Kódmásolatok karbantarthatóságra gyakorolt hatásainak kiértékelése

Ph.D. értekezés tézisei

Bakota Tibor

Témavezető:
Dr. Gyimóthy Tibor

Informatika Doktori Iskola
Informatikai Tanszékcsoport
Szegedi Tudományegyetem

Szeged
2012

Bevezetés

A szoftverek mára életünk szerves részévé váltak. Nem csupán kényelmesebbé teszik mindennapjainkat, hanem sokszor még az életünket is rájuk bízzuk. Az iparág robbanásszerű növekedése komoly kihívás elé állítja a szoftverfejlesztéssel foglalkozó cégeket és szakembereket egyaránt. A piaci nyomás arra kényszeríti az informatikai vezetőket, hogy egyre gyorsabban és olcsóbban állítsák elő a különböző szoftver termékeket. A rövid távú célok elérése érdekében kötött kompromisszumok óhatatlanul is a minőség rovására mennek hosszú távon.

A forráskód másolása (klónozása) a termelékenység növelésének egyik legvitatottabb módja, mivel rövid távon csökkenti ugyan a fejlesztés idejét, hosszú távon azonban a karbantartási költségek drasztikus növekedéséhez vezethet. Az egyik legsúlyosabb érv a klónozással szemben, hogy amennyiben valamely kódrészlet módosításra szorul, úgy valamennyi másolt szakasz ellenőrzésre és kiigazításra szorul. Amennyiben a másolt szakaszok módosítását elmulasztják, a működés során hibák és logikai inkonzisztenciák léphetnek fel. Ebből kifolyólag a kódmásolatokat általánosságban véve a forráskód karbantarthatóság fő ellenségének szokás tekinteni. A valódi aggályok nem is a másolatok jelenlétéből adódnak, hanem a forráskód evolúciójával összefüggésben merülnek fel. Annak elősegítése érdekében, hogy a kódmásolatoknak a forráskód karbantarthatóságára gyakorolt hatásait elemezni tudjuk, kidolgozásra került egy eljárás, amely segítségével a kódmásolatok időben követhetővé válnak az evolúció során.

Megalapozott stratégiai döntések meghozatala szempontjából elengedhetetlen a forráskód karbantarthatóságának számszerűsítése. A karbantarthatóság mérése a mai napig komoly kihívásnak számít a szoftverfejlesztés ipari és akadémiai berkeiben egyaránt. A legfőbb nehézséget a formális definíciók hiánya, valamint a fogalmakban rejlő szubjektivitás okozza. Az ISO/IEC 9126 szabvány [11] ugyan segítséget nyújt a karbantarthatóság fogalmának meghatározásához, azonban nem ad támpontot annak standard módon történő számszerűsítéséhez. A kutatók – kihasználva a szabvány rugalmasságát – több, a gyakorlatban alkalmazható karbantarthatósági modellt alkottak [10, 16, 4, 1]. A disszertációban egy új eljárást mutatunk be a forráskód-karbantarthatóság mérésére, amely több szempontból is különbözik a világban jelenleg fellelhető módszerektől, és amely azok több hiányosságára is megoldással szolgál. A kidolgozott eljárás megfelelően kezeli a fogalmak szubjektív értelmezéséből adódó problémákat, ugyanakkor az előálló modell abszolút módon fejezi ki egy rendszer karbantarthatóságának mértékét.

A karbantarthatóság jelentősége, a szoftver módosításának költségeivel összefüggésben mutatkozik meg. Munkánk során, kidolgoztunk egy közönséges differenciálegyenleteken alapuló, formális matematikai modellt, amely a fejlesztési költségek és a forráskód-karbantarthatóság között fennálló összefüggések leírására szolgál. Megfigyelhető, hogy bizonyos ésszerű feltételezések mellett a közöttük fennálló viszony exponenciális is lehet.

Annak érdekében, hogy a kódmásolatok karbantarthatóságra gyakorolt hatását elemezzük, bevezetjük az ún. *„clone smell"*-ek fogalmát, amelyek a gyanús klón-evolúciós minták leírására szolgálnak. A „clone smell"-ek felhasználásával lehetőség nyílik a valóban veszélyesnek tekinthető másolatok azonosítására, és a klónok hatékony kezelésének megvalósítására. Az aggályos kódrészek listája nagyságrendekkel kevesebb elemet tartalmaz, mint amennyi egy rendszerben lévő másolatok száma, ezáltal azok kézi kiértékelése is elvégezhető.
A disszertáció az alábbi három fő tézispontot tartalmazza:

1. **Valószínűségi forráskód-karbantarthatóság modell kifejlesztése.**
2. **Forráskód-karbantarthatóságon alapuló költség modell felállítása.**
3. **Klónok kiértékelése a forráskód-evolúció szempontjából.**

A következő fejezetekben röviden bemutatjuk a disszertációban kifejtett eredményeket, valamint hangsúlyozzuk a szerző önálló hozzájárulását az egyes tézisek vonatkozásában.

1. Valószínűségi forráskód-karbantarthatóság modell

A forráskód-karbantarthatóság mérésének kérdése mindig is központi jelentőségű volt a szoftverfejlesztés világában. Az alacsony szintű jellemzők (forráskód-metrikák, kódolási szabálysértések, kódmásolatok) bizonyíthatóan befolyásolják egy szoftver forráskódjának karbantarthatóságát [9]. Kihívást jelent viszont a karbantarthatóság szubjektivitásának és az egyes modellek hordozhatóságának kezelése [3]. Az általunk kidolgozott forráskód-karbantarthatóság modell a jelenleg létező megközelítésekkel kapcsolatos problémák többségére is megoldást nyújt.

