Szegedi Tudományegyetem
Szoftverfejlesztés Tanszék

Szoftver termék metrikák alkalmazása a szoftverkarbantartás területén

Ph.D. értekezés tézisei

Siket István

Témavezető:

Dr. Gyimóthy Tibor

Szeged
2010

Bevezetés

A szoftverfejlesztés egyik legnagyobb kihívása a megrendelők növekvő igényeinek kielégítése. Egyre nagyobb és komplexebb rendszereket kell kifejleszteni rövid idő alatt, és miután átadtuk a megrendelőnek, karban kell tartani azokat, illetve tovább kell fejleszteni az új igények figyelembe vételével. A szoros határidők és a komplex feladatok következtében a fejlesztőknek nem áll módjukban, hogy minden esetben a legtökéletesebb megoldást alkalmazzák, melynek következtében a szoftver minősége egyre rosszabb lesz. Mivel azokra a minőségi jellemzőkre kell leginkább odafigyelni, amelyek a megrendelők számára fontosak (például a használhatóság vagy a megbízhatóság), ezért a fejlesztők feláldozzák a felhasználók számára kevésbé fontos „belső minőséget" (például a karbantarthatóságot). Viszont a „belső minőség" legalább annyira fontos, mint a többi, így egyiket sem szabad elhanyagolnunk. Például egy hiba megtalálásának és kijavításának vagy egy új funkció kifejlesztésének a költsége erősen függ a szoftver karbantarthatóságától. Ha nem figyelünk oda a szoftver minőségére, akkor idővel nem tudjuk hatékonyan karbantartani vagy fejleszteni azt.

Az értekezésben a szoftverkarbantartással, és azon belül is a teszteléssel és a hibák megtalálásával foglalkozunk. Mivel a szoftverfejlesztési költségeknek egy jelentős részét ez teszi ki, ezért bármilyen módszer, ami a tesztelés hatékonyságát növeli, egyszerre növeli a szoftver minőségét és csökkenti a fejlesztés költségét. Korábban számos vizsgálat [1, 3, 6, 17, 18] igazolta, hogy van kapcsolat az objektum-orientált metrikák és az osztályokban található hibák száma között. Ez azt jelenti, hogy a metrikák folyamatos mérésével és figyelésével a szoftver minősége és karbantarthatósága is növelhető, illetve a tesztelés sokkal hatékonyabbá tehető.

A korábbi vizsgálatok egyik gyengesége, hogy csak kis vagy közepes méretű rendszereket használtak, azaz az összefüggéseket még nem igazolták ipari környezetben. A másik probléma, amit meg kellett oldani a metrikák mindennapi alkalmazhatóságához, hogy nagy és komplex rendszereket nem tudtunk elemezni, melynek következtében nem tudtuk kiszámolni a metrika értékeket. Ezen problémák megoldása nélkül a metrikák nem alkalmazhatóak a szoftverfejlesztési folyamatban.

Az értekezésben mindkét említett problémára mutatunk megoldást. Először a Columbus technológiát ismertetjük, amely lehetővé teszi a nagy rendszerek elemzését és az objektum-orientált metrikák kiszámítását még az általunk vizsgált legnagyobb rendszerek esetében is. A kísérlethez a Mozilla [10] 7 különböző verzióját elemeztük, és kiszámoltuk a metrika értékeket. Emellett minden bejelentett és kijavított hibát összegyűjtöttünk a Mozilla hibakövető rendszeréből, azaz a Bugzillából [4], és a kidolgozott heurisztikánk segítségével az osztályokhoz rendeltük azokat. Ezzel bebizonyítottuk, hogy az általunk kidolgozott technológia alkalmas a gyakorlati használatra, továbbá minden adat rendelkezésünkre állt ahhoz, hogy mi magunk is elemezzük a metrikák és a hibák kapcsolatát. Először a Chidamber és Kemerer által definiált metrikákat [5], valamit a hagyományos sorok száma metrikát vizsgáltuk. Később kiterjesztettük a kísérletet, melyben az egyes metrikák helyett metrika-kategóriák vizsgálatunk. A kapott eredményeket felhasználva metrika-alapú hiba-előrejelző modelleket készíthetünk, melyek segíthetik a szoftverfejlesztés folyamatát.

Végezetül a metrikák gyakorlati alkalmazhatóságát vizsgáltuk, pontosabban a fejlesztőket kérdeztük 4 metrika és a szoftver megértés és tesztelés kapcsolatáról. Az eredmények rávilágítottak a metrikák gyakorlati alkalmazásának néhány problémájára, mely szerint nem adhatjuk oda a fejlesztőknek a metrikákat a megfelelő oktatás nélkül. Az eredményeknek egy másik lehetséges felhasználása, hogy a minőségi modellek hatékonysága is növelhető a fejlesztők véleményének figyelembe vételével.

