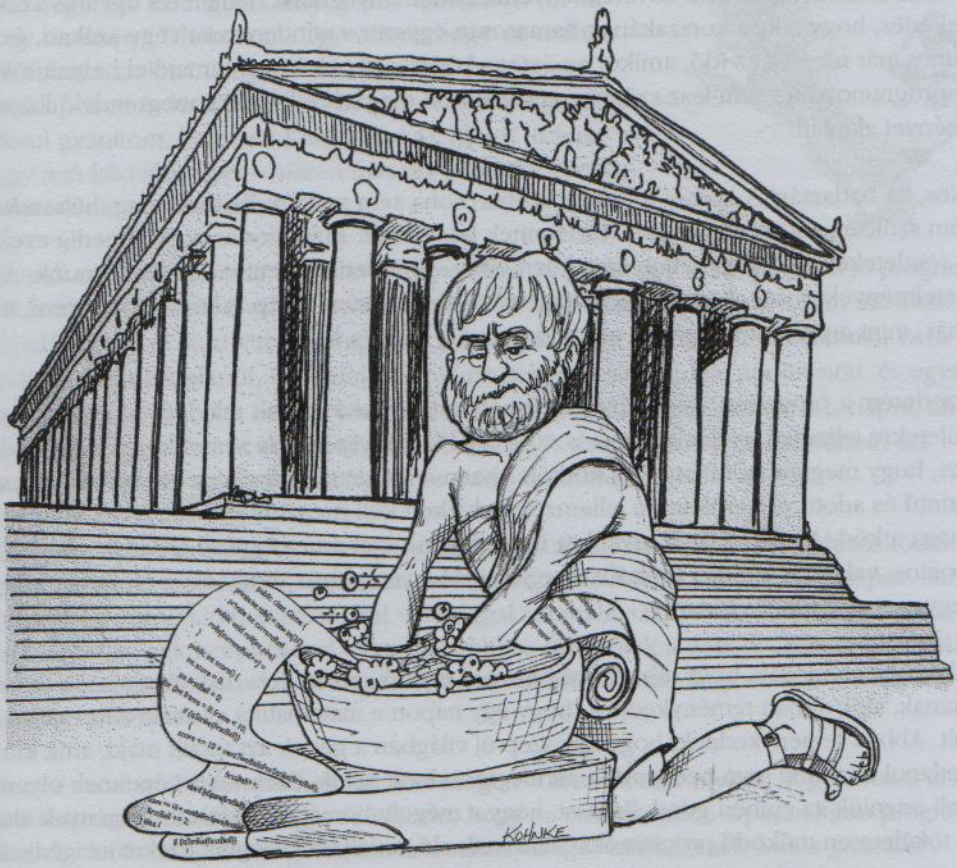
1

Tiszta kód



Két okból is olvashatjuk ezt a könyvet. Először is, mert programozók vagyunk. Másodszor pedig, mert jobb programozóvá szeretnénk válni. Nagyszerű. A világ epekedve várja a jobb programozókat!

Könyvünk a helyes programozásról szól, és mint ilyen, telis-tele van kóddal. A kód számunkra egy szobor, amelyet gondosan körüljárunk, és kritikus szemmel vizsgálgatunk minden lehetséges oldalról. Végigmérjük a tetejétől az aljáig, az aljától a tetejéig, sőt fel is boncoljuk. Ha pedig mindezen túlestünk, a kelleténél talán kicsit többet is tudunk majd a kódról – de ami még fontosabb, el tudjuk különíteni a jó kódot a rossztól. És arra is képesek leszünk, hogy a gondosan összeállított eszköztárunk segítségével jó kódot írjunk, és a rossz kódból jó kódot faragjunk.

A kód örök

Vannak, akik úgy tartják, hogy a kóddal foglalkozni idejétmúlt dolog – sokkal inkább érdemes a modelleket és a követelményeket tanulmányoznunk. Ismeretes ugyanis az a vélekedés, hogy a kód korszakának hamarosan egyszer s mindenkorra vége szakad, és nincs már messze az idő, amikor az összes kódot a gépek készítik majd el helyettünk. Így a programozókra sem lesz szükség, mivel a kód magától elkészül a megrendelő követelményei alapján.

Nos, ez badarság! A kódtól teljes egészében soha sem szabadulhatunk meg, hiszen kódra van szükség a követelmények részleteinek leírásához. Bizonyos szinteken pedig ezeket a részleteket nem hagyhatjuk figyelmen kívül – mindenképpen le kell őket írnunk. A követelmények olyan részletes meghatározása, amit a számítógép is képes megérteni, nem más, mint a programozás – az eredmény pedig maga a kód.

Szerintem a programozási nyelvek egyre elvontabbakká válnak majd, és az egyes részterületekre jellemző nyelvek száma is nőni fog. Ez nagyszerű, de semmiképpen sem jelenti azt, hogy megszabadulhatunk a kódtól, hiszen a meghatározásainkat ezeken a magasabb szintű és adott részterületekre jellemző nyelveken kell megadnunk, vagyis most ez lesz maga a kód. A megírásánál továbbra is ügyelnünk kell arra, hogy az eredmény szikár, pontos, valamint kellően formális és részletes legyen ahhoz, hogy egy számítógép képes legyen megérteni és futtatni.

Akik úgy gondolják, hogy a kód egyszer majd eltűnik, azokhoz a matematikusokhoz hasonlítanak, akik abban reménykednek, hogy egy napon a matematika elveszíti a formális jellegét. Abban reménykednek, hogy egy szép új világban a gépek azt teszik majd, amit kimondatlanul akarunk, nem pedig azt, amit megmondunk nekik. Ezeknek a gépeknek olyan jól kell érteniük az emberi gondolkodást, hogy a mégoly homályosan felvázolt igények alapján is tökéletesen működő programokat állítsanak elő, amelyek kielégítik ezeket az igényeket.

Ez az idő azonban soha sem jön el. Még az intuícóval és kreativitással megáldott ember sem volt képes olyan rendszereket létrehozni, amelyek a megbízó homályos érzései alapján képesek a megfelelő eredményt visszaadni. Sőt, ha valamit megtanultunk a követelmények megadásának szabályaiból, az az, hogy a megfelelően meghatározott igények éppen olyan formálisak, mint valamiféle kód, emellett a kód tesztjeiként is alkalmazhatók.

