

# Forráskód karbantarthatóság és annak kapcsolata verziókövető történeti metrikákkal

**Faragó Csaba**

Szoftverfejlesztés Tanszék  
Szegedi Tudományegyetem

Szeged, 2016

Témavezető:  
Dr. Ferenc Rudolf

PH.D. ÉRTEKEZÉS TÉZISEI



Szegedi Tudományegyetem  
Informatika Doktori Iskola



# Bevezetés

Ha a programokat kizárólag számítógépek számára íránk, és biztosak lennénk abban, hogy a forráskódot ember soha többé nem látná, akkor a kódminőség nem lenne fontos. Ebben az esetben ez az értekezés is egészen más témáról szólna. Viszont ez nem igaz: a fejlesztők az idejük nagy részét a forráskód *olvasásával*, és nem annak *írásával* töltik. Emiatt komoly figyelmet kell szentelniük a forráskód karbantarthatóságára annak érdekében, hogy segítsék a későbbi fejlesztőket, akik adott esetben hosszabb idő elteltével akár saját maguk is lehetnek.

A forráskód minőség nagyon fontos, mert egy túl összetett, nehezen karbantartható kód egyrészt több hibát eredményez, másrészt a kód továbbfejlesztését is drágábbá teszi. A fejlesztők többnyire a feladatra koncentrálnak: azt meg kell oldani, működésre kell bírni, és a kódminőség az időnyomás alatt gyakran másodlagos. Az értekezést alátámasztó tanulmányok mögötti motiváció a következő volt: segíteni a fejlesztőket könnyebben karbantartható kód készítésére.

Az értekezés két fő részből áll: a *programszeletelésből* és a *verziókövető történeti metrikáknak karbantarthatóságra gyakorolt hatásáról*. A két téma közötti szoros kapcsolatot a **szoftver karbantarthatóság** képezi.

A **programszeletelés** egy lehetséges felhasználása a forráskód karbantartásának az elősegítése azáltal, hogy a fejlesztő számára kiemeli a kód lényeges részeit. Ezzel a technikával a fejlesztő mellőzheti azokat a kódrészleteket, amelyek lényegtelenek egy adott probléma szempontjából, és azokra az utasításokra fókuszálhat, amelyek valóban befolyásolják a hibás részt. Ebben az értekezésben egy konkrét dinamikus programszeletelési algoritmuson belül a nem strukturált utasítások kezelésére fókuszálunk.

A **verziókövető történet elemzése** témában megpróbálunk magyarázatot találni arra a kérdésre, hogy miért csökken a kód karbantarthatósága. Lehman hetedik törvénye a csökkenő minőségről szól, és kimondja, hogy a hétköznapi szoftverrendszerek hanyatlanak, ha nem tartják szigorúan karban, és nem követik az üzemeltetési környezet változásait. Az értekezés második részében e kód romlás és néhány verziókövető metrika kapcsolatával foglalkozunk. Habár mind a karbantarthatóság, mind a szoftver verziókövető rendszerekből történő adatbányászás témaköre részletesen elemzett kutatási területek, nem tudunk olyan tanulmányról, ami egészen konkrétan a verziókövető metrikák és a forráskód karbantarthatóságának kapcsolatát elemzi. Így ebben az értekezésben egy úttörő munkát mutatunk be ezen a fiatal kutatási területen.

Az értekezés három tézispontból áll. Ebben az összefoglalóban egy-egy fejezet egy-egy tézispontról szól. Bizonyos szakkifejezések esetében zárójelben megadtuk a szó angol eredetijét annak érdekében, hogy egyértelművé tegyük, pontosan mire gondoltunk.

# Első tézis pont: A nem strukturált C utasítások kezelése egy dinamikus szeletelési algoritmusban

A program szelet egy program utasításainak és predikátumainak egy részhalmaza, amelyek hatással vannak adott változókra adott ponton. Függetlenül attól, hogy a szelet meghatározása során csak statikus vagy dinamikus információt is figyelembe veszünk, beszélhetünk statikus vagy dinamikus programszeletelési eljárásról. Ebben az értekezésben dinamikus programszeleteléssel (*dynamic slicing*) foglalkozunk.

Gyimóthy Tibor szerzőtársaival tanulmányukban [15] egy előre haladó számolási technikát alkalmazó dinamikus programszeletelési algoritmust tettek közzé. A módszer lényege dióhéjban az alábbi. Mindegyik programsorban statikusan meghatározzuk, hogy mely változó kap értéket, és az mely változók értékétől függ. Az elágazó és ciklus utasításokat virtuális predikátum változóként kezeljük. A programot instrumentáljuk és úgy futtatjuk, megkapva ezzel azt, hogy mely programsorok milyen sorrendben hajtottak végre. Előre számolva minden egyes végrehajtási lépésben kiszámoljuk azt, hogy az adott sor aktuálisan mely más soroktól függ. Ez azon változók utolsó módosítási helyeinek és aktuális függőségeinek uniója, melyektől a kérdéses sorban kiszámolt változó függ.

A módszer memória igénye a gyakorlatban lineárisnak mondható az eredeti program memória igényével, ami jelentős javulás a korábbi módszerek hatalmas memóriaigényéhez képest. Ugyanakkor az algoritmus eredeti formájában alkalmatlan volt valós programok szeletelésére, mivel az csak értékadó, elágazó és ciklus utasításokat kezelte.

Beszédes Árpád szerzőtársaival cikkükben [3] a módszert valós C programokra illesztették. Ebben számos problémát kellett megoldani, például a függvényhívásokat vagy a mutatók kezelését.

Az egyik megoldandó probléma a C programozási nyelvben jelen levő nem strukturált utasítások kezelése, melyek a következők: `goto`, `break`, `continue` és `switch-case-default`. Megoldásunkban [8, 9] úgynevezett címke változókat vezettünk be, melyek értéket a kérdéses utasítás végrehajtásakor kapnak, ettől a változótól pedig a címke helyét követő utasítások függenek. A `goto` esetén a függő utasítások halmaza az összes, címkét követő utasítás, adott függvényen belül. A `break` esetén a függő utasítások a vonatkozó blokk (pl. a `while` ciklus belseje) utáni összes utasítás. A `continue` esetén a vonatkozó egység első utasításától kezdve az függvény végéig bele kell helyezni a függőséget. Ez egyébként azt is jelenti, hogy a `continue` mindig függ saját magától. A `switch-case-default` kezelésénél a `switch` ugyanolyan predikátum változóként kezelendő, mint pl. a `while`. Ha legalább egy belső utasítást tartalmaz az eredmény, akkor az összes `case` címke az eredmény része lesz, a `default` címkével együtt.