Az értekezés négy fő eredménye a következő:

1. **Nagy rendszerek elemzésének kidolgozása, továbbá egy nyelvfüggetlen modell kialakítása, mely alkalmas az objektum-orientált nyelvek közös reprezentációjára, majd ennek felhasználása az objektum-orientált metrikák kiszámításában. Egy heurisztika kidolgozása a hibák osztályokhoz rendelésére.**

2. **Objektum-orientált tervezési metrikákon alapuló hiba-előrejelző modellek kidolgozása.**

3. **A metrika-kategóriák hiba-előrejelző képességeinek vizsgálata.**

4. **A szoftverfejlesztők véleményének megismerése, és az eredmények felhasználása a minőségi modellek javítására.**

1. A Columbus technológia

Ahhoz, hogy vizsgálni tudjuk a kapcsolatot az objektum-orientált metrikák és a hibák száma között, először meg kell határoznunk ezeket az értékeket. Bemutatjuk azt, hogy milyen megoldást dolgoztunk ki a nagy rendszerek elemzésére, majd röviden ismertetjük azt a nyelvfüggetlen objektum-orientált modellt, amely lehetővé teszi, hogy kiszámoljuk az objektum-orientált metrikákat még a legnagyobb rendszerek esetében is. Ezek segítségével a Mozilla 9 verzióját elemeztük, majd kiszámoltuk a metrikákat, továbbá egy heurisztika segítségével meghatároztuk az osztályokban talált és kijavított hibák számát.

A Columbus keretrendszer

Nagyon nehéz és összetett feladat egy ipari méretű szoftver elemzése. A forráskód általában fájlokra van osztva, a fájlok meg könyvtárakba és alkönyvtárakba vannak szervezve. Az az információ, hogy ezek a fájlok hogyan kapcsolódnak egymáshoz, és milyen további információ szükséges a fordításukhoz különböző makefile-okban és projekt fájlokban van eltárolva. Mindezen információkra szükségünk van, ha a rendszert le akarjuk fordítani. Sajnos ezek a fájlok nagyon különbözőek lehetnek, és szinte bármilyen információt tárolhatnak, ezért igen nehéz „megérteni" ezeket. Ez csak egy példa azok közül, amelyeket meg kell oldanunk ahhoz, hogy ipari méretű rendszereket is tudjuk elemezni.

Miután számos rendszert megvizsgáltunk, kifejlesztettük a Columbus keretrendszert [7], amely lehetővé teszi, hogy automatikusan elemezzünk egy tetszőleges rendszert anélkül, hogy bármit módosítanánk annak forráskódján. Annak ellenére, hogy az első változata csak C++ nyelven írt és GCC-vel Linux alatt fordítható rendszereket tudott elemezni, ma már képes Windows alatt is működni, támogatja többek között a Microsoft Visual Studiot, továbbá támogatja a Java és a C# nyelveket is. Több nyílt forráskódú (például Mozilla [10] vagy OpenOffice.org [12]) és ipari rendszert is sikeresen elemeztünk, melyek közül a legnagyobb 30 millió programsorból állt, ami bizonyítja a technológia használhatóságát.

Az kinyert információ felhasználható különböző visszatervezési célokra [14]. Most csak azt nézzük meg, hogy az objektum-orientált metrikákat hogyan lehet kiszámolni.

A nyelvfüggetlen modell

Kidolgoztunk egy olyan *nyelvfüggetlen modellt* (*Language Independent Model*, vagy röviden *LIM*), amely alkalmas az objektum-orientált nyelvek magas szintű ábrázolására. A modell négy nagy csomagból áll. Az *alap csomag* általános osztályokat tartalmaz, mint például a modellben található összes osztály közös ősosztálya, vagy a kommenteket ábrázoló osztály. A *fizikai csomag* olyan osztályokból áll, amelyek a rendszer fizikai struktúráját ábrázolhatják, azaz a könyvtárakat, fájlokat, illetve azok kapcsolatait (például az egyik fájl include-olja a másikat). A *logikai csomag* osztályai reprezentálják a rendszer főbb objektum-orientált elemeit (például csomagokat, osztályokat és metódusokat), illetve azok tulajdonságait és kapcsolatait. Ezen kívül néhány olyan alacsony szintű információt is eltárolunk ezekben az osztályokban, amelyek szükségesek a metrikák kiszámításához (például a metóduson belüli elágazások száma), de a LIM-en már nem lehet előállítani azokat. A *típus csomag* egyesíti a C++, Java és C# nyelvek típusreprezentációját egy picit egyszerűbb formában, így a „legextrémebb és legritkábban használt típusoktól" eltekintve minden további felépíthető. A metrika kiszámolásához nem lenne szükségünk ennyire pontos típusreprezentációra, de például a forráskód dokumentáció előállítására is a LIM-et használjuk, amelynél elengedhetetlen a típusok pontos ismerete.