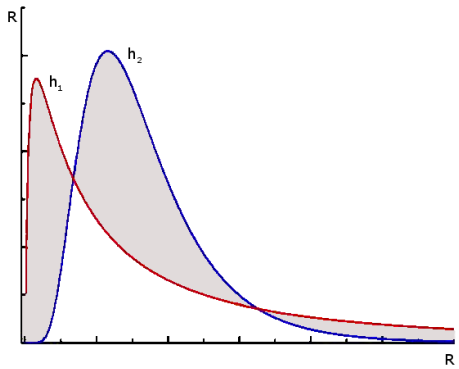
Valószínűségi karbantarthatósági modellek kialakításának elméleti háttere

Megközelítésünkben a különböző szintű minőség-jellemzők és karakterisztikák között fennálló kapcsolatot egy irányított körmentes gráffal ábrázoljuk, amelyet *attribútum függőségi gráfnak (AFG)* nevezünk. A legalacsonyabb szinten fekvő (bejövő éllel nem rendelkező) csúcsokat *érzékelőknek*, a többit pedig *származtatott* csúcsoknak nevezzük. A 2. ábrán egy példa AFG gráf látható.

Esetünkben az érzékelők egyes forráskód metrikáknak feleltethetők meg, amelyek értékei a forráskód elemzésével közvetlenül számíthatók. Egy szoftver esetében minden metrika tekinthető egy véletlen változónak, amely valós számokat vehet fel értékként adott valószínűségekkel. Két különböző szoftvert tekintve, legyenek $h_1(t)$ és $h_2(t)$ ugyanazon metrikához tartozó véletlen változók sűrűségfüggvényei. Ekkor az egyik rendszer másikhoz viszonyított *relatív jóság értéke* (az adott metrika szerint), az alábbi módon definiálható:

$$\mathcal{D}(h_1, h_2) = \int_{-\infty}^{\infty} (h_1(t) - h_2(t))\,\omega(t)\,dt,$$

ahol $\omega(t)$ egy súlyfüggvény, amely a „jóság" fogalmát fejezi ki, azaz, hogy a vízszintes tengely mely részein számítanak jobban a sűrűségfüggvények közötti különbségek. A 1. ábra szemlélteti a fenti képlet jelentését, vagyis, hogy a relatív jóság érték nem más, mint a két sűrűségfüggvény közötti, $\omega(t)$-vel súlyozott, előjeles terület.



1. ábra. Sűrűségfüggvények összehasonlítása

Rögzített h sűrűségfüggvény esetén, $\mathcal{D}(h, \_)$ szintén egy véletlen változó, amely független bármely más rendszertől, és amelyet a rendszer *abszolút jóságának* (vagy egyszerűen *jóságnak*) nevezünk (az adott rögzített metrika vonatkozásában). Ezen véletlen változó empirikus eloszlása közelíthető

megfelelő számú, különböző rendszerekhez tartozó sűrűségfüggvényeknek a második paraméter helyére történő behelyettesítésével, azaz egy forráskód-metrika eloszlásokat tartalmazó ún. referencia adatbázis felhasználásával. A véletlen változó fenti módon közelített sűrűségfüggvényét *jóság függvénynek*, a várható értékét pedig *jóság értéknek* nevezzük. A fenti módszert követve, minden érzékelő esetén kiszámítható a hozzá tartozó jóság függvény.

Az AFG éleit illetően, szakértőket kérünk fel, hogy az egyes élekhez súlyokat rendeljenek, tapasztalataik és érzéseik alapján, amelyek a függőségek mértékét hivatottak kifejezni. A súlyozás oly módon történik, hogy minden származtatott csúcs esetén, a bejövő éleken a súlyok összege 1 legyen. Következésképpen, minden származtatott csúcshoz egy többdimenziós véletlen változó ($\vec{Y_v} = \left(Y_v^1, Y_v^2, \ldots, Y_v^n\right)$) feleltethető meg, amely minden szavazó véleményét tartalmazza egyszerre. Hasonlóan a származtatott csúcsok esetén is definiálható egy jóság függvény az alábbiak szerint:

$$g_v(t) = \int_{\substack{t=\vec{q}\vec{r} \\ \vec{q}=(q_1,\ldots,q_n)\in\Delta^{n-1} \\ \vec{r}=(r_1,\ldots,r_n)\in C^n}} \vec{f}_{\vec{Y}_v}(\vec{q})\, g_1(r_1)\ldots g_n(r_n)\, d\vec{r}d\vec{q},$$

ahol $\vec{f}_{\vec{Y}_v}(\vec{q})$ a $\vec{Y}_v$ véletlen változó sűrűségfüggvénye, $g_1, g_2, \ldots g_n$ a bejövő élek kiindulási csúcsaihoz tartozó jóság függvények, Δ^{n-1} az $(n-1)$-szimplex $\Re^n$-ben, C^n pedig az n-dimenziós standard kocka $\Re^n$-ben.

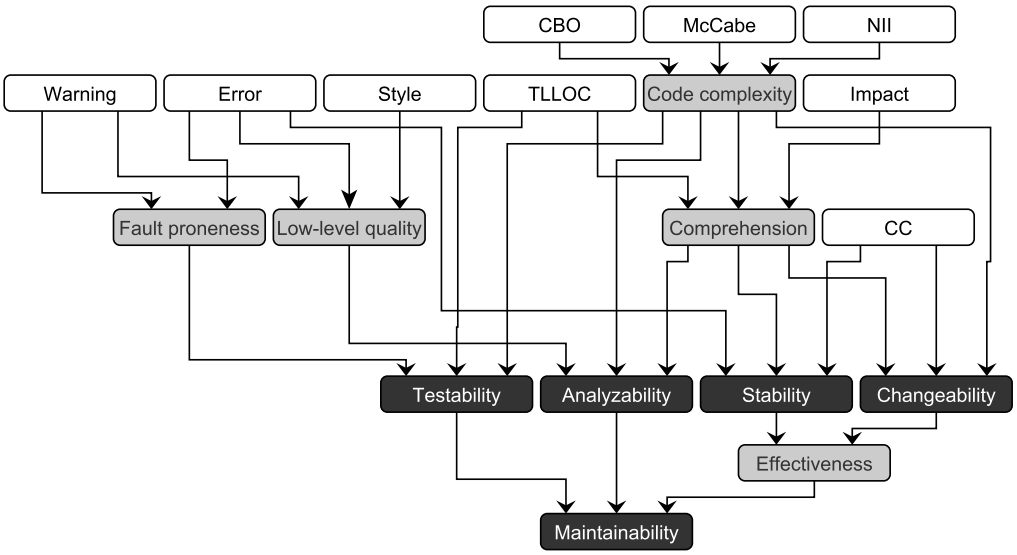
A fenti képlet első ránézésre ijesztőnek tűnhet, de valójában csak a klasszikus lineáris kombináción alapuló aggregációs eljárások általánosításáról van szó. Ugyanis, amikor valószínűségekről beszélünk az összes lehetséges (nem 0 valószínűségű) kombinációt figyelembe kell venni, és valószínűséget rendelni az egyes kimenetekhez. Ezek után az egyes származtatott csúcsokhoz is megfeleltethető egy-egy jóság függvény. A származtatott csúcsoknak megfelelő jóság függvények kifejezik a fogalmak szubjektivitását (mivel minden, az élek súlyozásában résztvevő szakértő véleménye figyelembe van véve), és abszolút mértéket jelentenek, mivel a referencia adatbázisban szereplő rendszerek biztosítják az abszolút jóság véletlen változó megfelelő közelítését.