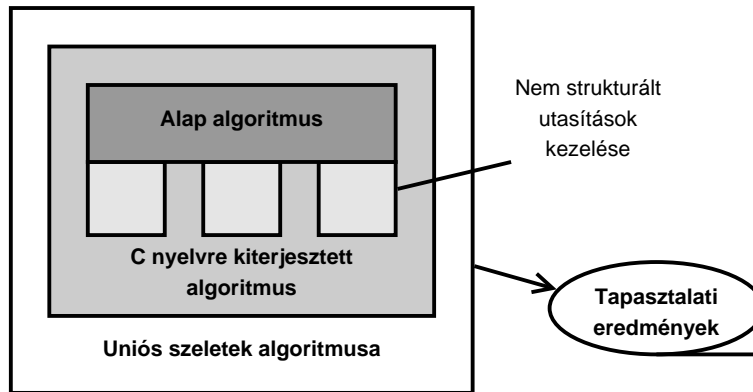
A módszert később kiterjesztettük [2], ahol a releváns programszeletet az összes lehetséges lefutás uniójaként definiáltuk. Jelentős programsor lefedettség mellett is az eredmény töredéke lett annak, amit egy statikus programszeletelő program kiszámolt.

## A szerző hozzájárulása az eredményekhez

Az értekezés szerzője az alábbiakkal járult hozzá az ebben a fejezetben bemutatott új eredményekhez:

- A nem strukturált utasítások kezelése a bemutatott dinamikus szeletelő algoritmusban.
- A megvalósításban való részvétel.
- A tesztek végrehajtása.

Az 1. ábrán bemutatjuk, hogy ezek a részek hogyan kapcsolódnak össze, és a rendszerben hol található a szerző hozzájárulása.



1. ábra. Az előre számoló dinamikus szeletelő algoritmusok áttekintése

## Második tézis pont: A verziókövető műveletek és a karbantarthatóság kapcsolata

Tanulmányok és tapasztalatok szerint is a kód minőségének – különösen a karbantarthatóságának – közvetlen hatása van a fejlesztés költségeire. Ugyanakkor a szoftver forráskódjának minősége romlik, ha nem fektetünk bele annak javításába. A motivációnk az volt, hogy megtaláljuk, hol és miért következik be a kód eróziója: léteznek-e tipikus fejlesztői műveletek, melyek hasonló változást okoznak a karbantarthatóságon? Ha ezt tudnánk, az segíthetne a kód romlásának megelőzésében.

Ebben a fejezetben bemutatjuk, hogy milyen kapcsolatot találtunk a verziókövető rendszer (*version control system*) műveletei és a vonatkozó kód változás okozta karbantarthatóság változás között. Először összegezzük, hogyan mértük meg a forráskód karbantarthatóságát. Ezt követően bemutatjuk a tulajdonképpeni új eredményeket: (2.A) először azt, hogy hogyan fedeztük fel a kapcsolat létezését, majd (2.B) hogy az egyes verziókövető műveleteknek milyen hatását találtuk, végül pedig (2.C) azt, hogy az eredményeket milyen új diagram típusokkal illusztráltuk.

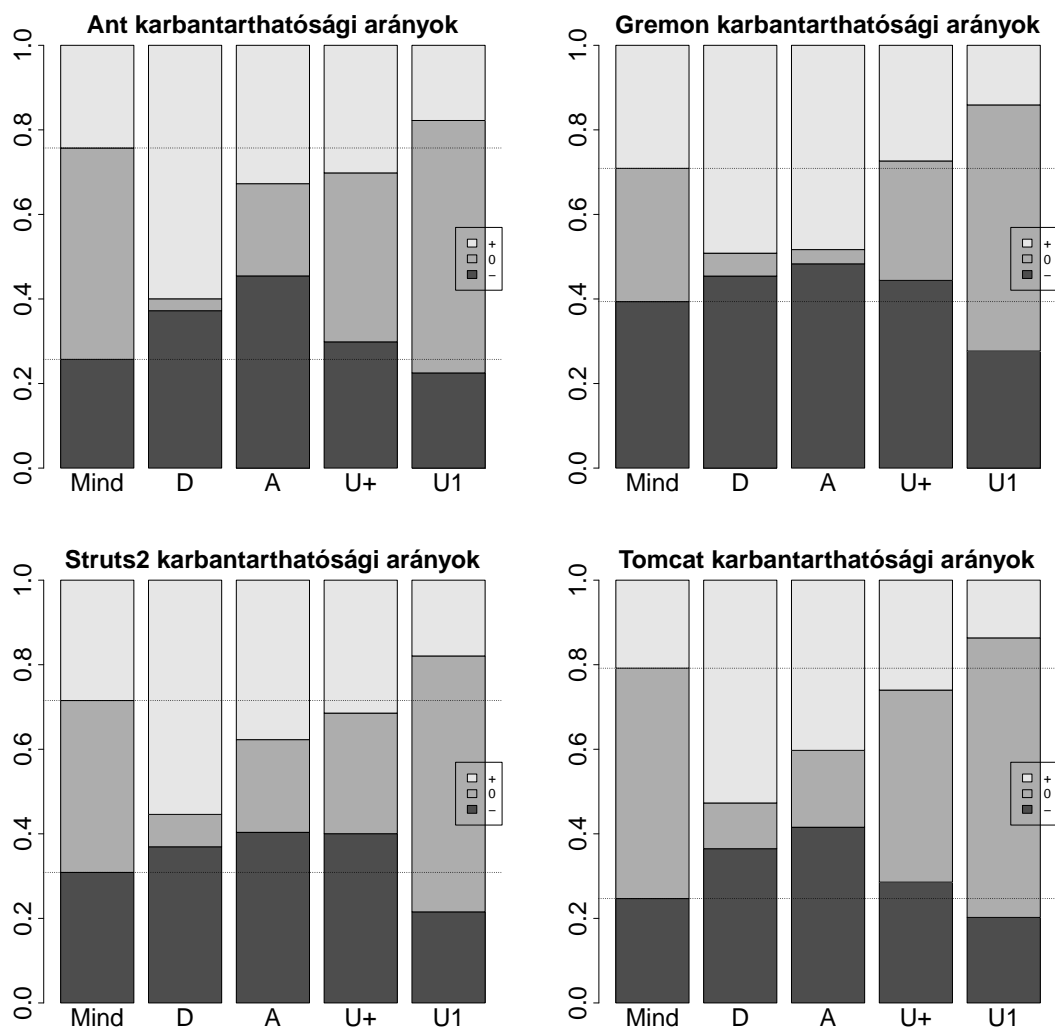
### A karbantarthatóság mérése

A szoftver egy adott verziójának karbantarthatóságát a ColumbusQM minőség-modellel határoztuk meg. Ezt Bakota Tibor publikálta szerzőtársaival [1], ami az ISO/IEC 9126 szabványra épül. Az algoritmus az alábbi forráskód metrikákat veszi figyelembe: programsorok száma, osztályok száma, beágyazottsági szint, objektumok közötti kapcsolódások száma, klón lefedettség, paraméterek száma, McCabe komplexitás, bejövő és kimenő függvényhívások száma, valamint a kódolási szabálysértések száma. Korábbi tanulmányok alapján állíthatjuk, hogy minél nagyobbak ezek az értékek (pl. minél hosszabb egy függvény), annál valószínűbb, hogy hibákat tartalmaz. Az algoritmus a metrika értékeket összehasonlítja egy külső rendszerekből álló szoftver halmaz hasonló értékeivel, végül az eredményeket összegzi.