A LIM egyik előnye a nyelvi reprezentációkkal szemben, hogy sokkal kevesebb memóriát használ, melynek következtében olyan nagy rendszerek is ábrázolhatóak a LIM-en, amelyeket a C++ sémában már nem. A másik előnye, hogy a magas szintű ábrázolásnak köszönhetően az objektum-orientált „eredményeket" sokkal egyszerűbben és gyorsabban elő lehet állítani. Továbbá, ha kifejlesztünk egy újabb programot, amely a LIM-en dolgozik, akkor működni fog minden olyan nyelvre, amelyet átkonvertálunk LIM-be. A másik előnye, hogy a különböző nyelvekre megírt programok helyett csak egy programot kell karbantartanunk.

Megvalósítottuk a C++, Java és C# nyelvek konverzióját LIM-re, így ezen nyelvek esetében ezt használjuk többek között az objektum-orientált metrikák kiszámítására. Jelen pillanatban 16 rendszer szintű, 65 osztály szintű és 17 metódus szintű metrikát tudunk kiszámolni.

A Mozilla elemzése

A Mozilla nyílt forráskódú böngésző és levelező rendszer volt az első ipari méretű szoftver, amelyet sikeresen elemeztünk. Az első kísérletek során 7 különböző verziót választottunk ki az 1.0-tól az 1.6-ig, amelyekre kiszámoltuk a metrika értékeket. Mind a 7 verzió több mint 1 millió nem üres és nem komment sort, több mint 4 500 osztályt, illetve majdnem 70 000 metódust tartalmazott.

A következő lépésünk a Mozilla osztályaiban megtalált és kijavított hibák számának meghatározása volt. Kidolgoztunk egy heurisztikát, hogy összegyűjtsük a szükséges bugokat a Mozilla hiba-követő rendszeréből, a Bugzillából [4], és az osztályokhoz rendeljük azokat. A Mozilla közösség biztosította számunkra a teljes Bugzilla adatbázist, amely tartalmazott minden hibát a szoftver fejlesztésének kezdete óta. A Bugzilla adatbázis 256 613 különböző hibabejegyzést tartalmazott, azonban nem volt mindre szükségünk. Ezért kiszűrtük azokat, amelyet más termékekre vonatkoztak (így 231 021 maradt), majd azokat, amelyek nem voltak kijavítva (57 553 maradt), továbbá azokat is, amelyek nem tartalmazták a javításhoz tartozó patch fájlt (22 553 maradt). Nem vettük figyelembe az 1.0-ás verzió előtt bejelentett hibákat sem (9 539 maradt), és hasonlóan az 1.7-es megjelenése utániakat is kiszűrtük. Összesen 8 936 különböző hibabejegyzés volt, amely teljesítette az összes feltételt.

Felhasználva a hiba bejelentésének és kijavításának az időpontját, a hibákat a megfelelő verziók

osztályaihoz tudtuk rendelni. Ehhez feltételeztük, hogy a bejelentés és a kijavítás időpontja között a hiba végig létezett. A patch fájlokat használtuk arra, hogy megtaláljuk, hogy az adott hiba mely osztályokat érintette. Ezzel a módszerrel a Mozilla esetében 3 961 hibát sikerült az osztályokhoz rendelni, amit az első kísérletünkben használtunk [7]. Az eredményt véletlenszerűen kiválasztott példákon kézzel ellenőriztük, amely alapján azt mondhatjuk, hogy a heurisztika helyesen működik.

A módszer legnagyobb hátránya, hogy a hibák összegyűjtéséhez szükségünk van a Bugzilla adatbázisra. Ezért továbbfejlesztettük a módszert, hogy a szükséges információkat az interneten keresztül közvetlenül a Bugzillából nyerjük ki. Később megismételtük a kísérletet a Mozilla 9 verziójával, de a hibákat már az új módszerrel gyűjtöttük össze. Az eredményeket az újabb vizsgálatainkban [8] elemeztük.

A szerző hozzájárulása az eredményekhez

Kifejlesztettük a Columbus keretrendszert, amelynek segítségével lehetővé vált az ipari méretű rendszerek elemzése. Kidolgoztunk egy nyelvfüggetlen modellt, amely alkalmas az objektum-orientált nyelvek magas szintű ábrázolására, illetve ezen modell segítségével képesek vagyunk kiszámolni az objektum-orientált metrikákat. A kialakított heurisztikánk segítségével összegyűjtöttük a Mozilla bejelentett és kijavított hibáit, majd az osztályokhoz rendeltük azokat.

A Columbus keretrendszer és a Mozilla elemzése nem a szerző munkája. A nyelvfüggetlen modell ötlete és kifejlesztése, valamint a C++ nyelvről történő konverzió kidolgozása és megvalósítása, illetve a nyelvfüggetlen modellen történő metrika számolás a szerző önálló munkája. Továbbá a hibákat a hiba-követő rendszerből összegyűjtő, illetve azokat az osztályokhoz rendelő heurisztika kidolgozása, illetve sikeres alkalmazása a Mozilla esetében szintén a szerző munkája.