Végeredményben elmondhatjuk, hogy a kód nem más, mint az a nyelv, amelyen az igényeinket kellően pontos alakban megadhatjuk. Létrehozhatunk olyan nyelveket, amelyek eleve közelebb állnak az igényekhez. Készíthetünk eszközöket, amelyekkel az igényeket elemezhetjük és formális rendszerbe állíthatjuk. A kellő pontosságról azonban soha nem mondhatunk le – így a kód örökké velünk marad.

A rossz kód

Nemrég olvastam Kent Beck *Implementation Patterns*[1] című könyvének előszavát, amelyben így fogalmaz: „...ez a könyv egy meglehetősen törékeny feltételezésre épül: arra, hogy a jó kód igenis lényeges..." *Törékeny?* Mi az, hogy törékeny? Tiltakozom! Úgy vélem, hogy az említett feltételezés szakmánk egyik legszilárdabb, legjobban megalapozott és legtöbb irányból igazolt feltételezése (ráadásul gyanítom, hogy ezt Kent is tudja). Igenis tudjuk, hogy a jó kód lényeges – hiszen már oly hosszú ideje szenvedünk a hiánya miatt.

Ismerek egy céget, amely a nyolcvanas években készített egy *irgalmatlanul jó* programot, ami hamar népszerűvé vált, és rengeteg profi kezdte használni. Egy idő után azonban a kiadások egyre feszítettebb tempóban kezdték követni egymást, a hibákat már nem javították ki maradéktalanul, így halmozódni kezdtek, a betöltés ideje nőttön-nőtt, és egyre gyakrabban fordult elő, hogy a program lefagyott. Tisztán emlékszem arra a napra, amikor végül elegem lett az egészből, kikapcsoltam, és azontúl undorral elkerültem ezt a szörnyűséget. Nem telt bele sok idő, és a cég is tönkrement...

Húsz évvel később összefutottam a cég egyik régi alkalmazottjával, és kérdezősködni kezdtem: mi történt? Nos, a félelmeim utólag beigazolódtak. A cég engedett a piac sürgetésének, és az idő előtti kiadásokkal valósággal romhalmazzá tették a kódot. Az újabb és újabb lehetőségek beépítésével egyre mélyebbre süllyedtek a dagonyában, míg végül fenntarthatatlanná vált a kód. A *rossz kód volt az oka annak, hogy ez a jobb sorsra érdemes cég tönkrement.*

Mindez azonban bizonyára nekünk sem idegen – éreztük már olykor, hogy a rossz kód hátráltatja a munkánkat. Meggyőződésem, hogy az érzés minden tapasztaltabb programozónak ismerős. Még neve is van ennek a szörnyűséges kínlódásnak – ez a *gázolás.*
Ez a szó fájdalmasan nagyszerűen írja le mindazt a szenvedést, ahogy keresztülesszük magunkat a rossz kódon, próbáljuk áttörni az indák dzsungelét, és elkerülni a félhomályban

[1] [Beck07].

váratlanul megjelenő alattomos vermeket. Próbálunk kikecmeregni a labirintusból, segítséget, egy apró reménysugarat várunk, hogy legalább valami halvány képet alkossunk arról, mi is folyik itt – de csak mélyebbre és mélyebbre jutunk a kód sötét őserdejében.

Hát persze, hogy volt részünk a rossz kód „élményében" – de akkor… miért írtuk meg így?

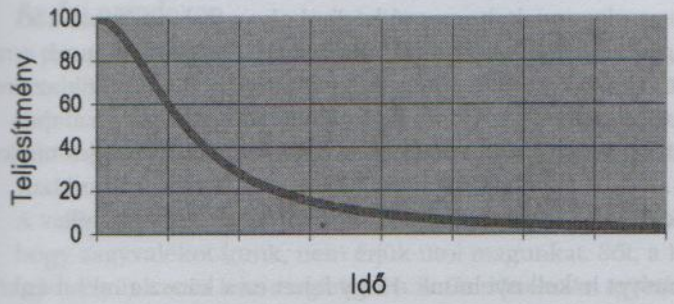
Gyorsan kellett elkészülnünk? Siettünk valahová? Valószínűleg igen. Talán úgy éreztük, nincs időnk a tisztességes munkára, és azt gondoltuk, hogy a főnökünk eltorzult arccal üvöltené le a fejünket, ha még több időt kérnénk a kód tisztogatására. Vagy megeshet, hogy csak elegünk volt már a programból, és azt kívántuk, hogy minél előbb elkészüljünk. Az is lehet, hogy más programozási ígéreteket is teljesítenünk kellett, és úgy véltük, nem lehet nagy gond abból, ha ezt a modulocskát most csak úgy összecsapjuk. Igen, mindannyian vétkeztünk már…

És mindannyian eljutottunk arra a pontra, amikor undorral néztünk végig saját torzszülöttünkön, és úgy döntöttünk, várhat még egy napot, amíg gatyába rázzuk. Ismerős, ugye, az a nagyszerű érzés is, amikor megkönnyebbülten látjuk, hogy a torzszülött így vagy úgy, de működik – és egy működő torzszülött még mindig jobb, mint a semmi. Persze, később visszatérünk, és rendbe tesszük a dolgokat. Nos, valószínűleg akkoriban még nem volt ismeretes LeBlanc egyszerű, de fájdalmasan helytálló törvénye: A *„később" egyenértékű a „soha" fogalmával.*

A zagyvalék teljes fenntartási költsége

Ha már legalább két-három éve programozással keressük a kenyerünket, bizonyára előfordult, hogy komolyan megszenvedtük mások zavaros kódjának hatását. Márpedig e hatás igencsak számottevő lehet – olyannyira, hogy egy rohamtempóban nekiinduló fejlesztőcsapat munkája egy-két év alatt csigalassúságúvá válhat. Minden módosítás kiüt két-három másik kódrészletet, és egyetlen változtatás sem egyszerű. Minden újabb lehetőség beépítéséhez, illetve minden módosításhoz meg kell „értenünk" az indák, folyondárok és göcsök rendszerét – és minden lépésünkkel tovább bonyolítjuk az amúgy is átláthatatlan szövedéket. Végül olyan kiterjedt és áthatolhatatlan dzsungelhez jutunk, amelyet nem vagyunk képesek megtisztítani. Nem csoda, hiszen ezen a ponton erre már nincs is mód.