## 2.A. A verziókövető műveletek és a karbantarthatóság kapcsolatának létezése

Mindegyik kód változásra (*commit*) meghatároztuk a karbantarthatóság változást, és ennek alapján három csoportba soroltuk őket: nőtt, nem változott vagy csökkent a karbantarthatóság. Másrészt a verziókövető műveletek szerint a következő négy kategóriába soroltuk: (D) az adott változás tartalmaz törlést; (A) nem tartalmaz törlést, de tartalmaz hozzáadást; (U+) kizárólag több módosítást tartalmaz; (U1) egyetlen módosítást tartalmaz. A kettő kombinációja egy 12 cellát tartalmazó mátrixot alkot. A mátrix cellái az adott feltételeknek megfelelő módosítások számát tartalmazzák.

Mindegyik vizsgált rendszerre végrehajtottuk a kontingencia Khi-négyzet tesztet, mely megmondja, hogy szignifikáns-e az eltérés a várható és a tényleges eloszlás között. Az elemzést négy szoftverrendszeren hajtottuk végre: három közülük nyílt forráskódú volt (Ant, Struts 2 és Tomcat), és egy ipari (Gremon).



2. ábra. Karbantarthatóság arányok

A 2. ábrán az adatok grafikus áttekintését láthatjuk, ahol művelet szerinti kategóriánként ábrázoljuk a karbantarthatóság változás kategóriáinak arányait. A világos szürke a pozitív, a közepes szürke a semleges, míg a sötét szürke a negatív karbantarthatóság változás arányát szemlélteti. A jobb áttekinthetőség érdekében az összes módosításra vonatkozó arányokat is bemutatjuk.

A teszt során a legtöbb cellára azt kaptuk, hogy szignifikáns az eltérés a várttól. Majdnem mindig ugyanabba az irányba, és többnyire hasonló mértékben tért el a várttól. Az 1. táblázat a tesztek eredményeinek p-értékeit tartalmazza.

Rendszer	p-érték	Szignifikancia
Gremon	$1,19 \cdot 10^{-52}$	nagyon erős
Ant	$1,60 \cdot 10^{-151}$	nagyon erős
Struts 2	$4,47 \cdot 10^{-64}$	nagyon erős
Tomcat	$4,84 \cdot 10^{-33}$	nagyon erős

1. táblázat. A Khi-négyzet tesztek p-értékei

Ezek az eredmények alapján megállapítottuk, hogy van kapcsolat a verziókövető műveletek és a karbantarthatóság között [14].

## 2.B. A verziókövető műveleteknek a karbantarthatóság értékére és varianciájára gyakorolt hatása

Egyenként megvizsgáltuk a fájl hozzáadás, módosítás és törlés műveletek hatását a karbantarthatóság változásának értékére [10], valamint annak varianciájára [4, 5]. A módszer során a módosításokat különböző szempontok szerint részhalmazokra bontottuk, és a hozzájuk tartozó karbantarthatóság változás értékeit hasonlítottuk össze. Mindhárom műveletre az alábbi felosztásokat vettük. Először is háromféleképpen definiáltuk a fő adathalmazt: (1) az összes változás, (2) azok a változások, melyben előfordul a kérdéses művelet, (3) azok a változások, melyek kizárólag az adott műveletből állnak. Mindegyik esetben a módosításokon a következő felosztásokat hajtottuk végre: (a) felosztás a vizsgált művelet száma szerinti medián alapján, (b) felosztás a vizsgált művelet arányának mediánja alapján, (c) a fő adathalmaz és annak komplementere. Ezzel elvileg 9, a nem valódi felosztásokat kihagyva valójában a következő 7 felosztást definiáltuk, melyeket a hozzáadás művelettel illusztrálunk:

**DIV1:** Az összes változtatás két részre osztása a vizsgált művelet számának mediánja alapján. Ezzel azt vizsgáltuk, hogy azok a változtatások, amelyek nagyszámú hozzáadás műveletet tartalmaznak jobb hatással vannak-e a karbantarthatóságra mint azok, amelyek kisebb számú hozzáadás műveletet tartalmaznak.

**DIV2:** Az összes változtatás két részre osztása a vizsgált művelet arányának mediánja alapján. Ezzel azt vizsgáltuk, hogy azoknak a módosításoknak jobb-e a karbantarthatóságra gyakorolt hatása, amelyek nagy arányban tartalmaznak hozzáadás műveletet azokhoz viszonyítva, ahol a hozzáadások aránya alacsony. A DIV1 és DIV2 felosztások közötti különbség szemléltetésére tekintsünk egy változtatást, ami 100 műveletet tartalmaz, 10 közülük hozzáadás (az abszolút száma magas, de az aránya alacsony), valamint egy olyan változtatást, ami 3 műveletet tartalmaz, és 2 közülük fájl hozzáadás (az abszolút száma alacsony, az aránya viszont magas).

**DIV3:** Az első részhalmaz azokat a változtatásokat tartalmazza, amelyek legalább egyet tartalmaznak a vizsgált műveletből, a másik pedig azokból áll, amelyek nem tartalmazzák a vizsgált műveletet. Ezzel azt vizsgáltuk, hogy azoknak a változtatásoknak, amelyek tartalmaznak fájl hozzáadást, jobb-e a karbantarthatóságra gyakorolt hatása azokhoz viszonyítva, amelyek nem tartalmazzák ezt a műveletet.

**DIV4:** *Csak azokat a változtatásokat figyelembe véve, amelyek tartalmaznak legalább egyet a vizsgált műveletből, azok felosztása a vizsgált művelet száma szerint, annak mediánjánál.* Ez hasonló a DIV1-hez, a különbség annyi, hogy nem vesszük figyelembe azokat a változtatásokat, amelyek nem tartalmaznak fájl hozzáadás műveletet. Ez a felosztás különösen a hozzáadás művelet esetén hasznos, mivel ez a művelet viszonylag ritkán fordul elő a fájl módosításhoz képest, így pontosabb összehasonlítást tesz lehetővé.

**DIV5:** *Csak azokat a változtatásokat figyelembe véve, amelyek tartalmaznak legalább egyet a vizsgált műveletből, azok felosztása a vizsgált művelet aránya szerint, annak mediánjánál.* Hasonló a DIV2-höz; az ottani magyarázat itt is érvényes.

**DIV6:** *Az első részhalmaz azokat a változtatásokat tartalmazza, amelyek csak a vizsgált műveletből állnak, a másik részhalmaz pedig azokat, amelyek tartalmaznak más műveletet is.* Ezzel azt vizsgáltuk, hogy azoknak a módosításoknak, amelyek csak fájl hozzáadást tartalmaznak, jobb-e a karbantarthatóságra gyakorolt hatásuk azokhoz viszonyítva, amelyek tartalmaznak legalább egy nem hozzáadás műveletet. Ez a felosztás a fájl módosítások vizsgálata esetén különösen hasznos.

**DIV7:** *Azoknak a változtatásoknak a felosztása, amelyek csak a vizsgált műveletet tartalmazzák, a vizsgált művelet száma szerinti felosztás, annak mediánjánál.* Ezzel a felosztással azt vizsgáltuk, hogy igaz-e az, hogy azok a változtatások, amelyek több fájl módosításból állnak, jobb a karbantarthatóságra gyakorolt hatása azokhoz viszonyítva, amelyek kevesebb számú fájl módosításból állnak. Ez a felosztás szintén a fájl módosítások esetén hasznos, mert a legtöbb változtatás kizárólag ezt a műveletet tartalmazza.

