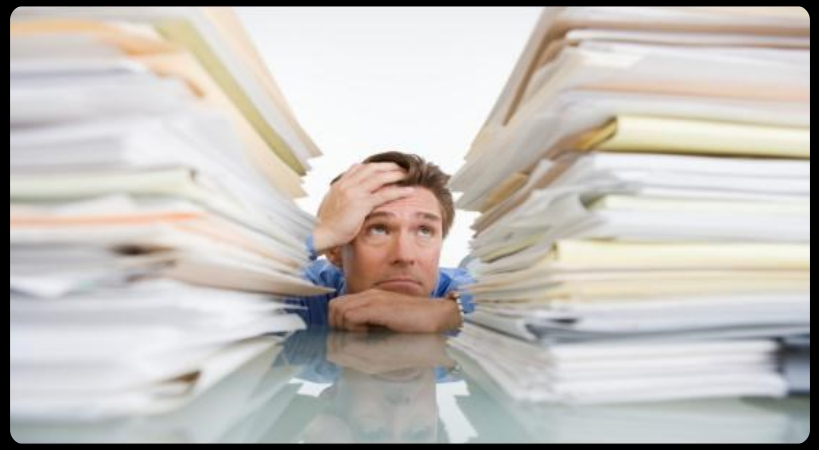
Документиране и коментиране на кода

Коментари в програмата и самоописателен код

Учителски екип

Обучение за ИТ кариера

https://it-kariera.mon.bg/e-learning/



Съдържание

1. Концепция за самоописателен код
2. Лошо написани коментари
3. Добър стил на програмиране
4. Да коментираш или не?
5. Ключови аспекти на коментирането
6. Препоръчителни практики
7. C# XML документация

Какво е проектна документация?

- Състои се от документи и информация
  - Както в програмния код, така и извън него
- Външна документация
  - На по-високо ниво от кода
  - Описания на проблема, изисквания, проектиране, дизайн, планове за проекта, планове за тестване и т.н.
- Вътрешна документация
  - На по-ниско ниво – обяснява клас, метод или част от кода

Стил на програмиране

- Основен помощник при документацията на ниво код
  - Структура на програмата
  - Ясен, лесен за четене и разбиране код
  - Добър стил на именуване
  - Изчистено оформление и формат
  - Разбираеми абстракции
  - Възможно най-малка сложност
  - Слаба зависимост и силна специализация

Лоши коментари – Пример

```
public static List<int> FindPrimes(int start, int end)
{
    // Create new list of integers
    List<int> primesList = new List<int>();
    // Perform a loop from start to end
    for (int num = start; num <= end; num++)
    {
        // Declare boolean variable, initially true
        bool prime = true;
        // Perform loop from 2 to sqrt(num)
        for (int div = 2; div <= Math.Sqrt(num); div++)
        {
            // Check if div divides num with no remainder
            if (num % div == 0)
            {
                // We found a divider -> the number is not prime
                prime = false;
                // Exit from the loop
                break;
            }
```

(продължава на другия слайд)

Лоши коментари – Пример(2)

```
            // Continue with the next loop value
        }

        // Check if the number is prime
        if (prime)
        {
            // Add the number to the list of primes
            primesList.Add(num);
        }
    }

    // Return the list of primes
    return primesList;
}
```

Самоописателен код – пример

```
public static List<int> FindPrimes(int start, int end)
{
    List<int> primesList = new List<int>();
    for (int num = start; num <= end; num++)
    {
        bool isPrime = IsPrime(num);
        if (isPrime)
        {
            primesList.Add(num);
        }
    }

    return primesList;
}
```

Добрият код не се нуждае от коментари. Той е самоописателен

(продължава на другия слайд)

Самоописателен код – пример (2)

```
private static bool IsPrime(int num)
{
    bool isPrime = true;
    int maxDivider = (int) Math.Sqrt(num);
    for (int div = 2; div <= maxDivider; div++)
    {
        if (num % div == 0)
        {
            // We found a divider -> the number is not prime
            isPrime = false;
            break;
        }
    }
    return isPrime;
}
```

Добре написаните методи имат уместни имена и са лесни за четене и разбиране

Този коментар обяснява неочевидни детайли. Не повтаря кода

Лош стил на програмиране – пример

```
for (i = 1; i <= num; i++)
{
    meetsCriteria[i] = true;
}
for (i = 2; i <= num / 2; i++)
{
    j = i + i;
    while (j <= num)
    {
        meetsCriteria[j] = false;
        j = j + i;
    }
}
for (i = 1; i <= num; i++)
{
    if (meetsCriteria[i])
    {
        Console.WriteLine(i + " meets criteria.");
    }
}
```