Ahogy a zagyvalék egyre sűrűbbé válik, úgy csökken a fejlesztőcsapat teljesítménye, a végtelenben a nullához közelítve. Észlelve mindezt a vezetőség azt teszi, amire csak képes: további munkatársakat irányít a projekthez, remélve, hogy ezzel növelheti a teljesítményt. Az új emberek azonban nem ismerhetik a rendszer alapszerkezetét, így azt sem tudhatják, hogy melyik változtatás igazodik a program „lelkéhez", és melyik tökéletesen ellentétes azzal. Ráadásul, ha mindez nem lenne elég, az újak és a régiek egyaránt hihetetlen nyomást éreznek magukon a teljesítmény mielőbbi növelésére. Következésképpen egyre sebesebb tempóban süllyesztik az egész projektet a mocsárba, egyre csak csökkentve a teljesítményt. (Lásd az 1.1. ábrát.)



1.1. ábra

Teljesítmény az idő függvényében

A Nagy Újjáépítés

Elérkezik a pillanat, amikor a csapat fellázad. Jelzik a vezetőségnek, hogy képtelenek folytatni a fejlesztést ezzel a szörnyű kódalappal – és újratervezést követelnek. A vezetőség túl nagy áldozatnak látja a teljes projekt újratervezését, de azt elismeri, hogy a teljesítmény pocsék. Végül meghajlik a fejlesztői nyomás alatt, és zöld utat ad az újjáépítésnek.

A cég kijelöl egy új csapatot, amelyhez minden munkatárs szívesen csatlakozna, hiszen itt egy „zöldmezős" projektről van szó, és mindenki boldog, ha a nulláról valami csodálatos dolgot építhet fel. Mindazonáltal ebbe az elit alakulatba csak a legjobbak kerülhetnek be, a többiek tovább küzdenek a lövészárkokban az eredeti projekt fejlesztésével.

A következőkben verseny alakul ki a két csapat között. Az elit alakulatnak olyan rendszert kell készítenie, ami biztosítja a régi lehetőségeit, ráadásul lépést kell tartania az időközben bekövetkező változásokkal is. A vezetőség pedig mindaddig nem adja át a régi rendszer helyét az újnak, amíg az elit alakulat utol nem éri a „régieket".

Ez a verseny igen sokáig eltarthat; jómagam is tanúja voltam egy tíz éves küzdelemnek. Ennyi idő alatt az elit alakulat eredeti tagjai jó eséllyel már sehol sincsenek, az utódjaik pedig lassan követelni kezdik az új rendszer újratervezését, mert azt látják, hogy elhatalmasodott rajta a káosz.

Ha csak kis részben is megéltük a fent leírt történetet, láthatjuk, hogy a kód tisztán tartására fordított idő nem csak a költséghatékonyságot, de a szakmai túlélésünket is szolgálja.

Hozzáállás

Gázoltunk már olyan mélyen a kód mocsarában, hogy hetekig tartott egy olyan módosítás, amelyet egyébként pár óra alatt elvégeztünk volna? Volt már részünk olyan helyzetben, amikor tisztán láttuk, hogy egy egysoros változtatás mindent megoldott volna, mégis modulok százaihoz nyúltak hozzá? Ezek bizony igen gyakori tünetek.

Miért történik ez a kóddal? Mi az oka annak, hogy a jó kódból olyan gyorsan rossz kód válik? Sokféleképpen megmagyarázhatjuk magunknak a helyzetet. Panaszkodhatunk arra, hogy a követelmények annyit változtak az idők során, hogy szembe kell mennünk az eredeti tervekkel. Búslakodhatunk a határidők rövidségén, amelyek meggátolták az alapos munkát. Kifakadhatunk az ostoba vezetőkre, az érzéketlen ügyfelekre, a felesleges marketingesekre és telefonmosókra. A hiba azonban, kedves Dilbert, nem a csillagzatunkban, hanem saját magunkban rejlik. Be kell látnunk, hogy szakmailag gyengék vagyunk.

Igen, ez egy keserű pirula, amelyet le kell nyelnünk. Hogy lehet ez a káosz a mi hibánk? És mi van a követelményekkel? A határidőkkel? Az ostoba vezetőkkel és a felesleges marketingesekkel? Talán ők nem részesek a felelősségben?

Nem. A vezetők és a marketingesek tőlünk várják a tájékoztatást, amelynek alapján elkötelezik magukat és ígéreteket tesznek – de még ha nem is figyelnek ránk, nekünk akkor is elég hangosnak kell lennünk, hogy a gondolataink eljussanak hozzájuk. A felhasználók tőlünk várják, hogy megtervezzük, miként kerüljenek be a rendszerbe a követelmények. A projektvezetők a segítségünket várják a fejlesztés ütemezéséhez. Így hát magunk is mélyen benne vagyunk a rendszer tervezésében, vagyis komoly felelősségünk van a hibák bekövetkeztében, különösképpen a rossz kódot tekintve.

„Várjunk csak!" – mondanánk – „Ha nem tesszük meg, amit a főnök kér, minden bizonnyal kirúg!" Ez egyáltalán nem valószínű. A legtöbb vezető hallani szeretné az igazságot, még akkor is, ha nem úgy tűnik. A többségük jó kódot szeretne, még ha igen megszállottan védelmezi is az eredeti ütemtervet. Védje csak lelkesen a követelményeket és a határidőket – ezt a dolga! A mi dolgunk viszont az, hogy ugyanilyen szenvedéllyel megvédjük a kódot.

Képzeljük magunkat egy orvos helyébe, akit a betege arra kér, hogy hagyja a fenébe azt a hosszadalmas kézmosást a műtét előtt, kezdjen már hozzá végre a munkához.[2] Jóllehet nyilvánvalóan a beteg a „főnök", ez esetben az orvosnak nem szabad engednie. Miért? Nos, az orvos sokkal, de sokkal többet tud a betegénél a fertőzés veszélyéről. Ha engedne, igencsak szakmaiatlan (sőt büntetendő) döntést hozna.

Érthető tehát, hogy miért szakmaiatlan, ha egy programozó meghajlik a főnöke akarata előtt, akinek fogalma sincs róla, hogy milyen kockázatokkal jár, ha a kód megindul a dzsungellé válás útján.