A módosítások felosztása tehát a verziókövető műveletek szerint történt, majd mindegyik felosztáshoz vettük a vonatkozó karbantarthatóság változás értékeit. A karbantarthatóság érték megváltozását nem egyszerűen az egymást követő verziók karbantarthatósági értékeinek különbségeként definiáltuk, hanem figyelembe vettük annak karakterisztikáját, valamint a program aktuális méretét is.

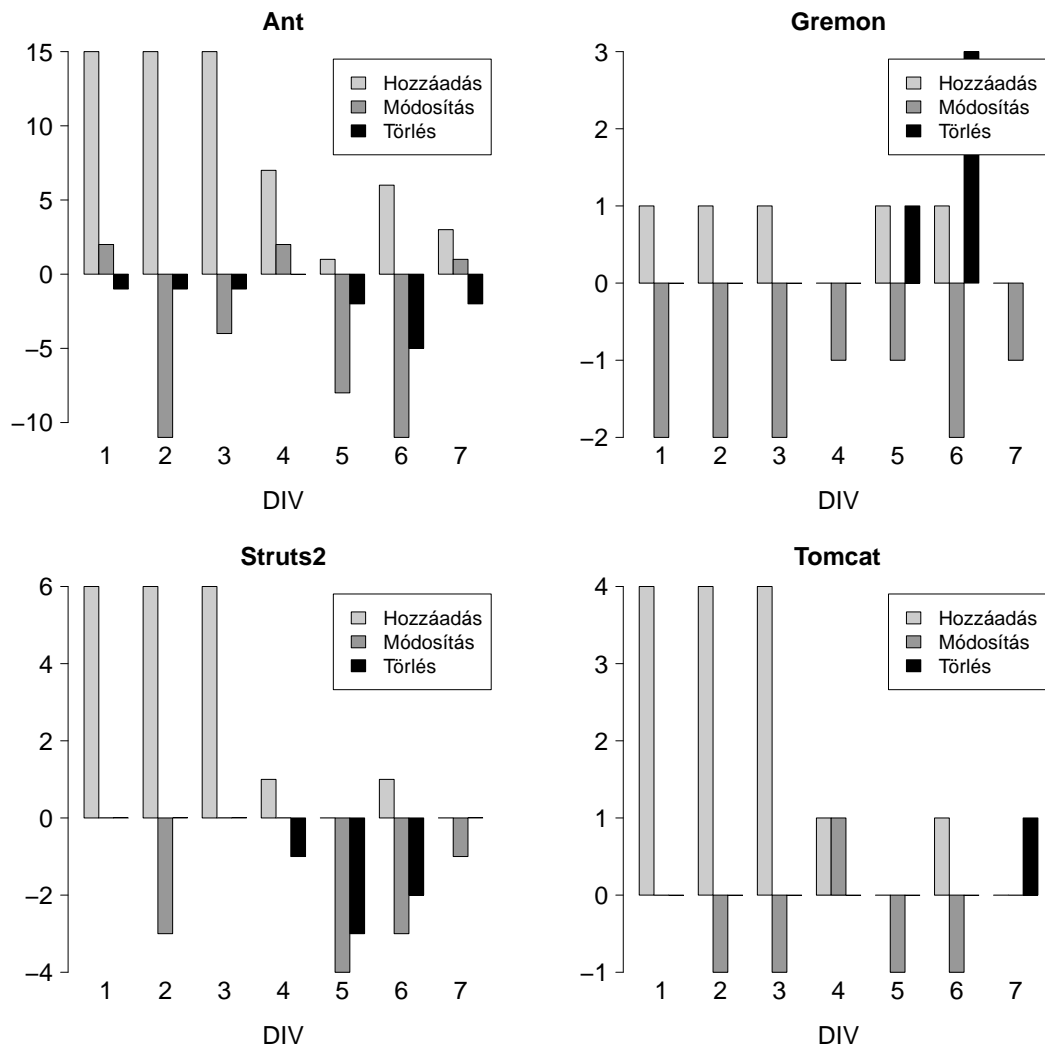
Ezeket az értékeket összehasonlítottuk. Az értékek összehasonlításánál a Wilcox tesztet alkalmaztuk, mely nem érzékeny a szélsőséges értékekre. Ezzel képet kaptunk arról, hogy mely műveleteknek milyen körülmények között milyen hatásuk van a karbantarthatóságra.

Az elemzést ugyanazokon a rendszereken hajtottuk végre, amit a 2.A tétel pontban említettünk. Eredményül azt kaptuk, hogy a fájl hozzáadások javítják a karbantarthatóságot, legalábbis kevésbé rontják, mint a fájl módosítások. A fájl módosítások ugyanis nagyrészt rontják azt. A fájl törlés hatását nem sikerült egyértelműen kimutatnunk, ott ellentmondásos eredményre jutottunk.

A 3. ábrán az összehasonlítások eredményeit illusztráljuk. A hosszú oszlopok szignifikáns eredményt jelentenek az elemzett szoftver rendszeren belül. A pozitív érték pozitív kapcsolatot jelent az aktuális verziókövető művelet és a karbantarthatóság között.

Összehasonlítottuk a két módosítás halmazhoz tartozó karbantarthatóság változás értékek varianciáit is. A variancia számítás rendkívül érzékeny a szélsőséges értékekre, márpedig a nem szokványos módosítások (például ilyen egy külön ág (*branch*) fejlesztéseinek beolvasztása (*merge*) a fő ágba) hatalmas kilengést okozhatnak. Emiatt ezeket a módosításokat a varianciák összehasonlítása során nem vettük figyelembe.

A variancia összehasonlítás szemléletes módja a varianciák hányadosa, amit a 4. ábrán szemléltetünk. Az ábra az egyes rendszerek hányados értékeinek a geometriai közepét is tartalmazza.