Forráskód-karbantarthatóság modell Java nyelvre

A módszer validációjának céljából, Java programozási nyelv esetében, kialakításra került egy konkrét attribútum függőségi gráf, amelyben az érzékelőket a legelterjedtebb forráskód-jellemzők alkotják: forráskód metrikák [7], kódolási szabálysértések és kódmásolatok [5]. Az AFG végleges állapotának kialakítása akadémiai és ipari szakértők bevonásával, több iteráción keresztül valósult meg. További iterációk során kialakult a gráf éleinek súlyozása, valamint a származtatott csúcsokhoz kapcsolódó szavazat-eloszlások. A 2. ábra szemlélteti az így kapott attribútum függőségi gráfot. Ezzel párhuzamosan kialakításra került egy referencia adatbázis, amely száz nyílt és zárt forráskódú, Java programozási nyelven íródott szoftver forráskódjának jellemzőit tartalmazza. A kialakított modellt használtuk a forráskód-karbantarthatóság mérése céljából megalkotott módszerünk validálására.

Forráskód-karbantarthatóság modell validációja

A fentiekben kialakított karbantarthatósági modell validációját két, Java nyelven implementált szoftverrendszer esetében végeztük el. Az első egy magyarországi cég által több éven át fejlesztett ipari



2. ábra. Az alacsony (fehér) és magas (fekete) szintű ISO/IEC 9126 szabványban definiált karakterisztikák közötti kapcsolatokat leíró *AFG*

rendszer (*System-1*). A második a Szegedi Tudományegyetem Szoftverfejlesztés Tanszéke által fejlesztett, *RÉM* névre keresztelt perzisztencia keretrendszer. Célunk a modell által szolgáltatott mérőszámok és az adott rendszer fejlesztésében résztvevő szakértők véleményének ütköztetése volt. A fejlesztők tizes skálán osztályozták az általuk fejlesztett rendszer, ISO/IEC 9126 által definiált karakterisztikáit és karbantarthatóságát. A 1. táblázat összegzi a szavazatok normalizált átlagát mindkét rendszer esetén (a zárójelben szereplő értékek a modell által szolgáltatott jóság értékek).

Verzió	Változtathatóság	Stabilitás	Elemezhetőség	Tesztelhetőség	Karbantarthatóság
REM v0.1	0.625 (0.7494)	0.4 (0.7249)	0.675 (0.7323)	0.825 (0.7409)	0.625 (0.7520)
REM v1.0	0.6 (0.7542)	0.65 (0.7427)	0.75 (0.7517)	0.8 (0.7063)	0.75 (0.7539)
REM v1.1	0.6 (0.7533)	0.66 (0.7445)	0.7 (0.7419)	0.66 (0.6954)	0.633 (0.7402)
REM v1.2	0.65 (0.7677)	0.65 (0.7543)	0.8 (0.7480)	0.775 (0.7059)	0.7 (0.7482)
Korreláció	**0.71**	**0.9**	**0.81**	**0.74**	**0.53**
System-1 v1.3	0.48 (0.4458)	0.33 (0.4535)	0.35 (0.4382)	0.43 (0.4627)	0.55 (0.4526)
System-1 v1.4	0.6 (0.4556)	0.55 (0.4602)	0.52 (0.4482)	0.4 (0.4235)	0.533 (0.4484)
System-1 v1.5	0.64 (0.4792)	0.64 (0.4966)	0.56 (0.4578)	0.46 (0.4511)	0.716 (0.4542)
Korreláció	**0.87**	**0.81**	**0.94**	**0.61**	**0.77**

1. táblázat. A szakértői vélemények normalizált átlaga és a modell által számított karbantarthatóság értékek

Az eredmények különbséget mutatnak a modell illetve a szakértők által szolgáltatott mérőszámok között. Valójában a szakértői vélemények is nagy szórást mutatnak, ami a tapasztalatbeli és tudásbeli eltérésekből adódik. A táblázat kiemelt sorai a szakértők és a modell által számított értékek közötti korrelációt mutatják. A mindenhol pozitív, és viszonylag magas értékek arra utalnak, hogy a forráskód-

karbantarthatóság modell mutatóinak változása összecseng a fejlesztők által elvárt elmozdulással.

Saját hozzájárulás

A szerző a fejezetben bemutatott minősítő modellhez a matematikai háttér kidolgozásával, a statisztikai aggregáló algoritmusok kifejlesztésével járult hozzá. Ezenkívül részt vett a Java nyelvű modell kialakításában és a validáció módszerének kidolgozásában. Valamint elsősorban a szerzőnek tulajdoníthatók a klasszikus metrika alapú modellek hordozhatóságával kapcsolatos eredmények.

2. Forráskód-karbantarthatóságon alapuló költség modell

A világban jelenleg fellelhető költségbecslő megoldások [6] elsősorban manuális szakértői módszerekből [13], benchmark- [17] és modell alapú [17] megközelítésekből állnak. Az általunk kidolgozott módszer egyszerű és ésszerű feltételezésekből kiindulva, azok matematikai formalizálása és megoldása által eredményez egy költségbecslő modellt, amelyet nyílt és zárt forráskódú szoftvereken ellenőriztünk. A folytatásban bemutatásra kerülő modell a korábbiakban kifejtett forráskód-karbantarthatósági modell eredményeit is felhasználja.