[2] A kézmosást elsőként Semmelweis Ignác javasolta az orvostársadalomnak 1847-ben, de ellenállásba ütközött: a kollégák azt mondták, nincs annyi idejük, hogy két beteg között kézmosással foglalkozzanak.

Az ősi paradoxon

A programozóknak alapértékek ütközésével kell szembenézniük. Minden fejlesztő, aki rendelkezik már pár éves tapasztalattal, tudja, hogy a korábbiakban létrejött zavaros kódok lassítják a munkát. Mégis, úgy érzik, hogy a határidők nyomása alatt nekik is zavaros kódot kell előállítaniuk. Röviden: nem szánnak rá elég időt, hogy gyorsabbak lehessenek.

A valódi profi programozók tudják, hogy a fenti paradoxon második része hamis. Attól, hogy zagyvalékot írunk, nem érjük utol magunkat. Sőt, a kódmocsár azonnal visszahúz, így esélyünk sem lesz megfelelni a határidőnek. Az *egyetlen* lehetséges kiút – az egyetlen mód a gyors haladásra – az, ha a kódunkat mindig a lehető legtisztábban tartjuk.

A tiszta kód művészete

Tételezzük fel, hogy elfogadtuk: a zavaros kód komolyan hátráltatja a munkánkat. Belenyugodtunk abba is, hogy csak úgy tudunk tisztességesen haladni, ha a kódunkat tisztán tartjuk. Ezek után fel kell tennünk magunknak a kérdést: „Hogyan készítsük el ezt a bizonyos tiszta kódot?" Semmi értelme úgy próbálkoznunk, hogy azt sem tudjuk, mit jelent az, hogy „tiszta".

Rossz hírt kell közölnünk: tiszta kódot készíteni olyasmi, mint festményt festeni. A legtöbbünk meg tud különböztetni egy rossz festményt egy jótól – ez a felismerés azonban még nem tesz alkalmassá arra, hogy magunk is jó festményeket készítsünk. Ha tehát tisztán látjuk is a különbséget a tiszta és a zavaros kód között, még hosszú út vezet a tiszta kód megírásáig.

Ahhoz, hogy erre képesek legyünk, egy seregnyi apró fogást kell alkalmaznunk, és az eredményt a „tisztaságérzékünk" szemüvegén keresztül kell ellenőriznünk. Ez a „kódérzék" a sikeres munka kulcsa. Vannak, akik a születésüktől fogva rendelkeznek ezzel az érzékkel, másoknak viszont meg kell küzdeniük érte. Ha azonban eljutottunk idáig, nem csak a jó és a rossz kód közötti különbséget tudjuk majd felismerni, de nyomban felsejlik a lelki szemeink előtt egy alkalmas stratégia, amelyet követve a rossz kódból jót faraghatunk.

A „kódérzék" nélküli programozó megáll egy elmocsarasodott modul előtt, látja a súlyos gondokat, de teljességgel tehetetlen. Ugyanakkor „kódérzék" birtokában lehetőségek és változatok jelennek meg a szemünk előtt – ez az érzék vezeti a kezünket, amikor a program viselkedését megőrző változatok során keresztül eljutunk a kívánt célállapothoz.

Röviden, a tiszta kódot készítő programozó egy művész, aki az üres képernyőből átalakítások sorával végül egy elegánsan kódolt rendszerhez jut.

Mi is az a tiszta kód?

A fogalomnak minden valószínűség szerint annyi meghatározása van, ahány programozó csak létezik a világon. Ezért hát megkértem néhány ismert és nagy tudású programozót, fejtsék ki, mi az ő véleményük ebben a témában.

Bjarne Stroustrup, a C++ atyja, a *The C++ Programming Language* című könyv szerzője

Azt szeretem, ha elegáns és hatékony a kód, ami kikerül a kezem alól. A logikai felépítése legyen egyszerű, így a hibák nehezen bújhatnak meg a sarkokban, a függőségek számát igyekszem a lehető legkisebbre csökkenteni a fenntarthatóság érdekében, a hibakezelést a kijelölt stratégia szerint teljessé teszem, továbbá ügyelek arra, hogy a program teljesítménye közel optimális legyen, így nem csábít arra másokat, hogy zavarossá tegyék holmi fegyelmezetlen finomhangolással. A tiszta kód egyetlen feladatot valósít meg, azt azonban hatékonyan teszi.

Bjarne az „elegáns" szót használja. Micsoda szó! A MacBook® gépem szótára ezt a meghatározást adja rá: *„kellemesen kecses és stílusos megjelenésében, illetve viselkedésében; kellemesen ötletes és egyszerű".* „Kellemesen". Bjarne szerint a tiszta kód kellemes olvasási élményt ad – vagyis azt érezzük, amit egy iparos mesterműve vagy egy jól megtervezett autó láttán: ösztönösen mosolyra görbül a szánk.

Bjarne emellett *kétszer* is szóba hozza a hatékonyságot. Ez talán nem meglepő a C++ kiötlőjétől, de úgy vélem, itt nem csupán a sebességvágyról van szó. A processzorciklusok elpocsékolása nem elegáns; nem kellemes együtt élnünk ezzel a tudattal. És figyeljük meg, milyen szót használ az elegancia hiányának következménye kapcsán – *csábítás!* A rossz kód a zavar fokozására *csábít!* Ha mások módosítanak egy rossz kódot, az jellemzően még rosszabbá válik.

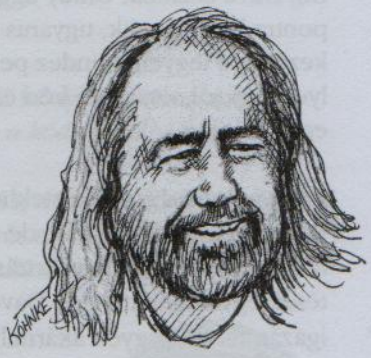
Dave Thomas és Andy Hunt, a pragmatikus programozók, másként fogalmaznak – a betört ablak metaforáját tartják alkalmasnak a helyzet leírására.[3] Egy épület, ami betört ablakkal áll előttünk, azt a képet sugallja, hogy nem törődnek vele. Az emberek pedig ennek megfelelően elhanyagolják. Így lassacskán több törött ablakot is látunk, sőt egy idő után már szinte csábító lesz az ablakok betörése. Hamarosan megjelennek a falfirkák a homlokzaton, és kezd felhalmozódni a szemét a házban. Így indíthat el egyetlen törött ablak egy épületet a romlás útján.