3. ábra. Wilcoxon teszt eredmény oszlopok

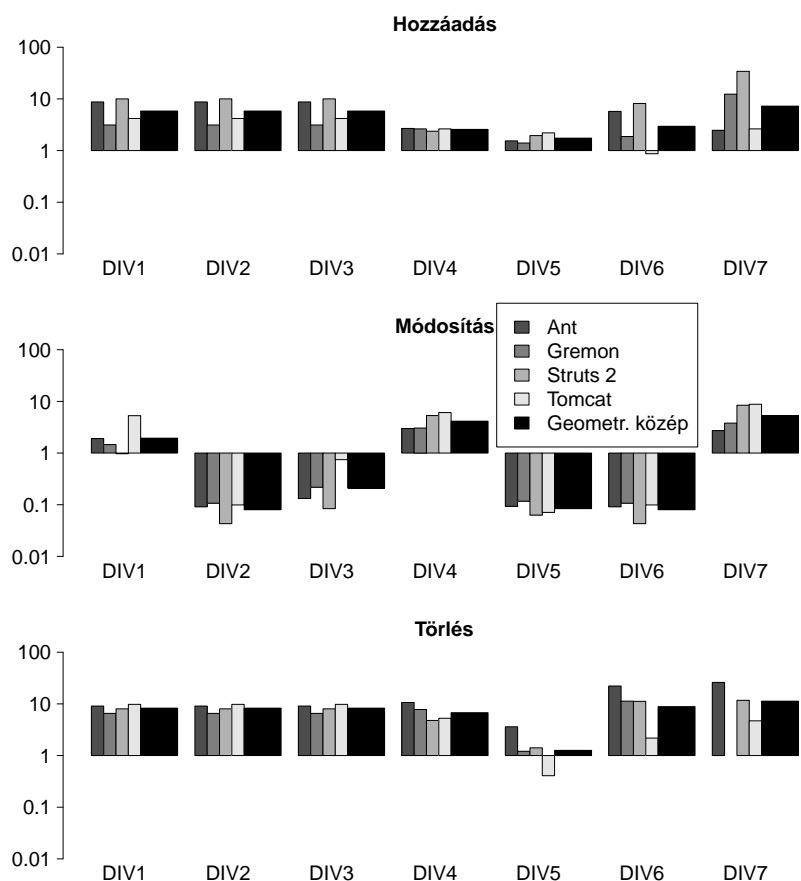
Megállapítottuk, hogy a fájl hozzáadása és a törlése növeli, míg a módosítás csökkenti a variációt. Mivel a kilengés sokkal nagyobb, mint az abszolút megváltozás, végső konklúzióként arra jutottunk, hogy a fájl hozzáadásokra érdemes különösen odafigyelni.

## 2.C. Halmozódó karakterisztika diagram és kvantilis különbség diagram

Kifejlesztettünk két vizualizációs módszert, amelyek alkalmasnak bizonyultak egyes publikált eredmények képi megjelenítésére [6].

A *halmozódó karakterisztika diagram* bemenete egy számhalmaz. Ezeket nem növekvő sorrendbe rendezzük, majd minden egyes indexre kiszámoljuk azok összegét az elsőtől az adott indexig. A indexek képezik az x-tengelyt, az összegek pedig az y-t. A karakterisztika úgy keletkezik, hogy ezeket a pontokat összeköltjük. Az összetett halmozódó karakterisztika diagramon kettő vagy több halmozódó karakterisztikát ábrázolunk.

E diagram típus alkalmasnak bizonyult a kontingencia Khi-négyzet teszt, a Wilcoxon teszt és a variancia teszt illusztrálására. Az 5. ábrán a 2.A. tézispont bemenő adatait illusztráljuk. Ez az említett felosztásokkal kapcsolatos karbantarthatóság változásokat ábrázolja. A diagramokon belüli különbségek,

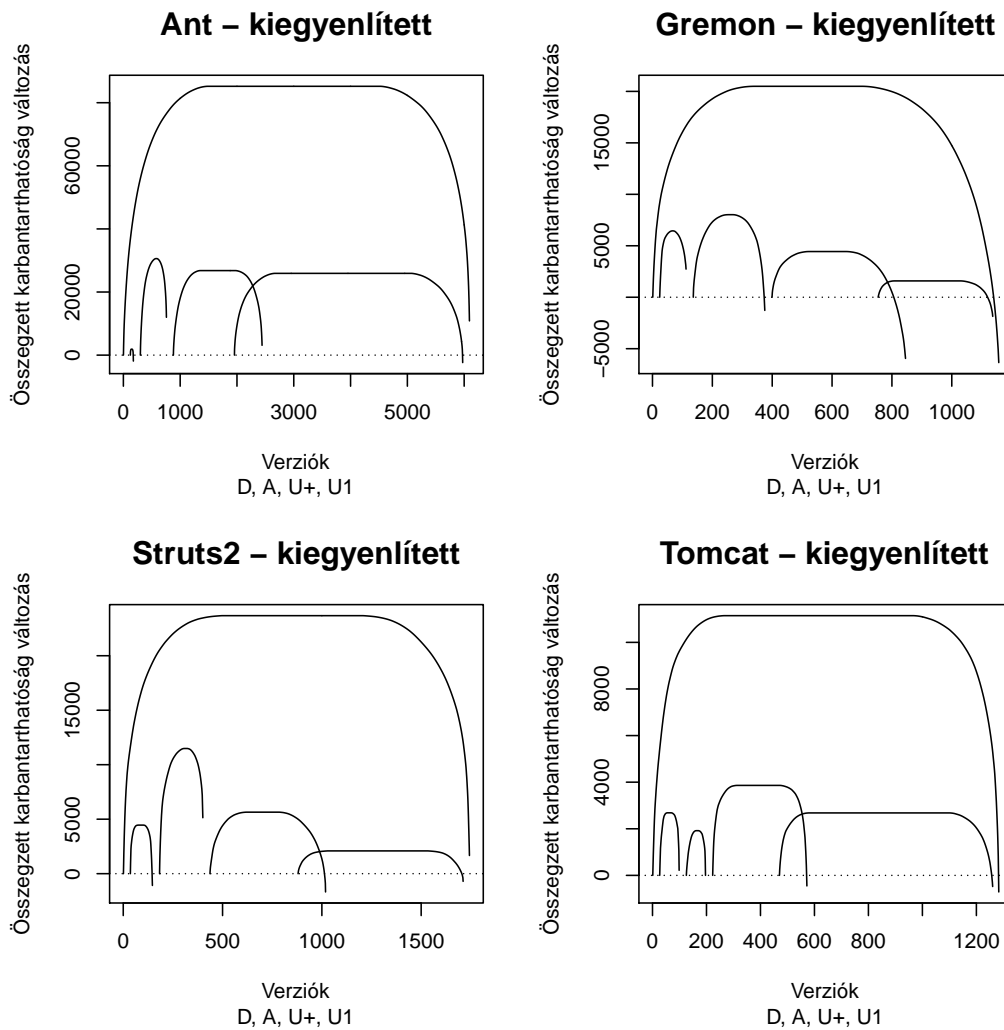


4. ábra. A varianciák illusztrálása

valamint a diagramok közötti hasonlóságok alátámasztják azt az eredményt, hogy létezik kapcsolat a verziókövető utasítások száma és a vonatkozó karbantarthatóság változás között. Például a középen levő kisebb görbék közül a második jobb végpontja mindegyik esetben a két jobb oldali görbe jobb végpontja fölött helyezkedik el, ami arra utal, hogy a hozzáadás műveletnek jobb hatása van a karbantarthatóságra, mint a módosítás műveletnek. Egy másik szembeűnő jellemző az, hogy az első két görbe szélessége jóval kisebb, mint a második két görbéé, ugyanakkor a magasságuk a legtöbb esetben nagyságrendileg összevethető, és ez azt jelenti, hogy a fájl törlésre és fájl hozzáadásra vonatkozó értékek varianciája jóval nagyobb, mint a módosításra vonatkozó értékeké.

A kvantilis különbség diagram segítségével két számhalmazt tudunk összehasonlítani. Ehhez külön-külön lerendezzük mindkét halmaz elemeit nem csökkenő sorrendben. Majd mindkét halmazból páronként vesszük mindegyik centilishez tartozó értékeket. Tehát például a medián a mediánnal lesz párban, a 90% a másik 90%-kal stb. Mindegyik pár esetén képezzük azok különbségét. A centilisekből képezve az x-koordinátát, a különbségekből az y-koordinátát, a keletkező pontokat pedig egyenes szakaszokkal összekötve kapjuk a kvantilis különbség diagramot. Ha az adathalmaz tartalmaz szélsőséges értékeket, célszerű a széleket nem ábrázolni; az alapértelmezett megvalósítás nem veszi figyelembe az alsó és a felső 5%-ot.