Неинформативни имена на променливи. Немарливо оформление

Добър стил на програмиране – пример

```
for (primeCandidate = 1; primeCandidate <= num; primeCandidate++)
{
    isPrime[primeCandidate] = true;
}

for (int factor = 2; factor < (num / 2); factor++)
{
    int factorableNumber = factor + factor;
    while (factorableNumber <= num)
    {
        isPrime[factorableNumber] = false;
        factorableNumber = factorableNumber + factor;
    }
}

for (primeCandidate = 1; primeCandidate <= num; primeCandidate++)
{
    if (isPrime[primeCandidate])
    {
        Console.WriteLine(primeCandidate + " is prime.");
    }
}
```

Самоописателен код

- Програмен код, разчитащ на добър стил на програмиране
  - Да предостави голяма част от информацията, предвидена за документацията
- Основни принципи на самоописателния код

Най-добрата документация е самият код.

Направете кода самообясняващ се и самоописателен, лесен за четене и разбиране.

Не документирайте лош код, пренапишете го!

Списък с напомняния за самоописателен код

- Класове
  - Представлява ли интерфейсът на класа завършена абстракция?
  - Ясно ли е от интерфейса на класа как ще се ползва този клас?
  - Добре именуван ли е класът? Името му описва ли целта му?
  - Може ли да третирате класа като „черна кутия“?
  - Използвате ли от много места кода (вместо да го повтаряте)?

Списък с напомняния за самоописателен код (2)

- Методи
  - Описва ли името това, което прави съответният метод?
  - Извършва ли всеки метод една-единствена добре дефинирана задача с минимална зависимост (от други методи)?
- Имена на данните
  - Имената на типовете достатъчно ли са описателни, за да подпомагат документирането на декларациите на данни?
  - Променливите ползват ли се с единствена цел? Тази цел добре дефинирана ли е?

Списък с напомняния за самоописателен код (3)

- Имена на данните
  - Конвенциите на именуване различават ли се при имената на типове, изброими типове, именувани константи, локални променливи, клас-променливи и глобални променливи?
- Други
  - Типовете данни достатъчно прости ли са, за да бъде сложността максимално ниска?
  - Свързаните команди групирани ли са заедно?

Ефективни коментари

- Ефективните коментари не повтарят кода
  - Обясняват го на по-високо ниво и разкриват неочевидни детайли
- Най-добрата софтуерна документация е самият програмен код – нека да е изчистен и четлив!
- Самоописателният код се подразбира и не се нуждае от коментари
  - Прост дизайн, малки добре именувани методи, силна свързаност и слаба зависимост, проста логика, удачни имена на променливите, добро форматиране, …

Ефективни коментари – Грешки

- Подвеждащи коментари

```
// write out the sums 1..n for all n from 1 to num
current = 1;
previous = 0;
sum = 1;
for (int i = 0; i < num; i++)
{
   Console.WriteLine("Sum = " + sum);
   sum = current + previous;
   previous = current;
   current = sum;
}
```

Какъв проблем разрешава този алгоритъм?

Можете ли да познаете коя сума е равна на i^{тото} число на Фибоначи?

Ефективни коментари – Грешки (2)

- Коментари, повтарящи кода:

```
// set product to "base"
product = base;

// loop from 2 to "num"
for ( int i = 2; i <= num; i++ )
{
    // multiply "base" by "product"
    product = product * base;
}

Console.WriteLine("Product = " + product);
```

Очевидно…

Не думай…

Ефективни коментари – Грешки (3)

- Лош стил на кода:

```
// compute the square root of num using
// the Newton-Raphson approximation
r = num / 2;
while (Math.Abs(r - (num / r)) > Tolerance)
{
    r = 0.5 * (r + (num / r) );
}
Console.WriteLine("r = " + r);
```

- Не коментирайте лош код, пренапишете го!

Ключови аспекти на ефективното коментиране

- Ползвайте стил, който не блокира промените и не им пречи

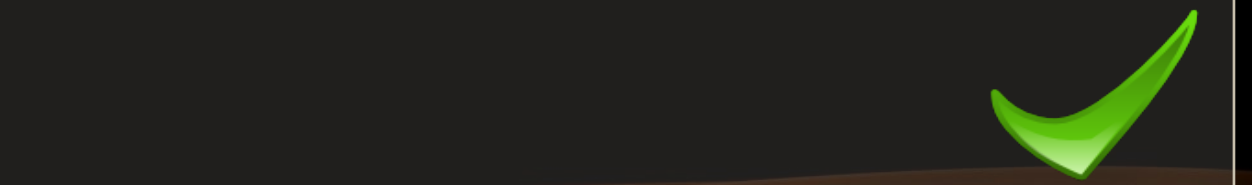
```
//  Variable              Meaning
//  --------              -------
//  xPos ................ X coordinate position (in meters)
//  yPos ................ Y coordinate position (in meters)
//  zPos ................ Z coordinate position (in meters)
//  radius .............. The radius of the sphere where the
                          battle ship is located (in meters)
//  distance ............ The distance from the start position
                          (in meters)
```