[3] http://www.pragmaticprogrammer.com/booksellers/2004-12.html

Bjarne szót ejt arról is, hogy a hibakezelésnek teljesnek kell lennie. Elérkeztünk egy újabb parancsolathoz, amely szerint kínos alapossággal kell ügyelnünk a részletekre. A rövidre fogott hibakeresés a programozói felületesség egyik formája. Ide tartozik azonban a memóriaszivárgás, a versenyhelyzetek figyelmen kívül hagyása és a nem megfelelő elnevezések használata is. Röviden összefoglalva: a tiszta kód különös figyelmet fordít az apró részletekre.

Bjarne zárómondatában arra hívja fel a figyelmet, hogy a tiszta kód egyetlen feladatot végez el, azt azonban kiválóan teszi. Nem véletlen, hogy a programtervezés számos szabályát erre a fontos tanulságra lehet visszavezetni, amint azt számos szerző hangsúlyozza a témában született írásaiban. A rossz kód túl sokat markol – a mögötte álló szándék zavaros, a rendeltetése pedig egyáltalán nem világos. A tiszta kód *összpontosít*. Minden függvény, osztály és modul pontosan meghatározott célt szolgál, amit nem homályosítanak el a külső körülmények.

Grady Booch, az *Object Oriented Analysis and Design with Applications* szerzője



A tiszta kód egyszerű és közvetlen. A tiszta kód úgy olvasható, mint egy jól megírt prózai mű. A tiszta kód sosem fedi el a tervező szándékait – éppen ellenkezőleg: absztrakciói világosak, vezérlési ágai pedig egyértelműek.

Grady bizonyos tekintetben megismétli Bjarne gondolatait, de inkább az *olvashatóság* irányából közelít. Különösképpen tetszik az a megállapítása, miszerint a tiszta kódnak éppoly olvasmányosnak kell lennie, mint egy jól megírt prózának. Gondoljunk csak vissza a legutóbbi igazán jó könyvre, amit olvastunk. Idézzük fel, miként tűntek el a szavak, hogy nyomban képeknek adják át a helyüket. Ugye olyan volt, mintha egy filmet néznénk? Magunk előtt láttuk a szereplőket, hallottuk a hangokat, és átéltük a humort és a fennköltséget egyaránt.

Persze a tiszta kód olvasása sohasem adhat olyan élményt, mintha *A gyűrűk urát* lapozgatnánk, de a hasonlat így sem rossz. A jó regényekhez hasonlóan egy program is tisztán visszaadhatja a megoldandó feladatban rejlő feszültségeket. A feszültségek pedig halmozódnak, mígnem elérünk egy csúcspontot, ahol az olvasó ösztönösen így kiált fel: „Aha! Hát persze!" – és az összes feszültség feloldódik egy ezen a ponton már nyilvánvaló megoldásban.

Magam is elcsodálkoztam, milyen meghökkentő önellentmondás rejlik a „világos absztrakció" szókapcsolatban. A szavak látszólagos szembenállásának ellenére azonban ez a szókapcsolat mégiscsak fontos üzenetet hordoz. A kód eszerint legyen lényegre törő, és csak azt tartalmazza, amire feltétlenül szükség van, aki pedig a kódunkat olvassa, érezze úgy, hogy képesek voltunk meghozni a szükséges döntéseket.

„Big" Dave Thomas, az OTI alapítója, az Eclipse stratégia keresztapja

A tiszta kódot a szerzőjén túl más fejlesztők is képesek olvasni és kiegészíteni. Rendelkezik egység- és elfogadási tesztekkel, értelmes neveket használ, és egy adott feladat megoldására több lehetséges mód helyett egyetlenegyet biztosít. A tiszta kód emellett minimális függőséget tartalmaz, amelyek világosan azonosíthatók, továbbá tiszta és minimális API-t biztosít. Végezetül fontos, hogy a kód olvasmányos legyen, hiszen a nyelvtől függően nem minden szükséges információ adható át pusztán a kód által.

Big Dave osztozik Grady aggodalmában az olvashatóságot illetően, de kicsit más nézőpontra helyezkedik, ugyanis azt tartja lényegesnek, hogy a tiszta kód *mások* számára is jól kezelhető legyen. Mindez persze nyilvánvalónak tűnhet, de sohasem lehet eléggé hangsúlyozni. A jól olvasható kód és a könnyen módosítható kód fogalma ugyanis nem teljesen egyezik meg.

Dave a tisztaságot a tesztekhez köti. Tíz évvel ezelőtt sokan felhúzták volna a szemöldöküket e mondat hallatán, de a tesztvezérelt fejlesztés (TDD, Test Driven Development) megjelenése komoly változásokat hozott az iparágban, olyannyira, hogy ez vált a fejlesztések egyik alapkövévé. Dave-nek pedig igaza van. A kód tesztek nélkül soha nem lehet igazán tiszta. Legyen akármilyen elegáns, olvasható és hozzáférhető – tesztek nélkül semmiképpen sem tiszta.

Dave kétszer is használja a *minimális* szót. Látható, hogy szívügye a rövidre fogott kódolás, és ellensége a dagályosságnak. Nos, a programozás irodalmában ez az álláspont a kezdetektől jelen van – a kisebb jobb.

Dave azt is megjegyzi, hogy a kódunknak *olvasmányosnak* kell lennie – ami laza utalás Knuth *olvasmányos programozására*[4]. A lényeg mindebben az, hogy a kódunkat úgy készítsük el, hogy az emberi agy számára könnyen befogadható legyen.

[4] [Knuth92]

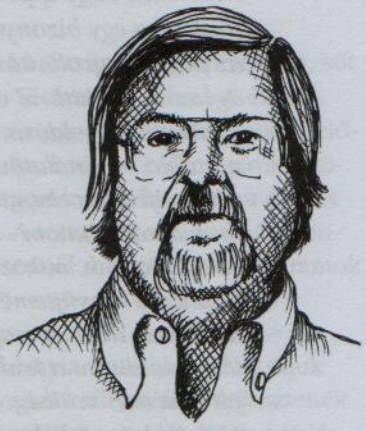
Michael Feathers, a *Working Effectively with Legacy Code* szerzője

Felsorolhatnám a tiszta kód minden, általam fontosnak tartott jellemzőjét, de létezik közöttük egy, amiből lényegében az összes többi következik. A tiszta kódon mindig érezni, hogy aki írta, nagy gonddal járt el. Ezt az mutatja a legjobban, hogy egyszerűen nem érezzük szükségét annak, hogy megváltoztassuk a jobb működés érdekében. A kód szerzője ugyanis számba vett minden lehetőséget a jobbításra – ha valamit kiötlünk, és megpróbáljuk továbbgondolni, hamarosan ugyanoda érkezünk vissza, ahonnan elindultunk, és egyre nagyobb csodálattal állunk a kód előtt, amelyet valaki ránk hagyott – valaki, aki a szakmájában a legnagyobb műgonddal járt el.