E diagram típus alkalmasnak bizonyult a Wilcoxon teszt és a variancia teszt illusztrálására. A 6. ábrán a 2.B. tézispont egyik eredményét illusztráljuk. A következő karbantarthatóság változás értékek összehasonlításáról van szó: változtatások, melyek tartalmaznak, ill. nem tartalmaznak fájl hozzáadást. Az a



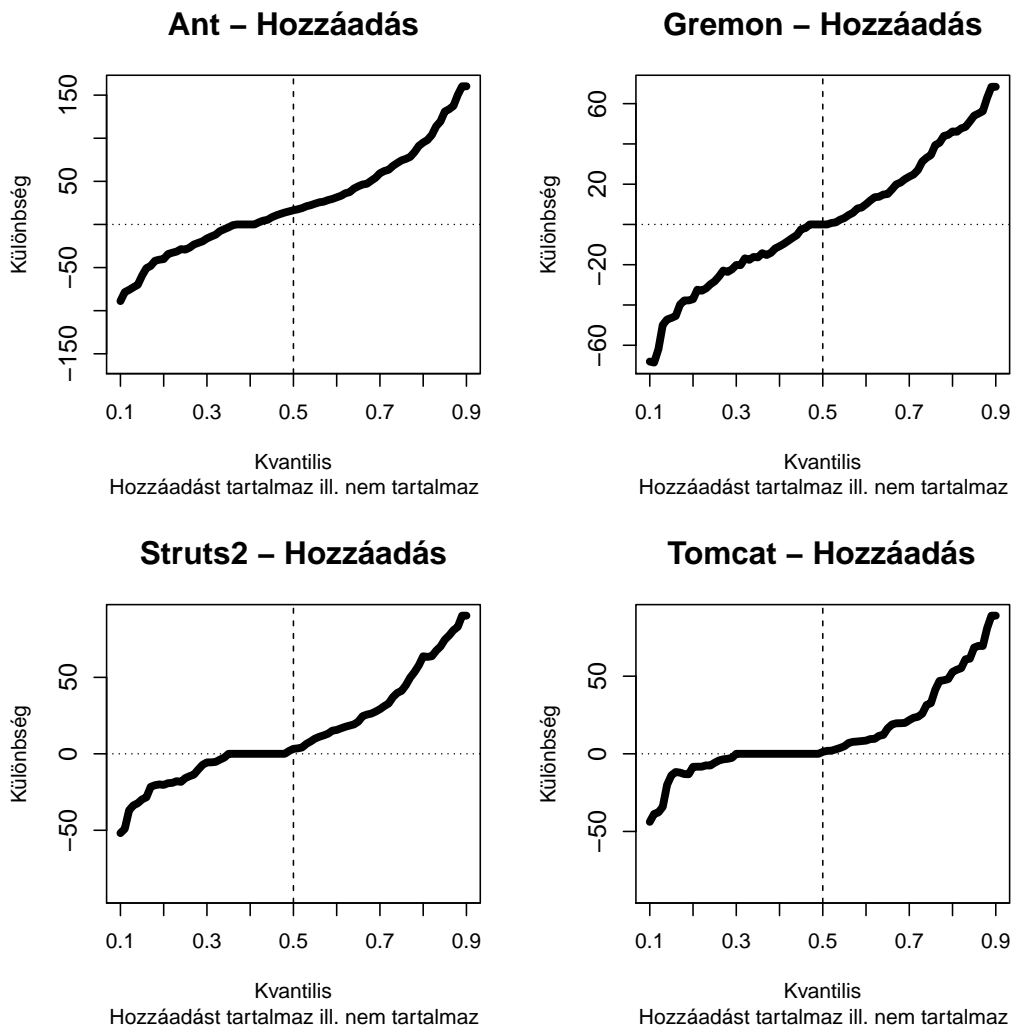
5. ábra. Összetett halmozódó karakterisztika diagram karbantarthatósági értékekkel

tény, hogy a görbék nagyobb része az x-tengely felett helyezkedik el azt jelzi, hogy a változtatások, melyek tartalmazznak fájl hozzáadást, jobb karbantarthatóság változást okoznak, mint azok, amelyek nem tartalmazznak fájl hozzáadást. A görbe meredeksége azt jelzi, hogy a két részhalmoz karbantarthatóság változás értékeinek varianciája eltérő.

## A szerző hozzájárulása az eredményekhez

Az értekezés szerzője az alábbiakkal járult hozzá az ebben a fejezetben bemutatott új eredményekhez:

- A verziókövető műveletek és a karbantarthatóság változás közötti kapcsolat módszerének kidolgozása, megvalósítása, végrehajtása és az eredmények kiértékelése. A módszertan az alábbiakat tartalmazza: a módosítások felosztásának ötlete egy főkomponens elemzés alapján; a kontingencia Khi-négyzet teszt alkalmazása a *karbantarthatóság változás* és *verziókövető műveletek alapján meghatározott kategóriák* mátrixra. Az eredmények kiértékelése magába foglalja a p-érték exponense alapján történő bemutatást, oszlopdiagramokkal megjelenítve.
- A verziókövető műveletek karbantarthatóságra gyakorolt hatásának módszertana, a megvalósítás, a végrehajtás és az eredmények kiértékelése. A módszertan a következőket tartalmazza: a



6. ábra. Halmozódó karakterisztika diagram fájl hozzáadást tartalmazó és nem tartalmazó módosítások karbantarthatóság változásainak összehasonlítására

módosítások hétféleképpen történő felosztása a verziókövető műveletek alapján, a Wilcox teszt és az F teszt alkalmazása ezekre a felosztásokra. Kiértékelés alatt a következőket értjük: a standard eltérések kiszámítása és oszlopdiagramokkal történő ábrázolása.

- A halmozódó karakterisztika diagram és a kvantilis különbség diagram ötlete, megvalósítása R-ben, a `vudc` R csomag [7] karbantartása.

## Harmadik tézis pont: Verziókövető történeti metrikák és a karbantarthatóság kapcsolata

Továbbléptünk a verziókövető rendszerben található információk elemzésével kapcsolatban. A korábbival ellentétben figyelembe vettük azt is, hogy az adott információ mely fájlra vonatkozik, ezáltal kapcsolatot teremtve különböző forráskód módosítási egységek között.

Először a forráskód múltbeli módosításai intenzitásának, valamint a kód tulajdonlás mértékének a hatását vizsgáltuk a későbbi karbantarthatóság megváltozására. Ezt követően definiáltunk 6 verziókövető történeti metrikát, és mindegyikre megvizsgáltuk annak kapcsolatát a karbantarthatósággal, és – mintegy ellenőrzésképpen – az utólagosan javított hibák számával.