Formális matematikai modell a fejlesztési költségek és karbantarthatóság kapcsolatának leírására

Költségbecslő modellünk két egyszerű feltételezésen alapul:

1. A forráskódon végrehajtott bármilyen módosítás, amely nem kifejezetten annak javítását célozza (pl. funkcionalitás hozzáadása) nem növeli annak karbantarthatóságát.

2. Kevésbé karbantartható szoftverek esetén a módosítások végrehajtása költségesebb.

Az első feltételezés az alábbiak szerint formalizálható:

$$\frac{d\mathcal{M}(t)}{dt} = -q\mathcal{S}(t)\lambda(t) \qquad (q \geq 0), \tag{1}$$

azaz, a karbantarthatóság ($\mathcal{M}(t)$) csökkenésének üteme egyenesen arányos a módosított sorok számával ($\mathcal{S}(t)\lambda(t)$) bármely *t* időpontban. A q állandót *eróziós tényezőnek* nevezzük, és azt fejezi ki, hogy mekkora „kárt" (karbantarthatóság romlást) okoz egyetlen sor módosítása.

A második feltevés az alábbiak szerint írható fel:

$$\frac{d\mathcal{C}(t)}{dt} = k\frac{\mathcal{S}(t)\lambda(t)}{\mathcal{M}(t)}. \tag{2}$$

A számláló a kódváltozás mértékét fejezi ki a t időpontban. A képlet szerint a t időpontban befektetett költség ($\frac{d\mathcal{C}(t)}{dt}$) kódváltozás formájában való hasznosulásának mértéke fordítottan arányos a forráskód karbantarthatóságának mértékével a t időpontban ($\mathcal{M}(t)$).

A fenti egyenletrendszert megoldva, az alábbi eredményhez jutunk:

$$\mathcal{M}(t_1) = \mathcal{M}(t_0)\, e^{-\frac{q}{k}(\mathcal{C}(t_1)-\mathcal{C}(t_0))}, \tag{3}$$

amely szerint a rendszer karbantarthatósága a változtatás céljából befektetett költség függvényében exponenciálisan csökken. Amíg k és $\mathcal{C}(t)$ értékek kiszámítása egyszerű, addig a q állandó számszerűsítése nem triviális. Mivel azonban a karbantarthatóság mérésére az előző fejezetben bemutatott eljárás segítségével lehetőség nyílik, ezért a (3) képlet alapján a q állandó is könnyen számszerűsíthető.

Amennyiben a költségbecslő modell paraméterei ismertek, a jövőbeli fejlesztési költségek az alábbi képlet szerint becsülhetők:

$$\mathcal{C}(t) = -\frac{k}{q}\ln\left|1 - \frac{q}{\mathcal{M}(0)}\int_0^t \mathcal{S}(s)\lambda(s)\,ds\right|. \tag{4}$$

Költségbecslő modell validációja valós rendszereken

A fentiekben vázolt költségbecslő modell validációját öt különböző, Java nyelven íródott szoftverrendszer több verziójának elemzésével hajtottuk végre, az alábbi lépések szerint:

1. Mindegyik rendszer összes elemzett verziója esetén – felhasználva a korábban bemutatott karbantarthatóság modellt – kiszámoltuk a megfelelő forráskód karbantarthatóságának mértékét [2]. Ezt az értéket a modellben szereplő $\mathcal{M}(t)$ függvény értékének közelítéséhez használtuk.

2. Minden forráskód-verzió esetén kiszámoltuk a módosult sorok számát az előző verzióhoz képest. Az így kapott értéket a modell $\mathcal{S}(t)\,\lambda(t)$ kifejezésének közelítésére használtuk.

3. A k és q állandók becslését a (2) és (3) képletek alapján számítottuk, valamely $T_0 > 0$ időpontban, az alábbiak szerint:

$$k = \mathcal{C}(T_0)\left(1/\int_0^{T_0} \frac{\mathcal{S}(t)\,\lambda(t)}{\mathcal{M}(t)}dt\right) \tag{5}$$

és

$$q = -\frac{k}{\mathcal{C}(T_0)}\ln\frac{\mathcal{M}(T_0)}{\mathcal{M}(0)}. \tag{6}$$

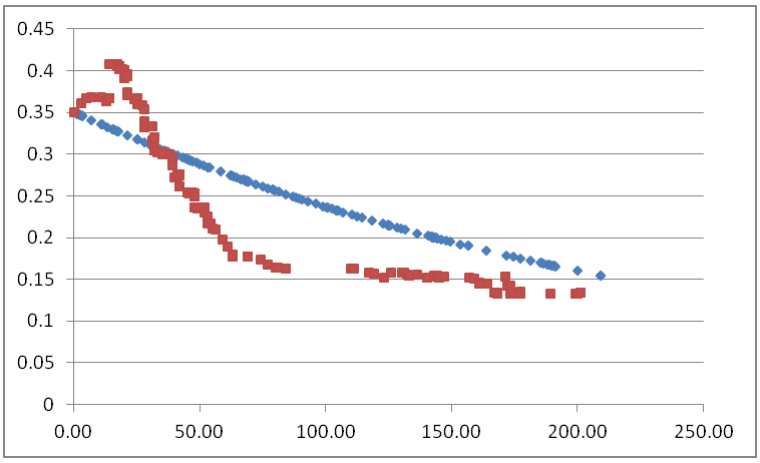
4. Ezek a becslések, mivel q és k állandók, érvényesnek tekinthetők minden $t > T_0$ esetén. Ebből adódóan a (4) képlet segítségével becslések adhatók a jövőbeli fejlesztési költségekre vonatkozóan.