Egyetlen szó: *gondosság*. Ezt a szót bátran megjelölhetnénk könyvünk témájaként is. Alcímként talán ezt írhattuk volna: *Miként gondoskodjunk a kódunkról?*.

Michael valóban fején találta a szöget. A tiszta kód nem más, mint gondosan megírt kód. Olyan kód, amelynek a szerzői ügyeltek az egyszerűségre és a rendezettségre, és kellő figyelmet fordítottak a részletekre – vagyis valóban a gondjaikba vették a kódot.

Ron Jeffries, az *Extreme Programming Installed* és az *Extreme Programming Adventures in C#* szerzője



Ron a pályafutását a Strategic Air Command Fortran-programozójaként kezdte, és azóta szinte minden programozási nyelvet kipróbált majdhogynem minden elképzelhető gépen. Mindennek tudatában érdemes odafigyelnünk arra, amit mond.

Az utóbbi években jobbára Beck egyszerű kódra vonatkozó szabályait tekintem elsődlegesnek. Eszerint (fontossági sorrendben) az egyszerű kód:

- *Minden teszten átmegy.*
- *Nem tartalmaz ismétlődő kódot.*
- *Kifejezi a rendszerben jelen levő összes tervezési elképzelést.*
- *A lehető legkisebbre csökkenti a logikai egységek, így az osztályok, tagfüggvények, függvények és más hasonlók számát.*

A fentiek közül elsősorban a kódismétlődésre hívnám fel a figyelmet. Ha a kódunk ugyanazt a feladatot újra és újra végrehajtja, az biztos jele annak, hogy van egy gondolat a fejünkben, amit nem megfelelő módon fejeztünk ki a kódban. Jómagam ilyenkor megkísérlek utánajárni, mi is ez a gondolat – ha megtaláltam, a következő lépésben a kódban is világosabban jelenítem meg.

A gondolatok alkalmas kifejezése számomra ügyesen megválasztott neveket is jelent, ezért gyakran előfordul, hogy egy nevet számtalanszor megváltoztatok, mielőtt elfogadnám a végleges alakot. A névváltoztatás manapság, az Eclipse és hozzá hasonló kódszerkesztő eszközöknek hála nem különösebben költséges feladat, így semmi sem tart vissza a módosítástól. A kifejezőkészség azonban túlmutat a neveken. Azt is érdemes megnézni, hogy egy objektum vagy tagfüggvény egynél több feladatot hajt-e végre. Amennyiben objektumról van szó, jó eséllyel tanácsos több részre bontani. Ha tagfüggvény az áldozat, a Tagfüggvény kiemelése újratervezési átalakítást alkalmazom rajta, így végül hozzájutok egy olyan tagfüggvényhez, amelynek a feladata nyilvánvalóbb, mint az elődjéé, valamint további segédtagfüggvényekhez, amelyek a végrehajtás részleteiről árulkodnak.

Az ismétlődés és a kifejezőkészség vizsgálatával igen közel juthatunk a tiszta kód ideáljához, így ha csak e két szempont alapján módosítjuk a „piszkos" kódot, már akkor is komoly eredményt érhetünk el. Munkamódszeremnek azonban van egy harmadik pillére is, amit azonban már nem ilyen könnyű körülírni.

Az évek munkája során kikristályosodott bennem, hogy a programok általában igencsak hasonló elemekből épülnek fel. Az egyik ilyen feladattípus a „keresés egy gyűjteményben". Legyen szó dolgozók adatbázisáról, kivonattábláról kulcsokkal és értékekkel, vagy éppen elemek valamilyen tömbjéről, a feladat ugyanaz – keressünk meg egy bizonyos elemet ebben a gyűjteményben. Ha ilyen feladatra akadok, az adott megvalósítást megkísérlem egy elvontabb tagfüggvénybe vagy osztályba ágyazni. Így kitárul előttem néhány új, izgalmas lehetőség.

A feladat megoldását most már valamilyen egyszerű módszerrel, például egy kivonattábla használatával is megvalósíthatom, hiszen a keresés hivatkozásait jótékonyan elfedi az elvont ábrázolás. A megvalósítást így kedvem szerint bármikor megváltoztathatom – következésképpen semmi akadálya annak, hogy gyorsan továbbhaladjak, miközben továbbra is megmarad a változtatás lehetősége.

Ráadásul a gyűjteménykezelés elvontsága rávilágít a feladat lényegére, így nem sétálok bele abba a csapdába, hogy az adott gyűjtemény tulajdonságainak megfelelően alakítsam a kódot, miközben a sikeres kereséshez mindössze pár egyszerű eljárásra van szükség.

Mérsékelt ismétlődés, kifejező kód és egyszerű absztrakciók a lehető legkorábban. Ezek jelentik számomra a tiszta kód lényegét.

Ron képes volt pár rövid bekezdésben összefoglalni könyvünk tartalmát: nincs ismétlődés, összpontosítás egyetlen feladatra, kifejezőkészség és apró absztrakciók. Ez minden, amire szükségünk van!

Ward Cunningham, a Wiki és a Fit atyja, az extrém programozás feltalálóinak egyike, a programtervezési minták használatának ösztönzője, a Smalltalk és az objektumközpontú programozás gondolati megalapozója. Akik valóban törődnek a kóddal, kicsit mind a keresztapjuknak tekintik.

Ha tiszta kóddal dolgozunk, minden eljárásnál úgy érezzük, hogy pontosan azt kapjuk, amire számítottunk. Sőt, vannak olyan szép kódok is, ahol nem tudunk szabadulni a gondolattól, hogy a nyelvet kifejezetten ennek a feladatnak a megoldására ötlötték ki.

