (double element in massiv)
```

```
            {
                SummaMassiva=SummaMassiva+element;
            }
            return SummaMassiva;
        }
        private static int FindRazmerMassiva(double[] massiv)
        {
            int KolichestvoElementov=0;
            foreach (double element in massiv)
            {
                KolichestvoElementov++;
            }
            return KolichestvoElementov;
        }
    }
```

Пример программы без использования критериев понятного кода:

```
using System;
public class Program
{
    public static void Main()
    {
     double[] nums3 = new double[] { 1, 2, 3, 5 };
     double st=sw(nums3);
     Console.Write(st);
    }
     private static double sw(double[] n)
     {
         int k=0;
         double s=0;
         foreach (double x in n)
      {
          k++;
          s=s+x;
      }
      double y=s/k;
      return y;
     }
    }
```

Сравнительный график времени прохождения тестов понятного и плохого кода представлен на рис. 1.

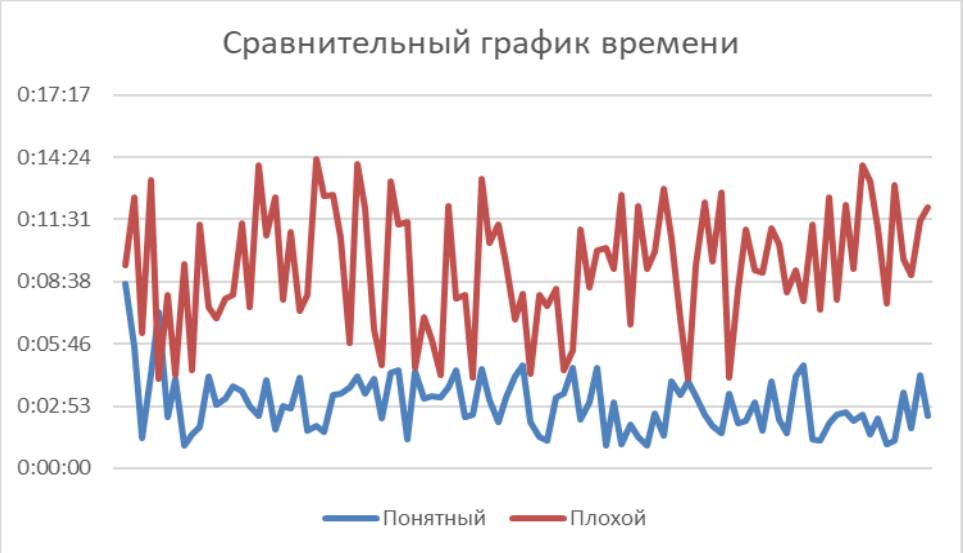

Рис. 1. График времени прохождения тестов

На графике видно, что время, затраченное для решения плохого кода больше, чем для понятного кода.

Среднее время прохождения каждого теста (рис. 2):

	Среднее
Понятный	0:02:58
Плохой	0:09:19

Рис. 2. Время прохождения теста

Таким образом можно сделать вывод, что для быстрого понимания кода необходимы дополнительные критерии, входящие в понятие понятного кода. По результатам тестов, понятный код читается и решается в три раза быстрее, чем плохой код.

Заключение

В данной статье был создан термин «понятный код.» Были описаны критерии написания понятного кода. Такие как: семантические имена переменных и функций, минимальный разряд функций, минимальное количество действий в функции. Проведен анализ критериев с помощью тестирования с участием 100 программистов. Понятный код читается в среднем в три раза быстрее.

Несмотря на то, что тестирование проводилось с программным кодом на языке C#, разработанные критерии могут быть использованы для программ, написанных на других языках программирования.

Библиографический список:

1. Горячкин Б.С., Черненький С.В., Саросек М.С. Сертификат эргономичности программного кода на языке Java//Международный научный журнал «Динамика сложных систем – XXI век»: Издательство «Радиотехника» - Москва, 2022 - № 1, С. 13-21.
2. Горячкин Б.С., Умряев Д.Т. Роль стандартов по эргономике программных средств при анализе, проектировании и оценке программного обеспечения информационных систем – Стр. 153-161. 2021г. DOI: 10.18411/lj-05-2021-123.
3. Международный стандарт ИСО 9241-12 - Эргономические требования к офисным работам с использованием видеодисплейных терминалов (VDT). Часть 12. Представление информации.
4. Актуальные проблемы психологии труда, инженерной психологии и эргономики. Выпуск 1 / под ред. В. А. Бодрова, А. Л. Журавлёва. - М.: Институт психологии РАН, 2009. - 608 с.
5. ГОСТ Р ИСО 10075-2-2009 Эргономические принципы обеспечения адекватности умственной нагрузки.
6. ГОСТ Р ИСО 26800-2013 Эргономика. Общие принципы и понятия.
7. Эргономический анализ программного обеспечения [URL: https://studbooks.net/2179766/informatika/ergonomicheskiy_analiz_programmnogo_obespecheniya] (Дата обращения: 17.11.2022).
8. Мартин Р. Чистый код: создание, анализ и рефакторинг. Библиотека программиста. – СПб.: Питер, 2013. – 464 с.
9. C# // Программирование на C, C# и Java: сайт. – URL: http://vscode.ru/category/prog-lessons/c-sharp (дата обращения: 14.10.2022).
10. Microsoft: сайт. – URL: https://docs.microsoft.com/ru-

ru/visualstudio/get-started/csharp/tutorial-console?view=vs-2022 (дата обращения: 13.10.2022).

11. Документация по C#: сайт. – URL: https://learn.microsoft.com/ru-ru/dotnet/csharp/ (дата обращения: 15.12.2022).