2. Objektum-orientált metrikákon alapuló hiba-előrejelző modellek

Korábban már számos esetben igazolták a metrikák hiba-előrejelző képességeit, azonban alá akartuk támasztani ezeket az eredményeket egy nagy ipari rendszeren elvégzett kísérlettel. Ehhez nyolc metrikát választottunk ki, melyből 6 a Chidamber és Kemerer által definiált metrika [5], melyek az egyik leggyakrabban vizsgált és alkalmazott metrikák. A hetedik metrika az LCOMN, amely a 6 metrika egyikének módosítása. Ahhoz, hogy vizsgálhassuk az objektum-orientált metrikák és a hagyományos méret metrika közti különbséget, nyolcadik metrikának a nem üres és nem komment sorok száma metrikát választottuk. A metrikák definíciója a következő:

- A *Number of Methods Local (NML)* metrika értéke az osztály metódusainak a száma.
- A *Number Of Ancestors (NOA)* metrika az osztály ősosztályainak a számát méri.
- A *Number Of Children (NOC)* metrika értéke az osztály közvetlen leszármazottjainak a száma.
- A *Coupling Between Object classes (CBO)* metrika azon osztályok száma, amelyekhez az adott osztály kapcsolódik. Egy osztály kapcsolódik egy másikhoz, ha használja annak valamely metódusát vagy attribútumát, vagy közvetlenül öröklődik belőle.
- A *Response Set for a Class (RFC)* metrika azon halmaz számossága, amelybe az osztály metódusai és az azok által közvetlenül hívott metódusok tartoznak bele.

- *Lack of COhesion in Methods (LCOM)* azon metóduspárok száma, amelyek nem használnak közös attribútumot, mínusz azok száma, amelyek használnak. Ha a különbség negatív lenne, akkor a metrika értéke nullának van definiálva.
- *Lack of COhesion in Methods allowing Negative values (LCOMN)* metrikát ugyanúgy számoljuk, mint az LCOM-ot, csak negatív értékeket is megengedünk.
- *Logical Lines Of Code (LLOC)* az osztály nem üres és nem komment sorainak a száma.

Annak ellenére, hogy az összes vizsgált Mozilla verzióra kiszámoltuk a metrikákat, és meghatároztuk a hibaszámokat, csak az 1.6-os verziót használtuk a vizsgálatok során. A Mozilla forráskódja tartalmazott olyan osztályokat, amelyek a fordítás során generálódnak, így ezekhez az osztályokhoz nem lehetett hibákat rendelni. Ezért ezeket nem vettük figyelembe a vizsgálatok során. Emellett kiszűrtük azokat a hibamentes osztályokat is, amelynek léteztek mind a hét verzióban, és a metrikái nem változtak. Így a 3 192 osztály maradt, amelyek 3 961 bugot tartalmaztak.

Négy különböző módszert alkalmaztunk a metrikák vizsgálatára, és mindegyik eredményét figyelembe vettük, amikor a metrikákat értékeltük. A Mozilla esetében sok olyan osztály volt, amelyik több hibát tartalmazott, ezért *lineáris regressziót* [11] alkalmaztunk, hogy vizsgáljuk a metrikák és a hibák száma közti kapcsolatot. Először a metrikákat külön-külön vizsgáltuk, hogy lássuk, melyek alkalmasak a hibák előrejelzésére. Ezután megpróbáltuk növelni a modellek hatékonyságát azáltal, hogy egyszerre több metrikát alkalmaztunk. Az 1. táblázat tartalmazza az regresszió eredményeit. A NOC metrika esetében nincs korreláció a metrika és a hibák száma között, viszont a többi metrika alkalmas a hibák előrejelzésére (a p-értékek kisebbek, mint 0,01), bár különböző mértékben. A legjobb prediktor a CBO (ennek van a legnagyobb R^2 értéke), de az LLOC csak egy kicsivel gyengébb. Ahogy azt vártuk, több metrika együttes használatával sokkal jobb eredményt értünk el (az R^2 érték jelentősen nagyobb).

	NML	NOA	NOC	CBO	RFC	LCOM	LCOMN	LLOC	Többv.
p-érték	0.000	0.000	0.728	0.000	0.000	0.000	0.000	0.000	0.000
R^2	0.321	0.139	0.000	0.349	0.280	0.210	0.157	0.342	0.430

1. táblázat. A lineáris regresszió eredménye

A másik három módszerrel (*logisztikus regresszió* [9], *döntési fa* [13] és *neuron háló* [2]) az osztályok hibára való hajlamosságát vizsgáltuk, amihez két csoportra osztottuk azokat. Ez egyik csoportba a hibamentes osztályok tartoztak, a másikba azok, amelyek legalább egy hibát tartalmaztak. Mind a három módszer esetében először külön-külön vizsgáltuk a metrikákat, majd megnéztük, hogy több metrika együttes használatával jobb eredményt lehet-e elérni. Habár ezek a módszerek nem a hibaszámokat becsülték, ha megszámoljuk a hibásnak jelzett osztályokban található hibák számát, akkor megkapjuk, hogy hány hibát talált meg a modell.