Az empirikus adatok elemzése során az alábbi következtetéseket vontuk le:

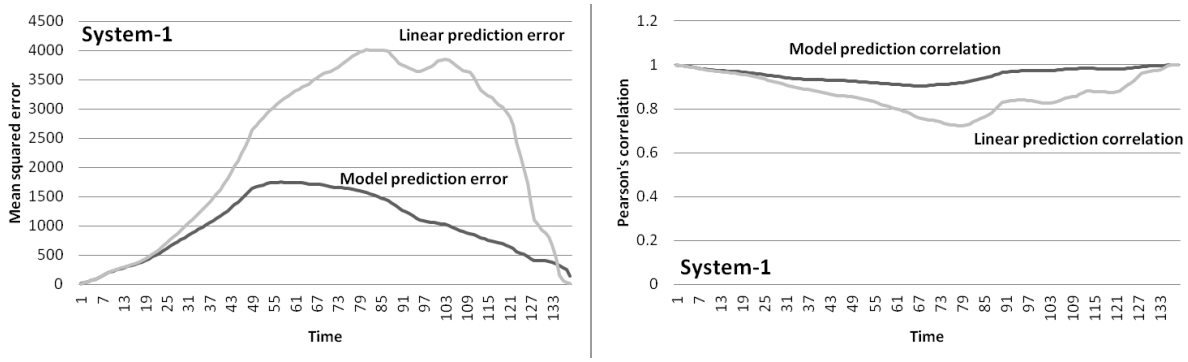
1. Általánosságban véve elmondható, hogy egy fejlesztés alatt álló szoftver karbantarthatósága az idő során csökken.

2. Egy rendszer karbantarthatósági mértéke és a fejlesztési költségek alakulása nagy korrelációt mutat a modell által becsült értékekkel, azaz egymással közel exponenciális kapcsolatban állnak. Az 3. ábra szemlélteti a karbantarthatóság mértékének alakulását a fejlesztési költségek függvényében a „*System-1*" rendszer esetén.

3. A bemutatott modell nagy pontossággal képes előre jelezni a fejlesztések jövőbeli költségeit a kódváltozás mértéke alapján. A (4) képlet felhasználásával a jövőbeli költségek, a karbantarthatóság várható alakulásának ismerete nélkül is becsülhetők. A modell pontosabb eredményeket szolgáltat, mint a klasszikus, lineáris regresszión alapuló megközelítések, amelyek nem veszik figyelembe a karbantarthatóság változását. Különösen a hosszútávú előrejelzések esetén szembetűnőek a különbségek, ami természetes, hiszen a karbantarthatóság változásának hatásai elsősorban hosszabb időszakot tekintve jelentősek. A 4. ábra szemlélteti a modell előrejelzési pontosságát a klasszikus lineáris megoldásokhoz képest. Az x tengely az előrejelzés tárgyát képező idő intervallum hosszát jelzi.



3. ábra. A karbantarthatóság ($\mathcal{M}(t)$) alakulása a költség ($\mathcal{C}(t)$) függvényében. Pirossal a mérési eredmények, kékkel a modell által előrejelzett értékek szerepelnek.



4. ábra. A modell előrejelző képessége, összehasonlítva a klasszikus lineáris regresszión alapuló megoldásokkal.

Saját hozzájárulás

A szerző saját hozzájárulása volt a formális matematikai háttér kidolgozása, valamint az empirikus validáció módszerének kidolgozása.

3. Klónok kiértékelése a forráskód-evolúció szempontjából

A kódmásolatok sokak szerint a szoftver karbantarthatóságának legfőbb ellenségei. Ennek megfelelően a korábban bemutatott forráskód-karbantarthatóság modellünkben is fontos szerepet játszanak. A kódmásolatok evolúciójának időbeli követése elengedhetetlen a kódmásolatok karbantarthatóságra gyakorolt hatásainak kiértékeléséhez.

Megoldás a kódmásolatok evolúciójának időbeli követésére

Megközelítésünkben a kódmásolatok, a szoftver egymást követő verzióiban, egymástól függetlenül kerülnek azonosításra, majd az így talált klónok egy heurisztikus eljárás segítségével kerülnek megfeleltetésre. Az *evolúciós megfeleltetés* a v_1 forráskód verzióban szereplő kódmásolatoknak egy parciális és injektív leképezése a v_2 verzióban szereplő másolatok halmazára:

$$e : G \subset CI^{v_1} \rightarrow CI^{v_2}.$$

Minden lehetséges klón pár esetén egy *hasonlósági távolság* érték kerül kiszámításra, amely az alábbi összetevőkből kerül származtatásra:

F_1 : A kódmásolat példányokat tartalmazó állományok nevei.

F_2 : A kódmásolat példányok azonosításának sorrendje a klón osztályon belül.

F_3 : Kódmásolat példányok egyedi azonosítói (amennyiben egyedi névvel rendelkező forráskód elemmel esnek egybe).

F_4 : A kódmásolat példányokat tartalmazó, egyedi névvel rendelkező forráskód elem azonosítója.

F_5 : Kódmásolat példányok relatív elhelyezkedése az őket tartalmazó forráskód elemen belül.

F_6 : Kódmásolat példányok szöveges hasonlósága.

A szöveges jellemzők (F_1, F_3, F_4 és F_6) esetében az ún. *Levenshtein távolságfüggvényt* [14] vettük alapul, az F_2 és F_5 attribútumok esetében pedig egyedi hasonlósági mértéket határoztunk meg. A fentieket felhasználva az alábbi származtatott hasonlósági távolságfüggvényt definiálhatjuk:

$$D\left(C_i, C_j\right) = \begin{cases} \sum_{k=1}^{6} \alpha_k D_k\left(C_i, C_j\right), & \text{if } \sum_{k=1}^{6} \alpha_k D_k\left(C_i, C_j\right) \leq \beta \\ \infty, & \text{különben} \end{cases},$$

ahol $\alpha_1, \ldots, \alpha_6$ és β paraméterek, amelyeket a továbbiakban bemutatásra kerülő optimalizációs algoritmus segítségével határozunk meg. Amennyiben a paraméterek ismertek, az evolúciós megfeleltetés kérdése visszavezethető egy hozzárendelési feladatra, amely a *Magyar módszer* segítségével hatékonyan megoldható.