Wardtól pontosan ilyen világos mondatokra számítottunk. Legszívesebben csak egyetértően bólintanánk, és jóleső érzéssel a lelkünkben továbblépnénk. Annyira magától értetődő, annyira ésszerű mondatok, hogy kissé nehéz is észrevenni a bennük rejlő mélységet. Igen, mi is pontosan erre gondoltunk – de várjunk csak, próbáljunk egy kicsit mélyebbre hatolni!

„... pontosan azt kapjuk, amire számítottunk." Mikor láttunk utoljára olyan modult, ami éppen olyan volt, amilyenre számítottunk? Ha a modulokra gondolunk, nem inkább a rejtvények és a fejtörők világa jelenik meg a lelki szemeink előtt? Nem a félrevezetés a főszabály? Nem szoktunk még hozzá ahhoz, hogy kétségbeesetten próbáljuk kézben tartani a logikai szálakat, amelyek egyre bonyolultabb szövedéket képeznek, ahogy mélyebbre hatolunk a modul szerkezetében? Mikor éreztük utoljára egy kódrészlet átolvasása után azt a megnyugtató érzést, ami Ward mondatainak olvasását követően töltött el bennünket?

Ward azt állítja, hogy ha tiszta kódot olvasunk, nem igen lesz részünk meglepetésben, sőt valójában az olvasás sem kerül különösebb fáradságunkba. Elolvassuk a kódot, és azt kapjuk, amire számítottunk. Magától értetődő, egyszerű és meggyőző. Minden modul előkészíti a terepet a következő számára. Mindegyik megmutatja, hogyan kell megírni a következőt. Az ennyire tiszta programok olyan mélységben átgondoltak, hogy ezt a rendet szinte észre sem vesszük, annyira természetesnek érezzük. A tervező keze nyomán minden olyan nevetségesen egyszerűvé válik – ez a különlegesen jól megtervezett programok feltűnő sajátsága.

De mi a helyzet azzal, amit Ward a szépségről mond? Mindannyian dühösen elutasítjuk azt a tényt, hogy a programozási nyelveket nem a mi igényeink kielégítésére fejlesztették ki. Ward állítása azonban a mi vállunkra helyezi a felelősséget. Azt mondja, hogy a valóban szép kód olvasásakor úgy érezzük, *mintha a nyelvet kifejezetten az adott feladathoz készítették volna.* Vagyis a *mi* felelősségünk úgy használni a nyelvet, hogy egyszerűnek mutatkozzon. Nyelvi megszállottak, figyelem! Nem a nyelv sajátosságaitól lesz egyszerű a program kódja – csakis a programozó felel ezért!

Filozófiai iskolák

De mit gondol minderről Bob bácsi (vagyis jómagam)? Mi is az a tiszta kód? Nos, a választ – méghozzá kínos részletességgel – megkapjuk ebben a könyvben. Munkatársaimmal karöltve elmondjuk majd, mit is értünk tiszta változónév, tiszta függvény, tiszta osztály és más hasonlók alatt. Véleményünket abszolútumként mutatjuk be, és sosem mentegetőzünk majd túlzott magabiztosságunk miatt. Számunkra ugyanis pályafutásunk jelen állomásán ezek valóban abszolút értékek. Ez a mi *filozófiai iskolánk* a tiszta kód tekintetében.

A harcművészek nem neveznek meg egyértelműen „legjobb" műfajt vagy technikát. Gyakori azonban, hogy a mesterek saját iskolákat alapítanak, és tanítványokat gyűjtenek maguk köré. Így született meg Brazíliában a *Gracie dzsiu-dzsicu* a Gracie család révén, de ismert a *Hakkoryu dzsiu-dzsicu* is, amelyet Okuyama Ryuho ad át tanítványainak Tokióban. Vagy ott van a *Jeet Kune Do* stílus, amelyet Bruce Lee dolgozott ki és oktatott az Egyesült Államokban.

Az irányzatokhoz csatlakozó tanítványok elmélyednek mesterük tanításaiban. Eltökélten követik a mestert, és gyakran kizárják más mesterek tanításait. Később, amikor a tanítványok tudása gyarapszik, elkerülhetnek más mesterekhez is a látókörük bővítése és a gyakorlatszerzés reményében. Végül, ahogy a képességeik magasabb szintre fejlődnek, új technikákat fedezhetnek fel, és saját iskolákat alapíthatnak.

Az iskolák egyike sem rendelkezik a *bölcsek kövével*. Mindazonáltal az iskolákon belül *úgy teszünk*, mintha az adott tanítás jelentené az egyetlen igaz utat. Nos, ez valahol rendjén is van, hiszen a Hakkoryu dzsiu-dzsicu vagy éppen a Jeet Kune Do művelésének létezik „helyes" módja. Az egyes iskolákon belüli „helyes út" azonban még nem teszi érvénytelenné a többi iskola tanításait.

Tekintsük úgy ezt a könyvet, mint a *Tiszta Kód Objektumközpontú Iskolájának* szabálykönyvét. Az itt található technikák és tanítások megmutatják, miként gyakoroljuk *mi* a *saját* „harcművészetünket". Annyit állítunk, hogy aki követi ezeket a tanításokat, részesül az általunk is élvezett jótéteményekből, és képes lesz arra, hogy tiszta és professzionális kódot írjon. Ne essünk azonban abba a csapdába, hogy azt gondoljuk, abszolút értelemben ez az „egyetlen helyes út". Léteznek más iskolák és mesterek, akik a szakértelem éppoly komoly igényével lépnek fel, mint mi. Sőt mi több, mindenkit csak biztatni tudunk, hogy tőlük is tanuljon!

Be kell ismernünk, e könyv ajánlásaiban több vitatható pontra is bukkanhatunk, és korántsem biztos, hogy mindegyikkel egyetértünk. Nos, ez teljesen rendben van. Szerzőként nem tekinthetjük magunkat döntőbírónak. Másrészről, az itt megfogalmazott tanácsok

mindegyikét alaposan meghánytuk-vetettük; évtizedek tapasztalata, kísérletei és kudarcai nyomán rajzolódott ki az itt vázolt kép. Így hát akár egyetértünk a leírtakkal, akár nem, a megfontolásuk mindenképpen üdvös lehet.