Ahhoz, hogy össze tudjuk hasonlítani a különböző módszerek eredményeit, három jellemzőt számoltunk ki. A modell *helyessége* (correctness) megmutatja, hogy a modell az osztályok hány százalékát osztályozta helyesen. A *pontosság* (precision) azt, hogy a hibásan osztályozott osztályok hány százaléka hibás valójában. A *teljesség* (completeness) pedig azt, hogy a hibák (itt a konkrét hibákról van szó, és nem az osztályokról) hány százalékát találja meg a modell.

A modellek eredményeinek összefoglalása a 2. táblázatban látható. A *CBO (Coupling Between Object classes)* metrika volt a legjobb előrejelző a lineáris regresszió esetében, továbbá a helyesség,

Metrika	Modell	Helyesség	Pontosság	Teljesség
NML	Log. reg.	65,38%	68,84%	55,24%
	Dec. tree	66,51%	62,34%	65,33%
	Neural n.	66,20%	65,75%	60,19%
NOA	Log. reg.	64,04%	65,06%	45,17%
	Dec. tree	63,66%	63,13%	41,09%
	Neural n.	63,47%	61,36%	40,52%
NOC	Log. reg.	57,96%	–	–
	Dec. tree	57,95%	–	–
	Neural n.	57,96%	–	–
CBO	Log. reg.	69,77%	70,38%	69,12%
	Dec. tree	69,77%	69,13%	67,02%
	Neural n.	69,46%	70,63%	65,13%
RFC	Log. reg.	66,01%	71,89%	53,60%
	Dec. tree	66,45%	65,15%	56,91%
	Neural n.	66,76%	63,99%	61,66%
LCOM	Log. reg.	64,69%	81,34%	43,68%
	Dec. tree	66,67%	63,96%	60,59%
	Neural n.	66,17%	63,92%	60,36%
LCOMN	Log. reg.	63,82%	85,02%	39,01%
	Dec. tree	66,67%	63,96%	60,59%
	Neural n.	67,17%	66,70%	60,62%
LLOC	Log. reg.	66,85%	72,98%	54,58%
	Dec. tree	67,98%	66,81%	64,41%
	Neural n.	67,58%	65,29%	65,85%
Többv.	Log. reg.	69,61%	72,57%	65,24%
	Dec. tree	69,58%	68,38%	67,84%
	Neural n.	68,77%	68,94%	64,76%

2. táblázat. A helyesség, pontosság és teljesség értékek összefoglalása

pontosság és teljesség értékei majdnem minden esetben a legjobb a metrikák közül. Sőt, a helyesség értéke jobb, mint a többváltozós modellé, míg a pontosság és teljesség értékei is jobban voltak néhány esetben. Ezek alapján kijelenthetjük, hogy a CBO a legjobb hiba-előrejelző. Az *LLOC (Logical Lines Of Code)* metrikának szintén kiváló értékei voltak az összes analízisben, és csak a CBO volt jobb nála. Az *NML (Number of Methods Local)* és az *RFC (Response set For a Class)* metrikák rosszabb eredményt adtak, de még mindig alkalmasak a hibák előrejelzésére. Habár az *LCOM (Lack of COhesion in Methods)* és *LCOMN (Lack of Cohesion in Methods allowing Negative value)* metrikák eredményei eltértek egy kicsit, mindkettő gyengének mondható, azaz kevésbe alkalmasak a predikcióra. A *NOA (Number Of Ancestors)* esetében azt mondhatjuk, hogy találtunk némi kapcsolatot a metrika és a hibák között, azonban az igen gyenge, és az alkalmazhatósága további vizsgálatokat igényel. A *NOC (Number Of Children)* esetében mind a három modell hibamentesnek osztályozta az összes osztályt, azaz a NOC nem alkalmas a hibák előrejelzésére.

Összefoglalva az eredményeket azt mondhatjuk, hogy a 8 metrikából 7 alkalmas a hibák előrejelzésére, amelyek közül a CBO volt a legjobb, azaz jobban teljesített, mint az LLOC. Ahogy korábban

már említettük, sok hasonló vizsgálat történt már korábban. Így ennek a kísérletnek az igazi jelentőségét az adja, hogy az elsők között igazoltuk ezeket az összefüggéseket ipari rendszer esetében.

A Mozilla változásának vizsgálata

Megvizsgáltuk a Mozilla metrikáit, hogy lássuk, hogyan változtak a 7 elemzett verzió esetében. Az NML, CBO, RFC, LCOM és LLOC metrikák értéke jelentősen nőtt az 1.2-es verzió esetében. Ebben a verzióban a hibák száma szintén nőtt, ami érdekes, mert az osztályok száma meg csökkent.

Ezen megfigyelések alapján az feltételezzük, hogy egy nagyobb átalakítás történ a Mozilla forráskódján, amely jelentősen megnövelte a metrika értékeket és a hibák számát. Természetesen más tényezők is befolyásolhatták a bejelentett és kijavított hibák számának növekedését. Ennek kiderítésére további vizsgálatok szükségesek.