- Горните коментари са трудни за поддръжка

Ключови аспекти на ефективното коментиране (2)

- Коментирайте целта на кода, не детайли от изпълнението

```
// Scan char by char to find the command-word terminator ($)
done = false;
maxLen = inputString.length;
i = 0;
while (!done && (i < maxLen))
{
    if (inputString[i] == '$')
    {
        done = true;
    }
    else
    {
        i++;
    }
}
```



Ключови аспекти на ефективното коментиране (3)

- Фокусът на усилията ви за документиране да е върху кода

```
// Find the command-word terminator
terminatorFound = false;
maxCommandLength = inputString.Length;
testCharPosition = 0;
while (!terminatorFound && (testCharPosition < maxCommandLength))
{
    if (inputString[testCharPosition] == CommandWordTerminator)
    {
        terminatorFound = true;
        terminatorPosition = testCharPosition;
    }
    else
    {
        testCharPosition = testCharPosition + 1;
    }
}
```

По-добър код → по-малко коментари

Ключови аспекти на ефективното коментиране (4)

- Коментарите над абзаца – да обясняват „защо", не „как"

```
// Establish a new account
if (accountType == AccountType.NewAccount)
{
    …
}
```

- Използвайте коментари, за да подготвите читателя за това, което следва
- Избягвайте съкращения

Ключови аспекти на ефективното коментиране (5)

- Коментирайте всичко, което е заобиколно решение на проблем или пък е недокументирана функция
  - Напр. `// This is a workaround for bug #3712`
- Това оправдава нарушение на добрия стил на програмиране
- Ползвайте вградените в средата начини за коментиране
- Не коментирайте сложен код – пренапишете го
- Пишете документацията с помощта на инструменти
  - XML коментари

Съвети за документация от по-високо ниво

- Опишете идеята за дизайна на класа
- Опишете ограничения, изисквания за работа и др.
- Коментирайте интерфейса на класа (публични методи / свойства / събития / конструктори)
- Не документирайте детайли по изпълнението в интерфейса на класа
- Опишете целта и съдържанието на всеки файл
- Именувайте файла според съдържанието му

Оправдания и провали в документацията

- „Нямам време да пиша коментари"
  - После дешифроването на кода ще отнеме повече време
- „По-късно ще напиша коментарите"
  - Най-вероятно това няма да се случи
- „Кодът ми е самоописателен, не му трябват коментари"
  - Може да са необходими коментари, за да обяснят някоя объркваща част, да опишат структурата и поведението на приложението и др.

C# XML документация

- В C# можете да документирате кода с XML тагове в специални коментари
  - Директно в програмния код
- Например:

```
/// <summary>
/// This class performs an important function.
/// </summary>
public class MyClass { }
```

- XML doc коментарите не са метаданни
  - Не са включени в компилираното асембли
  - Достъпни са като отделен XML файл след компилирането на кода

XML тагове за документация

- `<summary>`
  - Резюме на класа / метода / обекта
- `<param>`

```
<param name="name">description</param>
```

  - Описва някой от параметрите в метод
- `<returns>`
  - Описание на връщаната стойност
- `<remarks>`
  - Допълнителна информация (бележки)

XML тагове за документация (2)

- **<c>** / **<code>** - позволява обозначаването на код
- **<see>** / **<seealso>** + **cref** - препратка към кода

```
<see cref="GetConfigurationSettings"/>
```

```
<seealso cref="TestClass.Main"/>
```

- **<exception>**
  - Позволява да уточните кои изключения може да се хвърлят

```
<exception cref="System.Exception">Thrown when...</exception>
```

- Всички тагове: http://msdn.microsoft.com/en-us/library/5ast78ax.aspx

Пример за XML документация

```
/// <summary>
/// The GetZero method. Always returns zero.
/// </summary>
/// <example>
/// This sample shows how to call the <see cref="GetZero"/> method.
/// <code>
/// class TestClass
/// {
///     static int Main()
///     {
///         return GetZero();
///     }
/// }
/// </code>
/// </example>
public static int GetZero()
{
    return 0;
}
```