A modell paramétereinek meghatározásához szimulált hűtést alkalmaztunk. Minden rögzített paraméter hozzárendelés esetén kiszámításra került az evolúciós megfeleltetés, és annak „jósága" egy előre definiált mérték szerint. A 2. táblázat összegzi az algoritmus által meghatározott optimális paraméter hozzárendelést, amely segítségével bármely két forráskód verzió esetén az evolúciós megfeleltetés kiszámítható.

Paraméter	Kezdeti	Optimalizált	Hozzájárulás mértéke
α_1	0.4082	0.3122	14.2 %
α_2	0.4082	0.6365	28.9 %
α_3	0.4082	0.2066	9.4 %
α_4	0.4082	0.4293	19.5 %
α_5	0.4082	0.1101	5.0 %
α_6	0.4082	0.5080	23.0 %
β	0.4082	0.0284	

2. táblázat. A modell kezdeti és optimalizált súlyai

A táblázatból kiderül, hogy a megfeleltetésben a legfontosabb szerepet az F_2 tulajdonság (kódmásolat példány azonosításának sorrendje a klón osztályon belül) játssza. Ez a jellemző a döntést 28.9%-ban befolyásolja, amíg a szöveges hasonlóság befolyásának mértéke 23%.

Klónok evolúciós mintáinak osztályozása

A folytatásban bevezetjük a *„clone smell"*-ek fogalmát, amely a másolatok különböző gyanús evolúciós mintáinak leírására szolgál, és amelyek alkalmas kiindulási pontot jelentenek további kézi vizsgálatok elvégzésére. Két egymást követő forráskód verziót tekintve az alábbi öt „clone smell" típust különböztetjük meg:

1. Eltűnő klón osztály (DCC) – olyan klón osztály, amely létezett a forráskód előző verziójában, de a jelenlegiben már nem.

2. Megjelenő klón osztály (ACC) – olyan a klón osztály, amely nem létezett a forráskód előző verziójában, de a jelenlegiben már igen.

3. Eltűnő klón példány (DCI) – olyan klón példány, amely létezett az előző verzióban, de már nem létezik, azonban az őt korábban tartalmazó osztály még igen.

4. Megjelenő klón példány (ACI) – olyan jelenleg is létező klón példány, amely nem létezett az előző verzióban, de az őt jelenleg tartalmazó osztály igen.

5. Mozgó klón példány (MCI) – olyan klón példány, amely egy másik osztály példánya lett.

A „clone smell"-eket a *Mozilla Firefox* [15] és a *jEdit* [12] rendszereken értékeltük ki, 295 illetve 84 egymást követő forráskód verziót tekintve. Az azonosított találatokat kézzel értékeltük ki, hogy megbizonyosodjunk a módszer hatékonyságáról és pontosságáról. A 3. és a 4. táblázatok összegzik a Mozilla illetve a jEdit rendszerben azonosított „clone smell"-eket, és azok kiváltó okait. Az eredmények alapján világossá vált, hogy a „clone smell"-ek hasznosak a karbantarthatóság javítása szempontjából az alábbiak miatt:

- A módszer egy viszonylag rövid, manuálisan ellenőrizhető listát eredményez azokról a kritikus forráskód részekről, amelyek inkonzisztens módosulásokkal kapcsolatos veszélyeket rejthetnek.

- A „clone smell"-ek több mint fele inkonzisztens módosulásból adódik, ezért érdemesek lehetnek további kézi elemzésre.

	Cause	DCC	ACC	DCI	ACI	MCI	Σ
Konzisztens módosulások	**C1**: Minden példány törölve	26					**26**
	C2: A példányok túl röviddé váltak	19					**19**
	C3: Az állomány törölve	5					**5**
	C4: Szándékos refactoring	3					**3**
	C5: Minden példány új		51				**51**
	C6: A példányok elég hosszúak lettek		3				**3**
	Σ	**53**	**54**				**107**
Inkonzisztens módosulások	**C7**: Az osztály néhány példánya törölve	11		6			**17**
	C8: Inkonzisztensen módosított példányok	38	21	14	7	13	**93**
	C9: További új példányok jöttek létre	2	8		5		**15**
	Σ	**51**	**29**	**20**	**12**	**13**	**125**
Σ		**104**	**83**	**20**	**12**	**13**	**232**

3. táblázat. A „clone smell"-ek kiváltó okai a Mozilla rendszer esetén

	Cause	DCC	ACC	DCI	ACI	MCI	Σ
Konzisztens módosulások	**C1**: Minden példány törölve						**0**
	C2: A példányok túl röviddé váltak						**0**
	C3: Az állomány törölve	1					**1**
	C4: Szándékos refactoring	3					**3**
	C5: Minden példány új		9				**9**
	C6: A példányok elég hosszúak lettek		1				**1**
	Σ	**4**	**10**				**14**
Inkonzisztens módosulások	**C7**: Az osztály néhány példánya törölve	1					**1**
	C8: Inkonzisztensen módosított példányok	12	8	1			**21**
	C9: További új példányok jöttek létre		4				**4**
	Σ	**13**	**12**	**1**			**26**
Σ		**17**	**22**	**1**			**40**

4. táblázat. A „clone smell"-ek kiváltó okai jEdit rendszer esetén

- Inkonzisztens módosulások gyakrabban előfordulnak, mint konzisztens változtatások.
- A módszer segítségével inkonzisztens módosulásokból adódó kódolási problémákra is fény derülhet.

A kódmásolatok és a csatoltság viszonya

A *csatoltság* a szoftver- vagy forráskód-komponensek közötti függőségek és kapcsolatok mértékének kifejezésére szolgál. A forráskód elemek közötti magas csatoltság a karbantarthatóság szempontjából általánosságban hátrányosnak tekintendő [8]. Meglepő módon a kutatásaink arra utalnak, hogy fordított a viszony a klónmásolatok és az osztályok közötti csatoltság mértéke között.