A szerző hozzájárulása az eredményekhez

Megvizsgáltuk a Chidamber és Kemerer által definiált metrikák, valamit hagyományos sorok száma metrika és az osztályokban található hibák száma közti kapcsolatot. Ehhez különböző statisztikai és gépi tanulási módszereket alkalmaztunk, amelyek közel azonos eredményt adtak. Ezek alapján azt mondhatjuk, hogy a 8 vizsgált metrikából 7 alkalmas a hibák előrejelzésére, de különböző mértékben.

A metrikák hiba-előrejelző képességeinek vizsgálata különböző statisztikai és gépi tanulási módszerekkel, és a hiba-előrejelző modellek kialakítása a szerző saját eredménye. A Mozilla változásának elemzését a szerző nem tekinti saját eredménynek.

3. A metrika-kategóriák hiba-előrejelző képességeinek vizsgálata

Korábban csak 8 metrika hiba-előrejelző képességeit vizsgáltuk, amelyek alapján nem lehet általános következtetéseket levonni. Ezért kiterjesztettük a kísérletünket a metrika-kategóriák vizsgálatára is. Ehhez először elemeztünk két további Mozilla verziót, így rendelkezésünkre állt minden verzió az 1.0-tól az 1.8-ig. Mivel sok hibát javítottak ki az első elemzések óta, ezért a hibákat újra összegyűjtöttük és az osztályokhoz rendeltük. Az új módszernek köszönhetően többé már nem volt szükségünk a Bugzilla adatbázisra, mert a szükséges információkat közvetlenül a Bugzillából gyűjtöttük össze. Megint az 1.6-os verziót választottuk ki, és az előzőekhez hasonló módon kiszűrtük a generált és változatlan osztályokat. Így 3 209 osztály marat, amelyekben összesen 7 662 hibajavítás történt.

A kísérlethez öt metrika-kategóriát (*méret*, *komplexitás*, *csatolás*, *öröklődés* és *kohézió*) választottunk ki, és a kategóriák hiba-előrejelző képességeink megítélése a kategóriába tartozó metrikák alapján történt. Az 58 kiválasztott metrika az öt kategória egyikébe lett sorolva.

Mivel korábban azt tapasztaltuk, hogy a négy alkalmazott módszer közel azonos eredményt adott, ezért ebben a kísérletben csak a logisztikus regressziót alkalmaztuk. 17 esetben találtunk kapcsolatot a metrika értékek és a hibák között. Ezen eredmények alapján azt mondhatjuk, hogy a csatolás metrikák a legjobb hiba-előrejelzők, amely alátámasztja a korábbi eredményünket, amelyben a CBO teljesített legjobban. A komplexitás szintén jó eredményt ért el. A méret-alapú metrikák közül a sorok számát különböző módon mérő, illetve a metódusokat valamint a publikus metódusokat mérő metrikák szintén

alkalmasak a hibák előrejelzésére. Sajnos a többi méret-alapú metrika nem használható. A kohéziós metrikák nagyon gyenge eredményt értek el, ezért nem ajánljuk azokat hiba-előrejelzésre. Továbbá egyetlen öröklődés metrika sem alkalmas a hibák prediktálására. Ezek az eredmények összhangban állnak a korábbi kísérlet eredményével.

A szerző hozzájárulása az eredményekhez

Kiterjesztettük a korábbi vizsgálatainkat a metrika-kategóriákra, és igazoltuk, hogy a csatolás metrikák a legjobb hiba-előrejelzők, de a komplexitás és bizonyos méret-alapú metrikák szintén alkalmasak a hiba predikcióra. A kohéziós metrikák eredményei elhanyagolhatóak, míg az öröklődés metrikák teljesen alkalmatlanok a prediktálásra.

A metrika-kategóriák hiba-előrejelző képességeinek vizsgálata teljes egészében a szerző saját munkája.

4. A szoftverfejlesztők véleményének felhasználása a minőségi modellek javítására

Bemutattuk, hogy a Columbus technológia segítségével képesek vagyunk tetszőleges nagy rendszert elemezni, és a metrikákat egyszerűen kiszámolhatjuk. Egy ipari rendszer esetében is igazoltuk, hogy van kapcsolat a metrikák és a szoftver minősége között, így azt gondolhatnánk, hogy minden rendelkezésünkre áll a metrikák gyakorlati alkalmazásához. Azonban mielőtt alkalmaznánk a metrikákat a szoftverfejlesztési folyamatban, meg kell vizsgálnunk, hogy a szoftverfejlesztők hogyan használnák azokat. A nem megfelelő használat csak lelassítaná a munkájukat ahelyett, hogy növelné a szoftver minőségét, vagy ami még rosszabb, nem várt mellékhatásai lehetnek.