### 3.A. A kód módosításainak és a kód tulajdonlásnak a karbantarthatóságra gyakorolt hatása

Megvizsgáltuk, hogy a múltbeli kód módosítás halmozódó intenzitása (*cumulative code churn*) [12], valamint a kódot módosító fejlesztők száma [11] milyen hatással vannak a jelenlegi módosításnak a karbantarthatóságra gyakorolt hatására.

Minden egyes fájlra és verzióra kiszámoltuk azt, hogy kezdettől fogva összesen hány sort adtak hozzá és hány sort töröltek. Adott módosításnál ezeket az értékeket átlagoltuk. Az így kiszámolt értékeket két csoportba osztottuk aszerint, hogy a vonatkozó módosítás csökkentette vagy növelte a karbantarthatóságot (a karbantarthatóság változás szempontjából semleges esetekkel nem foglalkoztunk). Végül Wilcox teszt segítségével hasonlítottuk össze az értékeket.

A kód tulajdonlás esetén is hasonlóan jártunk el. Ott azt vizsgáltuk, hogy adott fájlt összesen hány különböző fejlesztő módosított, adott módosításnál pedig a geometriai közepét vettük. Az összehasonlítás itt is a fentihez hasonlóan, a Wilcox teszt segítségével történt.

Az elemzést ugyanazokon a rendszereken hajtottuk végre, mint a második tézis pontban. Eredményül azt kaptuk – ahogyan az a 2. táblázatban látható –, hogy a múltbeli intenzív módosítás és az egyértelmű kódtulajdonlás hiánya is a karbantarthatóság csökkenését vetíti előre.

Rendszer	Kód módosítás intenzitás		Kód tulajdonlás	
	p-érték	Szignifikancia	p-érték	Szignifikancia
Ant	0,00235	nagyon erős	0,03347	erős
Gremon	0,00436	nagyon erős	0,05960	szignifikáns
Struts 2	0,00018	nagyon erős	0,00001	nagyon erős
Tomcat	0,03616	erős	0,21384	nem szignifikáns

2. táblázat. A kód módosítás intenzitás és a kód tulajdonlás tesztek eredményei

### 3.B. Verziókövető történeti metrikák és a karbantarthatóság korrelációja

Végül definiáltunk hat verziókövető metrikát, és megvizsgáltuk azok kapcsolatát a karbantarthatósággal [13]. Ezek a metrikák a következők: halmozódó változás intenzitás, módosítások száma, módosítók száma, módosítók száma toleranciával, a kód kora és az utolsó módosítás időpontja.

A vizsgált szoftver adott verziójára mindegyik metrika szerint sorba rendeztük a forrásfájlokat. Ez a forrásfájlok hatféle sorrendjét eredményezte. A fájlokat sorba rendeztük a relatív karbantarthatósági index [16] alapján is. Az alapötlet dióhéjban az alábbi: végrehajtjuk a karbantarthatóság elemzést a teljes rendszeren, majd utána úgy, hogy kivesszük belőle az elemzett forráskód elemet. A relatív karbantarthatósági index az eredeti karbantarthatósági érték és a vizsgált forráskód elem nélküli karbantarthatósági érték különbsége. Ellenőrzésképpen felállítottuk a forrásfájlok sorrendjét annak alapján is, hogy melyik fájlban hány hibát találtak az adott verzió kiadását követően (*post-release bugs*).

A sorrendek hasonlóságát a Spearman sorrend korrelációs teszt segítségével állapítottuk meg. Az alábbi nyílt forráskódú szoftverrendszereken hajtottuk végre az elemzést: az Ant 5 verziója, a jEdit 4 verziója, a Log4J 3 verziója és a Xerces 2 verziója, összesen tehát 4 rendszer 14 verzióját elemeztük.

Az egyes metrikák alapján felállított sorrend és a relatív karbantarthatósági index által meghatározott sorrend összehasonlításainak eredményeit a 3. táblázat tartalmazza. Az eredmények alapján megállapítottuk, hogy a nagyobb változás intenzitás, a módosítások valamint a módosítók (tolerancia nélküli és toleranciával számított) magasabb száma, a régebbi kód és a friss utolsó módosítás rosszabb karbantarthatóságot és nagyobb számú hibát eredményez.

Rendszer	Verzió	Intenzitás	Módosítások	Tulajdonlás	Tol.tul.	Hozzáadott	Ut.mód.
Ant	1.3	-0,861	-0,598	-0,392	-0,556	0,239	-0,563
	1.4	-0,867	-0,656	-0,475	-0,609	0,339	-0,373
	1.5	-0,747	-0,631	-0,550	-0,628	0,269	-0,592
	1.6	-0,852	-0,719	-0,636	-0,704	0,276	-0,464
	1.7	-0,702	-0,612	-0,560	-0,532	0,279	-0,268
jEdit	4.0	-0,712	-0,506	NA	-0,160	0,098	-0,442
	4.1	-0,681	-0,552	-0,515	-0,461	0,105	-0,466
	4.2	-0,713	-0,505	NA	-0,103	0,091	-0,478
	4.3	-0,302	-0,570	-0,488	-0,553	0,226	-0,044
Log4J	1.0	-0,823	-0,351	NA	-0,055	0,221	-0,283
	1.1	-0,873	-0,779	-0,556	-0,504	0,227	-0,535
	1.2	-0,854	-0,410	-0,481	-0,362	0,167	-0,102
Xerces	1.3	-0,660	-0,468	-0,217	-0,430	0,069	-0,100
	1.4	-0,481	-0,523	-0,322	-0,455	0,151	-0,355

3. táblázat. A Spearman korrelációs elemzés  $\rho$  értékei

## A szerző hozzájárulása az eredményekhez

Az értekezés szerzője az alábbiakkal járult hozzá az ebben a fejezetben bemutatott új eredményekhez:

- A kód módosítás intenzitás és a kód tulajdonlás elemzések módszereinek kidolgozása, a statisztikai tesztek végrehajtása, az eredmények kiértékelése.
- A hat verziókövető metrika definiálása. A verziókövető metrikákat kinyerő program megvalósítása. A relatív karbantarthatósági indexszel és a verzió kiadását követő hibák számával történő korrelációs teszt módszerének kidolgozása, megvalósítása, a tesztek végrehajtása és az eredmények kiértékelése.

# Összegzés

Ebben az értekezésben két nagyobb témát érintettünk: a programszeletelést és a karbantarthatóság elemzést.

A *programszeletelés* kutatási terület óriási és érett. Az értekezésben bemutatott eredmények összegzését a következőképpen foglalhatjuk össze: egy bizonyos dinamikus programszeletelő algoritmus tovább finomított verziójához kapcsolódó nem strukturált utasítások kezelése. Tehát az értekezésben bemutatott munka olyan, mint egy csavar a gépezetben.

A *verziókövető metrikák és a szoftver karbantarthatóság kapcsolata* témaköre – szűkebb értelemben véve – egy fiatal kutatási terület. A szoftver karbantarthatóság és a szoftver verziókövető rendszerekből történő adatbányászás témaköre között helyezkedik el.