Esettanulmányunkban öt rendszert vizsgáltunk. Mindegyik esetben kiszámításra került a rendszer klónozottságának mértéke (CC), és két csatoltság mérték: az osztályok csatoltságának mértéke (CBO), valamint a kimenő hívások száma (NOI). A CC metrika rendszer szinten adott, azonban a CBO és a NOI mértékek esetében aggregáció végrehajtására volt szükség a rendszerszintű variánsok előállításához. A származtatott metrikákat *CBO-index*-nek illetve *NOI-index*-nek nevezzük.

A 3. táblázat alapján a csatoltság és a másolat metrikák között fordított viszony figyelhető meg, ami azt jelenti, hogy a rendszer karbantarthatóságát az egyik szempontból javítva, az a másik szempontból romolhat. A CBO és CC közötti korreláció -0.76, amíg a NOI és CC között -0.97. A fentiekből adódik, hogy szemben azokkal az elképzelésekkel, amelyek szerint csupán a csatoltság metrikák alapján becsülhető egy rendszer karbantarthatósága, a modelleknek a csatoltságot és a másolatokat egyaránt figyelembe kell venniük. Ezen a ponton visszautalunk a 2. ábrán látható modellre, amely magá-

Rendszer	A	B	C	D	E
CBO-index	-8.85	-7.60	-6.15	-3.74	1.17
NOI-index		-7.97	-4.67	-2.56	1.39
CC	32.7	16.9	11.44	9.94	7.47

5. táblázat. Az öt rendszer csatoltság és kódmásolat metrikái

ban foglalja az osztályok csatoltságának mértékét (CBO), a bejövő hívások számát (NII), valamint a kódmásolatok arányát (CC) is.

Saját hozzájárulás

A szerző hozzájárulása a tézisponthoz az alábbiak szerint foglalható össze:

- A kódmásolatok időbeli követését megvalósító módszer kidolgozása.
- Az evolúciós megfeleltetés számításához szükséges optimalizációs és szimulációs algoritmusok kifejlesztése.
- A „clone smell"-ek fogalmának bevezetése és definiálása.
- „Clone smell"-ek kinyerését végző algoritmusok és szoftver eszközök kifejlesztése.
- „Clone smell"-ek kinyerése két nagy rendszer egymást követő verzióiból.
- A kinyert „clone smell"-ek kézi kiértékelése.
- Kódmásolatokat kinyerő eszköz kifejlesztése a csatoltsággal kapcsolatos kutatások elősegítéséhez.

Konklúziók

Bemutattunk egy módszert a forráskód karbantarthatóságának mérésére, amely több szempontból is különbözik a korábbi megközelítésektől. A modellünk magában foglalja a szakértői tudást, kezeli a fogalmak szubjektivitásából adódó bizonytalanságot és ún. „jóság" függvényeket alkalmaz. Arra a következtetésre jutottunk, hogy a modell által számított karbantarthatóság érték változása kifejezi a fejlesztési tevékenységből adódó várakozást, azaz a fejlesztés során csökken, a karbantartás során pedig növekszik ez az érték. A modell által számított értékek ugyan nem esnek egybe a fejlesztők által becsülttel, azonban a kettő közötti korreláció mégis viszonylag magasnak mondható.

A folytatásban bemutattunk egy közönséges differenciálegyenlet-rendszeren alapuló, a forráskód-karbantarthatóság és a fejlesztési költségek viszonyának leírását elősegítő modellt. Az empirikus adatok elemzése az alábbiakra mutatott rá:

- Egy fejlesztés alatt álló szoftver karbantarthatósága idővel csökken.
- A forráskód-karbantarthatóság és a fejlesztési költségek közel exponenciális viszonyban vannak egymással.
- A modell viszonylag nagy pontossággal képes előrejelezni a jövőbeli fejlesztési költségeket a kódváltozás becsült mértéke alapján.

Bemutattunk továbbá egy módszert a kódmásolatok időbeli követésére, és bevezettük az ún. „clone smell"-ek fogalmát, amelyek segítségével azonosítani lehet a karbantarthatósági szempontból valóban veszélyes másolatokat. A kiértékelés során, a „clone smell"-ek valóban hatékony kódmásolat-menedzsment eszköznek bizonyultak.

A 6. táblázat összegzi, hogy mely publikációk, mely tézispontokhoz kapcsolódnak.

$\mathcal{N}o.$	[9]	[3]	[2]	[21]	[22]	[23]	[24]	[25]
1.	•	•	•				•	•
2.				•				
3.					•	•		

6. táblázat. A tézispontok és a publikációk között fennálló kapcsolatok

Köszönetnyilvánítás

Elsősorban szeretném megköszönni mentoromnak, Dr. Gyimóthy Tibornak, hogy hasznos ötletekkel, megjegyzésekkel és útmutatásokkal segítette kutatómunkámat. Köszönöm társszerzőmnek és szakmai irányítómnak, Dr. Ferenc Rudolfnak hogy irányt mutatott, motivált és a helyes úton tartott, amikor az szükségesnek bizonyult. Köszönet illeti továbbá munkatársaimat és társszerzőimet, Dr. Beszédes Árpádot, Dr. Siket Istvánt, Dr. Fülöp Lajost, Dr. Jász Juditot, Siket Pétert, Hegedűs Pétert, Dr. Schrettner Lajost, Dr. Gergely Tamást, Claudio Riva-t, Jianli Xu-t, Maarit Harsu-t, Kai Koskimies-t, Tarja Systa-t, Körtvélyesi Pétert, Illés Lászlót, Ladányi Gergelyt, Gyalai Milán Imrét és Füleki Dánielt. Szintén szeretném megköszönni publikációim névtelen bírálóinak, hogy hasznos javaslataikkal, kritikáikkal és megjegyzéseikkel segítették munkám. Külön köszönet David P. Curleynek, hogy nyelvi szempontból ellenőrizte és javította angol nyelven íródott disszertációmat.

Hálámat szeretném kifejezni szüleimnek, hogy megfelelő hátteret biztosítottak tanulmányaimhoz és bátorítottak az úton, amikor szükségem volt rá. Végül, de nem utolsó sorban köszönöm szeretett feleségemnek, Mónikának, hogy megértő és támogató volt oly sok éven át kutatásaim során.