Ezért készítettünk egy *kérdőívet* [16], aminek a segítségével vizsgáltuk a fejlesztők ismereteit és véleményét három objektum-orientált (LLOC, CBO és WMC) és egy kód duplikációhoz köthető (CI) metrikával kapcsolatban. A kérdőív több mint 50 kérdést tartalmazott. Azt vizsgáltuk, hogy a résztvevők szerint milyen kapcsolat van a metrikák és a program megértés, illetve tesztelés között. A kérdőív három nagyobb részre osztható. Az elején néhány általános kérdés segítségével felmértük a résztvevők tudását és tapasztalatát, hogy egy általános képet kapjunk róluk. Ezután a kérdések a résztvevők metrikákról kialakult véleményét vizsgálták egyesével, azaz minden esetben ugyanazt a kérdést tettük fel mind a négy metrikával. A kérdőív végén található kérdések a metrikákat együtt vizsgálták, és a résztvevőknek rangsorolniuk kellett azokat fontosságuk szerint.

Az 50 résztvevő, aki kitöltötte a kérdőívet a Szoftverfejlesztés Tanszéken különböző ipari és K+F projekteken dolgozott. A résztvevők tapasztalata és képessége széles skálán mozgott, mert a kezdő diáktól a tapasztalt programozóig mindenki megtalálható volt köztük. Ezért a válaszok és azok eloszlásának elemzése mellett statisztikai módszereket is használtunk annak vizsgálatára, hogy a tapasztalat hogyan befolyásolta a metrikák gyakorlati alkalmazhatóságának megítélését.

Először az általános kérdéseket és azok kapcsolatait néztük meg, amiben felfedezhettük a tanszék összetételének és munkájának jellegzetességeit. Továbbá ezen kérdések válaszai alapján a résztvevőket két csoportra (junior és senior) osztottuk azok adott területen szerzett tudása illetve tapasztalata alapján. Minden egyes további kérdés esetében vizsgáltuk, hogy volt-e szignifikáns különbség a két csoport válaszai között.

A második részen található kérdések azt vizsgálták, hogy hogyan ítélnék meg a résztvevők egy ismeretlen program megértésének vagy tesztelésének a nehézségét pusztán a metrikák felhasználásával. Arra is kíváncsiak voltunk, hogy milyen indokokat (például generált kód vagy egy jól ismert tervezési minta) tudnának elfogadni a „rossz" metrika értékekre. A harmadik kérdéscsoport azt vizsgálta, hogy a résztvevők hogyan osztanák szét a rendelkezésre álló erőforrásokat a metrika értékek felhasználásával. A kérdések szerepeltek mind a négy metrikával külön-külön. Számos esetben tapasztaltuk, hogy a junior és senior résztvevők véleménye szignifikánsan különbözött, ami azt jelenti, hogy a különböző területeken szerzett tapasztalat jelentősen befolyásolja a metrikák gyakorlati alkalmazását.

A kérdőív végén többek között arra voltunk kíváncsiak, hogy a tesztelés szempontjából mely metrikák a fontosak. A résztvevők szerint a WMC volt a legfontosabb, míg a másik három metrikát (LLOC, CBO és CI) közel azonosnak, de kevésbé fontosnak ítélték. A korábbi vizsgálatok eredményeit figyelembe véve ez az eredmény meglepő volt.

A kérdőívnek több érdekes eredménye is volt, amelyeket nem hagyhatunk figyelmen kívül a metrikák gyakorlati alkalmazásakor. Az egyik észrevételünk, hogy maguk a válaszok, illetve az a tény, hogy a tapasztalt programozók másképp ítélik meg a metrikák gyakorlati alkalmazhatóságát rávilágítottak arra, hogy a fejlesztőknek meg kell tanítanunk a metrikák helyes használatát.

Emellett sok esetben kiderült, hogy a metrikákat a speciális körülmények között másképp kell értelmezni. Például a résztvevők több mint a fele gondolta úgy, hogy a generált kód esetében a rossz metrika értékek elfogadhatóak. Ezen vélemények figyelembe vételével javíthatjuk a minőségi modelljeinket is.

A szerző hozzájárulása az eredményekhez

A metrikák gyakorlati alkalmazásának vizsgálatára készítettünk egy kérdőívet. Ennek segítségével vizsgáltuk, hogy a szoftverfejlesztők hogyan alkalmaznák a metrikákat a szoftverkarbantartás különböző területein, illetve, hogy számukra melyek a fontos metrikák. Igazoltuk, hogy a különböző területeken szerzett tapasztalat szignifikánsan befolyásolja a metrikák gyakorlati használatának megítélését. Ezek az eredmények rávilágítottak arra, hogy a fejlesztőknek meg kell tanítani a metrikák helyes használatát. Továbbá az eredmények felhasználhatóak a minőségi modelljeink javítására is.

A kérdőív ötlete és témája nem a szerző eredménye. A kérdőív részletes kidolgozása, végrehajtása és az eredmények kiértékelése a szerző saját eredménye.

Összefoglalás

Az értekezés az objektum-orientált metrikák vizsgálatával foglalkozik. Először bemutattuk a Columbus technológiát, amely alkalmas nagy C++, Java és C# rendszerek elemzésére és az objektum-orientált metrikák kiszámítására. Kidolgoztunk egy heurisztikát, melynek segítségével automatikusan összegyűjthetjük a bejelentett és kijavított hibákat egy rendszer hibakövető rendszeréből, majd az osztályokhoz rendelhetjük azokat.