C# XML документация

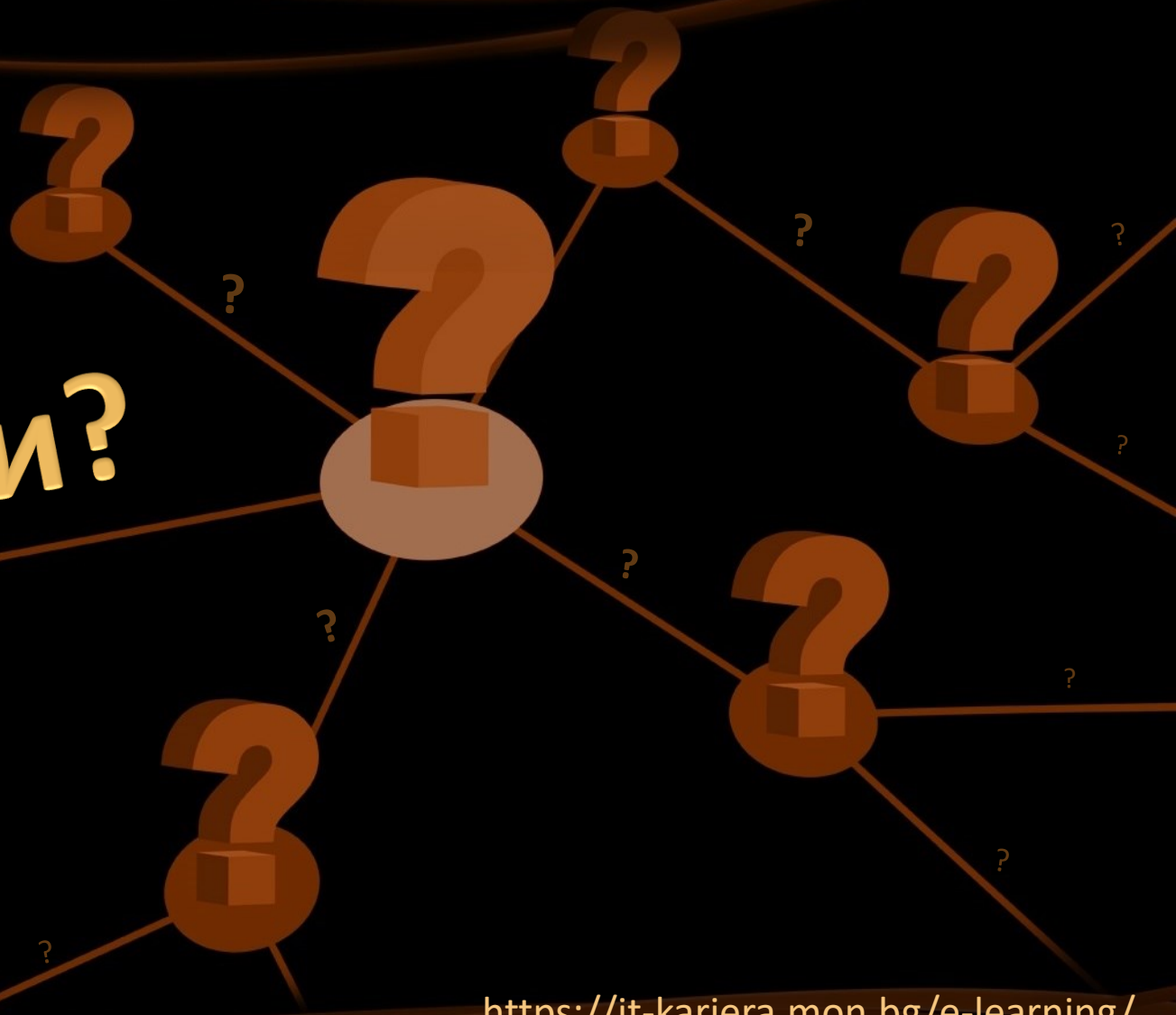
- Visual Studio ще ползва XML документацията за автоматично допълване
  - Автоматично е, просто ползва XML документите
- Компилиране на XML документацията:
  - Компилирайте кода с опция /doc, за да извлечете XML документ във външен XML файл
  - Ползвайте Sandcastle или друг инструмент да генерирате CHM / PDF / HTML / друг MSDN стил на документация
    - Пример: http://www.ewoodruff.us/shfbdocs/

Обобщение

1. Писане на коментари
2. Съвети за коментиране
   - Убедете се, че поясняват кода
   - Убедете се, че не повтарят кода
3. Самоописателен код
   - Самоописателният код не е извинение да прескочите писането на коментари където е необходимо
4. XML документация

Документиране и коментиране на кода

Въпроси?



Министерство на образованието и науката (МОН)

- Настоящият курс (презентации, примери, задачи, упражнения и др.) е разработен за нуждите на Национална програма "**Обучение за ИТ кариера**" на МОН за подготовка по професия "Приложен програмист"

- Курсът е базиран на учебно съдържание и методика, предоставени от **фондация "Софтуерен университет"** и се разпространява под свободен лиценз **CC-BY-NC-SA**