Mindannyian szerzők vagyunk

A Javadoc-dokumentumok `@author` (szerző) mezője pontosan megmutatja, kik is vagyunk valójában. Igen, szerzők vagyunk. A szerzőknek pedig olvasóik vannak, így *felelősek* azért, hogy miként szólnak hozzájuk. A következő alkalommal, amikor kódolni kezdünk, idézzük emlékezetünkbe, hogy szerzők vagyunk, így a munkánkat az olvasóink ítéletére kell bíznunk.

Megkérdezhetnénk persze, hogy egyáltalán mennyit olvassuk a kódot? Hiszen úgy gondolhatnánk, hogy a nagyobb energiát a kód megírása emészti fel!

Nos, megesett már, hogy visszajátszottuk a szerkesztési műveleteinket? A nyolcvanas-kilencvenes évek szerkesztői, mint az Emacs, minden egyes billentyűleütést számon tartottak, így megtehettük, hogy egy órát dolgoztunk, majd az egészet felgyorsítva visszanéztük. Félelmetes élmény, én próbáltam!

A „film" legnagyobb része görgetésből és más modulok felkereséséből állt!

Bob belép a modulba.
Lefelé görgeti a képernyőt egy módosítandó függvényhez.
Megáll, elgondolkodva a lehetőségeken.
Hoppá, most felfelé görget a modul tetejére, hogy megnézze egy változó kezdőértékét.
Most pedig visszagörget, és írni kezd.
Hoppá, hirtelen törölte, amit beírt.
Ismét ír...
És megint töröl!
Most félig beír valamit, azután ezt is törli.
Lefelé görgeti a képernyőt, hogy megnézze a módosítandó függvény egy hívásának alakját.
Visszagörget, és beírja ugyanazt, amit az előbb törölt.
Megáll egy pillanatra.
És ismét törli a kódot!
Megnyit egy új ablakot, és megnéz egy alosztályt. Vajon felülbírálja a függvényt?
...

Nos, ennyi bőven elég, hogy lássuk, hogyan folynak a dolgok. Igazság szerint az olvasásra és az írásra fordított idő aránya jóval 10:1 felett van. Ez nem meglepő, hiszen állandóan olvassuk a régi kódot annak érdekében, hogy az újat megfelelő módon írhassuk meg.

Ez a magas arány indokolja, hogy törekedjünk a kód olvashatóvá tételére még akkor is, ha ez némiképp körülményesebbé teszi az írást. Persze kódot nem írhatunk olvasás nélkül, így azzal, hogy *olvashatóbbá tesszük*, egyúttal *az írását is megkönnyítjük*.

Nincs menekvés ebből a logikai körből. Képtelenek vagyunk jó kódot írni, ha nem tudjuk elolvasni a környezetében levő régit. A ma megírt kód jövőbeni jó vagy rossz olvashatósága a környező jelenlegi kód olvashatóságának függvénye. Ha tehát gyorsan szeretnénk haladni, és hamar el szeretnénk készülni a munkával, ha azt szeretnénk, hogy könnyedén megírhassuk a kódot, tegyük jól olvashatóvá.

A cserkészek szabálya

Nem elég jól megírnunk a kódot – *folyamatosan tisztán kell tartanunk*. Mindannyian tapasztalhattuk, hogy a kód idővel hajlamos „megromlani". Ezt a romlást aktív munkával kell ellensúlyoznunk.

Itt vehetjük hasznát az amerikai cserkészek intelmének:

A táborhelyet tisztábban hagyd magad után, mint ahogy találtad![5]

Előzmények a szabályokhoz...

Ez a kötet több okból is a 2002-es könyvem, az *Agile Software Development: Principles, Patterns, and Practices* (PPP) előzményének tekinthető, noha később íródott. A PPP-könyv az objektumközpontú tervezés szabályaival foglalkozik, bemutatva a hivatásos programozók számos gyakran használt módszerét. Amennyiben nem olvastuk a könyvet, úgy érezhetjük majd, mintha jelen mű folytatása lenne. Ha viszont olvastuk, gyakran lehet majd olyan érzésünk, hogy az ott leírtak visszaköszönnek – ezúttal a kód szintjén.

Könyvünkben elvétve hivatkozunk olyan tervezési szabályokra, mint a Single Responsibility Principle (SRP, az egyetlen felelősség elve), az Open Closed Principle (OCP, a zárt megnyitás elve) és a Dependency Inversion Principle (DIP, a függőség-megfordítás elve) – ezekről a PPP-könyvben bővebben olvashatunk.

[5] Ez az intelem Robert Stephenson Smyth Baden-Powell cserkészeknek szánt búcsúüzenetéből ered, ami így hangzik: „Igyekezzetek ezt a világot egy kicsit jobb állapotban magatok mögött hagyni, mint ahogy találtátok..."

Összefoglalás

A művészetről szóló könyvek nem kecsegtetnek azzal, hogy művészt faragnak belőlünk. Mindössze annyit tehetnek, hogy bemutatják, milyen eszközöket és eljárásokat alkalmaztak, és milyen gondolkodásmódot követtek más művészek. Könyvünk sem tesz ígéretet arra, hogy minden olvasójából jó programozót nevel – nem csepegtethet mindenkibe „kódérzéket". Éppen csak annyit tehet, hogy bemutatja más programozók gondolkodásmódját, valamint az általuk használt fogásokat és eszközöket.

A művészetről szóló könyvekhez hasonlóan itt is rengeteg részlettel találkozunk majd, vagyis sok-sok kóddal. Jó és rossz kóddal egyaránt. Látjuk majd azt is, miként válhat a rossz kódból jó. Látunk majd árulkodó szagokat, szabályokat és módszereket, egyik példát a másik után. Hogy ebből mi ragad meg és mi nem, az csak rajtunk múlik!

Van egy régi vicc egy zenekari hegedűsről, aki eltéved a koncertterem felé vezető úton, ezért megkérdez egy öregembert a sarkon, hogy miként juthatna el a Carnegie Hallba. Az öreg végigméri a hegedűművészt a hóna alatt a hegedűvel, és csak ennyit mond: „Gyakorlással, fiam. Sok-sok gyakorlással!".

Irodalomjegyzék

[Beck07]: Kent Beck: *Implementation Patterns.* Addison-Wesley, 2007.

[Knuth92]: Donald E. Knuth: *Literate Programming.* Center for the Study of Language and Information, Leland Stanford Junior University, 1992.