Bakota Tibor, 2012

Hivatkozások

[1] Motoei Azuma. Software products evaluation system: quality models, metrics and processes - international standards and japanese practice. *Information and Software Technology*, 38(3):145 – 154, 1996.

[2] T. Bakota, P. Hegedus, P. Kortvelyesi, R. Ferenc, and T. Gyimothy. A probabilistic software quality model. In *Software Maintenance (ICSM), 2011 27th IEEE International Conference on*, pages 243 –252, Sept. 2011.

[3] Tibor Bakota, Rudolf Ferenc, Tibor Gyimothy, Claudio Riva, and Jianli Xu. Towards portable metrics-based models for software maintenance problems. *Software Maintenance, IEEE International Conference on*, 0:483–486, 2006.

[4] J. Bansiya and C.G. Davis. A Hierarchical Model for Object-Oriented Design Quality Assessment. *IEEE Transactions on Software Engineering*, 28:4–17, 2002.

[5] Ira D. Baxter, Andrew Yahin, Leonardo Moura, Marcelo Sant'Anna, and Lorraine Bier. Clone Detection Using Abstract Syntax Trees. In *Proceedings of the International Conference on Software Maintenance*, ICSM '98, pages 368–377, Washington, DC, USA, 1998. IEEE Computer Society.

[6] Barry Boehm, Chris Abts, and Sunita Chulani. Software development cost estimation approaches - a survey. *Annals of Software Engineering*, 10:177–205, 2000. 10.1023/A:1018991717352.

[7] S. R. Chidamber and C. F. Kemerer. A Metrics Suite for Object Oriented Design. *IEEE Trans. Softw. Eng.*, pages 476–493, June 1994.

[8] Tibor Gyimóthy, Rudolf Ferenc, and István Siket. Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction. *IEEE Transactions on Software Engineering*, pages 897–910, 2005.

[9] Péter Hegedus, Tibor Bakota, László Illés, Gergely Ladányi, Rudolf Ferenc, and Tibor Gyimóthy. Source code metrics and maintainability: A case study. In Tai-hoon Kim, Hojjat Adeli, Haeng-kon Kim, Heau-jo Kang, KyungJung Kim, Akingbehin Kiumi, and Byeong-Ho Kang, editors, *Software Engineering, Business Continuity, and Education*, Volume 257 of *Communications in Computer and Information Science*, pages 272–284. Springer Berlin Heidelberg, 2011.

[10] Ilja Heitlager, Tobias Kuipers, and Joost Visser. A practical model for measuring maintainability. In *Proceedings of the 6th International Conference on Quality of Information and Communications Technology*, QUATIC '07, pages 30–39, Washington, DC, USA, 2007. IEEE Computer Society.

[11] ISO/IEC. *ISO/IEC 9126. Software engineering – Product quality*. ISO/IEC, 2001.

[12] jEdit Homepage. `http://www.jedit.org`.

[13] M. Jorgensen, B. Boehm, and S. Rifkin. Software development effort estimation: Formal models or expert judgment? *Software, IEEE*, 26(2):14 –19, March-April 2009.

[14] Levenshtein distance.
`http://en.wikipedia.org/wiki/Levenshtein_distance`.

[15] The Mozilla Firefox Homepage.
`http://www.firefox.com`.

[16] S. Muthanna, K. Ponnambalam, K. Kontogiannis, and B. Stacey. A maintainability model for industrial software systems using design level metrics. In *Proceedings of the Seventh Working Conference on Reverse Engineering (WCRE'00)*, WCRE '00, pages 248–, Washington, DC, USA, 2000. IEEE Computer Society.

[17] K. Srinivasan and D. Fisher. Machine learning approaches to estimating software development effort. *Software Engineering, IEEE Transactions on*, 21(2):126 –137, Feb 1995.

Felhasznált publikációk

[18] Péter Hegedus, Tibor Bakota, László Illés, Gergely Ladányi, Rudolf Ferenc, and Tibor Gyimóthy. Source code metrics and maintainability: A case study. In Tai-hoon Kim, Hojjat Adeli, Haeng-kon Kim, Heau-jo Kang, KyungJung Kim, Akingbehin Kiumi, and Byeong-Ho Kang, editors, *Software Engineering, Business Continuity, and Education*, Volume 257 of *Communications in Computer and Information Science*, pages 272–284. Springer Berlin Heidelberg, 2011.

[19] Tibor Bakota, Rudolf Ferenc, Tibor Gyimothy, Claudio Riva, and Jianli Xu. Towards portable metrics-based models for software maintenance problems. *Software Maintenance, IEEE International Conference on*, 0:483–486, 2006.

[20] T. Bakota, P. Hegedus, P. Kortvelyesi, R. Ferenc, and T. Gyimothy. A probabilistic software quality model. In *Software Maintenance (ICSM), 2011 27th IEEE International Conference on*, pages 243 –252, Sept. 2011.

[21] T. Bakota, P. Hegedus, G. Ladányi, P. Kortvelyesi, R. Ferenc, and T. Gyimothy. A cost model based on software maintainability. In *28th IEEE International Conference on Software Maintenance (ICSM), 2012*, page to appear, Sept. 2012.

[22] Tibor Bakota. Tracking the evolution of code clones. In *Proceedings of the 37th international conference on Current trends in theory and practice of computer science*, SOFSEM'11, pages 86–98, Berlin, Heidelberg, 2011. Springer-Verlag.

[23] Tibor Bakota, Rudolf Ferenc, and Tibor Gyimothy. Clone smells in software evolution. *Proceedings of the 23rd International Conference on Software Maintenance (ICSM 2007)*, pages 24–33, 2-5 Oct. 2007.

[24] Marit Harsu, Tibor Bakota, Siket István, Kai Koskimies, and Systä Tarja. Code clones: Good, bad, or ugly? In *Proceedings of 11th Symposium on Programming Languages and Software Tools and 7th Nordic Workshop on Model Driven Software Engineering*, 2009.

[25] Marit Harsu, Tibor Bakota, Siket István, Kai Koskimies, and Systä Tarja. Code clones: Good, bad, or ugly? In *Nordic Journal of Computing special issue dedicated to SPLST'09 and NW-MODE'09*, 2010.