# Какво е качествен програмен код?

Писането на качествен код е важна част от работата на всеки програмист, защото така кода става по-четим, по-лесен за поддръжка и дебъгване. При добре написан код няма да има проблеми някой друг програмист да вземе кода и да разбере какво прави.

Всеки софтуер има две страни, които трябва да разгледаме. Имаме външно качество и вътрешно качество.

Външното качество на софтуера се изразява в отговорът на няколко въпроса.

1. Програмата държи ли се коректно?
2. Резултатът, който връща програмата, коректен ли е?
3. Работи ли бързо програмата?
4. Лесна ли е за ползване програмата от потребителите?
5. Добре защитен ли е кодът на програмата?

Ако отговорът на всички тези въпроси е положителен, следва че ние и нашият екип сме си свършили работата от страната на Външното качество (external quality) на софтуера.

Както споменах преди малко има и вътрешно качество (internal quality) на кода. Ако външното се отнасяше по-скоро до потребителя, то вътрешното е насочено към програмистите. Тук също има няколко въпроса, на които е нужно да си отговорим.

1. Кодът, който пишем лесен ли е за четене и разбиране?
2. Добре структуриран ли е кодът?
3. Лесен за промяна ли е кодът?

Досещате се, че ако и тук сме отговорили с „да“, то ние сме най-добрите и сме си свършили прекрасно работата. Но това не се случва много често в реалния живот.

Ето защо трябва да се стремим да спазваме някои правила докато пишем код, за да можем да си улесним живота в последствие, когато след време се върнем към кода и трябва да го променим.

Сега ще изброя част от нещата, които можем да направим, за да пишем качествен програмен код и след това ще кажа по няколко думи за всяко.

- Лесно четим и разбираем код
- Коректно поведение на методите, функциите, класовете
- Добра йерархия на кода съчетана с добър дизайн(на кода)
- Добра документация – самодокументиращ се код
- Добро именуване
- Добро форматиране
- Strong cohesion
- Loose coupling

**Лесно четим и разбираем код**

Това за мен е доста общо понятие и съчетава в себе си всичко останало. Т.е. ако спазваме останалите правила в крайна сметка ще получим един лесен за четене и разбиране код.

**Коректно поведение на методите/функциите, класовете**

Тук става въпрос да не позволяваме на дадена част от кода ни да връща грешен резултат. Много е важно да тестваме и дебъгваме добре нашето приложение. Другата страна на тази точка е да пишем така своя код, че да можем да преизползваме възможно най-много от него. Може би само се досещате, че ако пишем един и същ код на няколко места в приложението си и после решим да променим нещо малко трябва да обходим всички редове и да ги променим където ни е необходими. Тук опасността е, че можем да изпуснем някой ред и да се объркa всичко. Ето защо трябва да изнасяме повтарящия се код в методи/функции. Така Ще спазваме и DRY (Don’t repeat yourself).

**Добра йерархия на кода съчетана с добър дизайн(на кода)**

Тази точка от листа със съветите до някъде се приближава до предходната, защото ако сме направили добър дизайн и добра йерархия на кода си, ще можем да преизползваме голяма част от него. Предварително ще сме забелязали кое би се повтаряло в приложението и ще можем да го изнесем в отделен метод/функция. Ще можем да направим добра йерархия на класовете си, за да можем да програмираме по-лесно. Ако клас-йерархията ни е зле направена това ще ни струва много нерви и време. Ето защо не трябва да пренебрегваме и тази точка.

**Добра документация – самодокументиращ се код**

Документацията отново има две страни. Една за потребителя и една за програмиста.

Техническата документация показва кой метод какво върши, какви параметри получава и какво връща. Включва в себе си и клас-йерархията и кой клас какви пропъртита, конструктори и методи има в себе си. Всичко необходимо на един програмист, който не е виждал и не е писал даден код да може да се ориентира бързо и лесно какво прави и как го прави.

Самодокументиращият се код е код, който е така написан, че няма нужда от коментари. Имената на методите и променливите сами казват какво правят или съдържат. Не е нужно изрично да пишем коментар, който да обяснява какво се случва. Естествено понякога се налага, защото използваме някой алгоритъм или нещо подобно, но в повечето случай е достатъчно да кръщаваме добре своите променливи и останалите части от кода.

**Добро именуване**

Доброто именуване не се отнася само за променливите. Необходимо е да именуваме описателно и методите/функциите, класовете, структурите и всичко останало. Ако използваме „I,j,k“ като имена в нашия код ще ни е доста трудно, пък и не само на нас, да се

оправим в кода. Ако е от 10-15 реда няма да имаме проблем, но щом стане повече нещата загрубяват. Естествени има и случаи в, които е прието да се използват отделни букви за имена на променливи. Ако имаме for-цикъл и той е сравнително кратък няма никакъв проблем да използваме „i“ или “j”.

**Добро форматиране**

Доброто форматиране помага кодът да стане четим за нас програмистите и да можем бързо да се ориентираме в него. Използването на IDE(Integrated Development Environment) облекчава в голяма степен работата по тази точка. Всички те са достатъчно умни, за да знаят какво пишем ние и да го форматират по правилния начин съобразен с конвенциите на съответния език.

**Strong cohesion и Loose coupling**

Тези две концепции са свързани с работата ни с методи/функции и класове. Strong cohesion изисква всяка една част или модул от нашия код да върши само едно нещо и нищо повече. Не би трябвало методът, в който смятаме средният успех на студент, да му изпраща и писмо, което му честити имения ден. Тези действия трябва да са разделени в два метода.

Loose coupling пък изисква различните части от кода ни да не са тясно свързани с останалите. По този начин можем лесно да вземем едно парче код и да го сложим в друг наш проект и то да тръгне без големи промени по него.

Това бяха част от добрите практики при писането на качествен програмен код. Очаквайте скоро статии на същата тема, в които ще навлезем по-дълбоко в нещата.

Източници: http://academy.telerik.com/student-courses/programming/high-quality-code/about