Igazoltuk, hogy van kapcsolat a metrika értékek és a hibák száma között, azaz a metrikákon alapuló minőségi modellek javíthatják a szoftverek minőségét. Később kiterjesztettük a kísérletet a metrika kategóriák vizsgálatára is. Az eredményeket felhasználva olyan eszközöket fejlesztettünk ki, amelyek képesek mérni az adott rendszer minőségét, illetve követik annak fejlődését.

Egy kérdőív segítségével vizsgáltuk a metrikák gyakorlati alkalmazását, felhívva a figyelmet arra, hogy nem elegendő odaadni a fejlesztőknek a metrikákat, meg is kell tanítani őket a helyes használatukra. Emellett a fejlesztők véleményének figyelembe vételével a minőségi modelljeinket is javíthatjuk.

Végül, a 3. táblázat összefoglalja, hogy mely publikációk tartalmazzák az értekezés téziseit. Megjegyezzük, hogy a disszertációban ismertetett téma iránt nagy nemzetközi érdeklődés mutatkozik, amit az is jelez, hogy a Ferenc és munkatársai által publikált eredményekre [7] a dolgozat beadásáig 34 független hivatkozás, míg a Gyimóthy és munkatársai eredményeire [8] már több mint 100 független hivatkozás történt.

	[7]	[8]	[15]	[16]
I.	•	•		
II.		•		
III.			•	
IV.				•

3. táblázat. A tézisek és a velük kapcsolatos publikációk viszonya

Köszönetnyilvánítás

Először is szeretnék köszönetet mondani témavezetőmnek, Gyimóthy Tibornak, aki megismertetett ezzel az érdekes témával, és hosszú évek óta irányítja a kutatásaimat. Továbbá szeretnék külön köszönetet mondani Ferenc Rudolfnak, Fülöp Lajosnak, Jász Juditnak, Siket Péternek, Sógor Zoltánnak, valamint Szokody Fedornak a sok segítségért és támogatásért, amit a munkám során tőlük kaptam.

Hivatkozások

[1] Victor R. Basili, Lionel C. Briand, and Walcélio L. Melo. A Validation of Object-Oriented Design Metrics as Quality Indicators. In *IEEE Transactions on Software Engineering*, volume 22, pages 751–761, October 1996.

[2] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Clarendon Press, Oxford., 1995.

[3] Lionel C. Briand and Jürgen Wüst. Empirical Studies of Quality Models in Object-Oriented Systems. In *Advances in Computers*, volume 56, September 2002.

[4] Bugzilla for Mozilla.
`http://bugzilla.mozilla.org`.

[5] S.R. Chidamber and C.F. Kemerer. A Metrics Suite for Object-Oriented Design. In *IEEE Transactions on Software Engineering 20,6(1994)*, pages 476–493, 1994.

[6] Giovanni Denaro and Mauro Pezzè. An Empirical Evaluation of Fault-Proneness Models. In *ICSE 2002: Proceedings of the 24th International Conference on Software Engineering*, pages 241–251, March 2002.

[7] Rudolf Ferenc, István Siket, and Tibor Gyimóthy. Extracting Facts from Open Source Software. In *Proceedings of the 20th International Conference on Software Maintenance (ICSM 2004)*, pages 60–69. IEEE Computer Society, September 2004.

[8] Tibor Gyimóthy, Rudolf Ferenc, and István Siket. Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction. In *IEEE Transactions on Software Engineering*, volume 31, pages 897–910. IEEE Computer Society, October 2005.

[9] D. Hosmer and S. Lemeshow. *Applied Logistic Regression*. Wiley-Interscience, 1989.

[10] The Mozilla Homepage.
`http://www.mozilla.org`.

[11] J. Neter, W. Wasserman, and M. H. Kutner. *Applied Linear Statistical Models, 3rd Ed.* Richard D. Irwin, 1990.

[12] OpenOffice.org Home Page.
`http://www.openoffice.org/`.

[13] J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[14] Ferenc Rudolf. *Modelling and Reverse Engineering C++ Source Code*. PhD thesis, University of Szeged, 2004.

[15] István Siket. Evaluating the Effectiveness of Object-Oriented Metrics for Bug Prediction. *Periodica Polytechnica*, Budapest, 2009 Accepted for publication.

[16] István Siket and Tibor Gyimóthy. The Software Developers' View on Product Metrics; *A Survey-based Experiment*. *Annales Mathematicae et Informaticae*, Eger, 2010 Accepted for publication.

[17] Mei-Huei Tang, Ming-Hung Kao, and Mei-Hwa Chen. An Empirical Study on Object-Oriented Metrics. In *Proceedings of the 6th International Symposium on Software Metrics*, pages 242–249, November 1999.

[18] Ping Yu, Tarja Systä, and Hausi Müller. Predicting Fault-Proneness using OO Metrics: An Industrial Case Study. In *Sixth European Conference on Software Maintenance and Reengineering (CSMR 2002)*, pages 99–107, March 2002.