Бързо действие и оптимизация на кода

**Computer performance** is characterized by the amount of useful work accomplished by a computer system compared to the time and resources used.

Хубаво е ако например пуснем даден торент да се сваля, то това да се случва паралелно на отделно място и така да може да продължим да си работим на компютъра.

Хубаво е като пуснем дадена програма, то тя да не ни взима целия performance на процесора. Изключение може да прави дадена игра

Ако даден код не е направил конкретен проблем със своя performance, то няма нужда да се захващаме с неговата оптимизация.

Трябва да се стремим, колкото е възможно повече, сорс кода да е качествен, защото това се отплаща с времето. Много по- евтино е да се купи добър hardware (който да подобри performance- а), отколкото да се наемат програмисти, които да напишат качествен софтуер в продължение на месеци.

При web приложенията, качеството на кода трябва да е преди бързината/performance- а му. Но ако пишем софтуер за embedded приложения, то тогава е обратното – performance- а трябва да е на първо място.

Не е задължително вярно, че по- голямото парче код е по- бавно от по- малкото парче код.

Една програма трябва на първо място да е коректна!

Обикновено оптимизирането на кода влошава неговото качество

**Когато пишем код, на първо място трябва да се стремим той да е качествен и коректен. Чак след това трябва да го оптимизираме!**

Много пъти има конфликт между бързото действие на кода и неговото качество. В повечето случаи трябва да наблягаме първо на качеството на кода, но в някои случаи трябва да се съсредоточаваме върху бързото действие на кода

dotTrace – инструмент за проследяване времето, за което се изпълнява програмния код

Може да използваме някаква програма напр. dotTrace, когато не знаем къде ни се бави програмата или когато искаме цяла картина на performance- а на програмата

Ако искаме да видим само колко бързо се изпълнява даден метод, то тогава е добре да използваме Stopwatch класа.

!string- овете са read-only (immutable). Т.е. когато присвояваме даден стринг на друг стринг, то се създава нов трети стринг

! Един T[] масив е винаги по- бърз от един лист List<T>. Това е така защото листа използва отдолу масив за да функционира.

Ако в един for- цикъл се проверява условието i < str.Length, то това е по- бавно в сравнение с това ако изведем този str.Length в отделна променлива, тъй като при първия случай, на всяка итерация се извиква str.Length property- то. Т.е. :

int count = 0;

for (int i = 0; i < str.Length; i++)

е по-бавно от:

int count = 0;

int len = str.Length;

for (int i = 0; i < len; i++)

В повечето случаи bottleneck-ците се получават поради използване на неподходящите структури от данни или при грешни алгоритми, а не поради някакви незначителни разлики в скоростта, които се получават поради например случаи като горния

Using an inappropriate data structure or algorithm is often the reason for a bottleneck!

**Извод: трябва да оптимизираме кода, само когато има такава нужда!**