A szoftver karbantarthatóság a régebbi és nagyobb; a jelentősebb cikkeknek mintegy fele már az ezredforduló előtt megjelent. A tanulmányok jellemzően forráskód metrikákkal és hiba előrejelzéssel foglalkoznak. A szoftver verziókövető rendszerekből történő adatbányászás egy nagy lehetőségeket tartalmazó újabb és fejlődő terület; a cikkek elsősorban többsége 2000 után jelent meg. Némely közülük szintén a hiba előrejelzésről szól.

Nincs tudomásunk más olyan publikációról, amely a verziókövető adatok és olyan értékek kapcsolatát vizsgálja, amelyet egy szoftver karbantarthatósági modell számít. Ezért az értekezés második része egy új és még kis terület, mely nagy lehetőségeket rejt magában.

Először kinyitottuk a dobozt és megmutattuk, hogy létezik a kapcsolat a verziókövető történeti adatok és a szoftver karbantarthatóság között. Kezdetben csak a verziókövető műveleteket vettük figyelembe, később pedig más adatokkal is számoltunk, úgy mint a fájl neve, a fejlesztő neve vagy a módosítás dátuma.

Úgy gondoljuk, hogy jelentős lehetőségek vannak még ezen a területen. Mindenekelőtt a verziókövető történeti adatokat még nem sikerült teljesen kiaknázni: még számos metrikát lehet definiálni és elemezni. Az értékek aggregálása – a minőségi modellekhez hasonlóan, amelyek a forráskód metrikákat aggregálják – szintén egy teljesen nyitott téma.

Összefoglalva, az első részben tettünk egy lépést előre egy nagy kutatási területen, míg a második részben úttörő munkát végeztünk egy fiatal kutatási területen.

A 4. táblázatban az egyes tézispontokhoz tartozó publikációkat találjuk.

	[8]	[9]	[2]	[14]	[10]	[4]	[5]	[6]	[12]	[11]	[13]	[16]
1.	•	•	•									
2.A				•								
2.B					•	•	•					
2.C								•				
3.A									•	•		
3.B											•	•

4. táblázat. A tézispontokat alátámasztó publikációk

# Köszönetnyilvánítás

Köszönetet mondok témavezetőmnek, Dr. Ferenc Rudolfnak azért az önzetlen segítségért, amit távolból nyújtott az egyéni felkészülésem során. Szintén köszönetet mondok Dr. Hegedűs Péternek a gyümölcsöző virtuális együttműködésért. Kiváló volt az együttműködés az eseti szerzőtársaimmal is: Dr. Bakota Tiborral, Ladányi Gergellyel és Végh Ádám Zoltánnal.

Több mint egy évtizeddel korábban a programszeletelés témakörében nagyszerű volt a közös munka Dr. Beszédes Árpáddal, Dr. Gergely Tamással és Szabó Zsolt Mihállyal, Dr. Csirik János és Dr. Gyimóthy Tibor vezetése mellett. Velük nagyon hasznos programozási tapasztalatra tettem szert.

Munkahelyem, a Lufthansa Systems Hungária Kft. anyagilag és szabadidő formájában is támogatta egyes kutatásaimat. Az angol nyelvű szöveget Bruce Derby lektorálta. Mindezt nem tudtam volna végigcsinálni a feleségem, Lilla bátorítása, türelme és támogatása nélkül. Köszönöm mindenkinek, akik segítettek a kutatási tevékenységeimet!

*Faragó Csaba, 2016*

# Hivatkozások

- [1] Tibor Bakota, Péter Hegedűs, Péter Körtvélyesi, Rudolf Ferenc, and Tibor Gyimóthy. A probabilistic software quality model. In *Proceedings of the 27th International Conference on Software Maintenance (ICSM)*, pages 243–252. IEEE Computer Society, 2011.
- [2] Árpád Beszédes, Csaba Faragó, Zsolt Mihály Szabó, János Csirik, and Tibor Gyimóthy. Union slices for program maintenance. In *Proceedings of the 18th International Conference on Software Maintenance (ICSM)*, pages 12–21. IEEE Computer Society, 2002.
- [3] Árpád Beszédes, Tamás Gergely, Zsolt Mihály Szabó, Janos Csirik, and Tibor Gyimothy. Dynamic slicing method for maintenance of large C programs. In *Proceedings of the 5th European Conference on Software Maintenance and Reengineering (CSMR)*, pages 105–113. IEEE Computer Society, 2001.
- [4] Csaba Faragó. Variance of source code quality change caused by version control operations. In *Proceedings of the 9th Conference of PhD Students in Computer Science (CSCS)*, pages 12–13, 2014.
- [5] Csaba Faragó. Variance of source code quality change caused by version control operations. *Acta Cybernetica*, 22(1):35–56, 2015.
- [6] Csaba Faragó. Visualization of univariate data for comparison. *Annales Mathematicae et Informaticae*, 45:39–53, 2015.
- [7] Csaba Faragó. *vudc: Visualization of Univariate Data for Comparison*, 2016. R package version 1.1.
- [8] Csaba Faragó and Tamás Gergely. Handling the unstructured statements in the forward dynamic slice algorithm. In *Proceedings of the 7th Symposium on Programming Languages and Software Tools (SPLST)*, pages 71–83, 2001.

- [9] Csaba Faragó and Tamás Gergely. Handling pointers and unstructured statements in the forward computed dynamic slice algorithm. *Acta Cybernetica*, 15(4):489–508, 2002.
- [10] Csaba Faragó, Péter Hegedűs, and Rudolf Ferenc. The impact of version control operations on the quality change of the source code. In *Proceedings of the 14th International Conference on Computational Science and Its Applications (ICCSA)*, volume 8583 Lecture Notes in Computer Science (LNCS), pages 353–369. Springer International Publishing, 2014.
- [11] Csaba Faragó, Péter Hegedűs, and Rudolf Ferenc. Code ownership: Impact on maintainability. In *Proceedings of the 15th International Conference on Computational Science and Its Applications (ICCSA)*, volume 9159 Lecture Notes in Computer Science (LNCS), pages 3–19. Springer International Publishing, 2015.
- [12] Csaba Faragó, Péter Hegedűs, and Rudolf Ferenc. Cumulative code churn: Impact on maintainability. In *Proceedings of the 15th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pages 141–150. IEEE Computer Society, 2015.
- [13] Csaba Faragó, Péter Hegedűs, Gergely Ladányi, and Rudolf Ferenc. Impact of version history metrics on maintainability. In *Proceedings of the 8th International Conference on Advanced Software Engineering & Its Applications (ASEA)*, pages 30–35. IEEE Computer Society, 2015.
- [14] Csaba Faragó, Péter Hegedűs, Ádám Zoltán Végh, and Rudolf Ferenc. Connection between version control operations and quality change of the source code. *Acta Cybernetica*, 21(4):585–607, 2014.
- [15] Tibor Gyimóthy, Árpád Beszédes, and István Forgács. An efficient relevant slicing method for debugging. In *Proceedings of the Joint 7th European Software Engineering Conference and 7th ACM SIGSOFT International Symposium on the Foundations of Software Engineering (ESEC/FSE)*, pages 303–321. Springer International Publishing, 1999.
- [16] Péter Hegedűs, Tibor Bakota, Gergely Ladányi, Csaba Faragó, and Rudolf Ferenc. A drill-down approach for measuring maintainability at source code element level. *Electronic Communications of the EASST*, 60:1–21, 2013.