

TDK-dolgozat

Balogh Gergő

Szegedi Tudományegyetem
Természettudományi és Informatikai Kar
Informatikai Tanszékcsoport
Szoftverfejlesztés Tanszék

Balogh Gergő
MSc II. éves
prog. terv. inf. szakos
hallgató

Témavezető
Dr. Beszédes Árpád
egyetemi adjunktus

**Fejlesztői produktivitás megbízható
mérése és előrejelzése
szoftverfejlesztési projekteken**

Szeged

2012

Tartalomjegyzék

Tartalmi összefoglaló	6
1. Bevezetés	8
2. A probléma ismertetése	10
2.1. Projekt fejlődésének előre jelzése	10
2.2. A produktivitás	11
2.3. Produktivitás a szoftver fejlesztésben	12
2.3.1. A nyereség értelmezésének lehetőségei	12
2.3.2. Az erőforrás értelmezési lehetőségei	13
3. Alapkutatások	14
3.1. A használt keretrendszer	15
3.1.1. Az Eclipse plug-in ismertetése	15
3.1.2. A metrikák beszerzése	15
3.2. Termékmétrikák	15
3.3. Folyamatmetrikák	16
3.4. A használt produktivitás fogalom	17
3.5. Osztályozó	17
3.5.1. Tanuló halmaz	17

3.5.2.	Teljesítmény mérés	18
4.	Típusos módosítás használata	19
4.1.	A módosítások csoportosítása	19
4.1.1.	Művelet alapú csoportosítás	22
4.1.2.	Érintett elem alapú csoportosítás	23
4.2.	A módosítások súlyozása	23
4.3.	A módosítások meghatározása	24
4.3.1.	Statikus forráskód elemzés	24
4.3.2.	Revíziók közötti módosítások meghatározása	25
4.3.3.	Módosítás szabályok	27
4.4.	Módosítások összekapcsolása a metrikákkal	29
4.4.1.	Sorszám egyesítése a módosítás értékével	29
4.5.	Tanulóhalmaz létrehozása	30
4.6.	Célfüggvény értékek címkézése	31
4.7.	Osztályozó betanítása és értékelése	31
5.	A genetikus algoritmus	32
5.1.	Egyedek	32
5.1.1.	Az egyedek élet ciklusa	33
5.2.	Populáció	33
6.	Eredmények	35
6.1.	Mért projektek ismertetése	35
6.2.	A használt metrikák	36
6.3.	A TMOD, típusos módosítás kiszámítása	36
6.3.1.	A vizsgált módosítás típusok	36

6.3.2.	A csoportosítás során használt reguláris kifejezések	37
6.3.3.	A csoportok súlyainak meghatározása	37
6.4.	Eredmények ismertetése és elemzése	41
6.4.1.	Alapkutatás	42
6.4.2.	Kézi súlyválasztás	42
6.4.3.	Genetikus algoritmus által számolt súlyok	42
7.	Összefoglalás és további kutatási lehetőségek	44
7.1.	Kapott súlyok kiértékelése	44
7.1.1.	Nagyobb méretű tanulóhalmaz	44
7.1.2.	Szakértői vélemény bevonása	45
7.1.3.	Projektek közötti mérések	45
7.1.4.	Technológiák feladat specifikus produktivitásának becslése .	45
7.1.5.	Micro Productivity Profile bevezetése	45
7.2.	További csoportosítás létrehozása	46
7.2.1.	Érintett rendszer komponens alapú csoportosítás	46
7.3.	Projekt menedzser eszköz megvalósítása	46

Tartalmi összefoglaló

A szoftverfejlesztés területén egy projekt jövőbeli produktivitásának megbízható becslése nagy segítséget jelent a menedzsment számára a projektre fordítandó erőforrások elosztása szempontjából.

E metrikát általában a befektetett erőforrás és a megtermelt nyereség arányaként definiálják. A programozás során erőforrásnak tekinthetjük a fejlesztésre fordított időt. A nyereség egyik lehetséges értelmezése a keletkezett vagy módosult kódsor mennyisége. E területen az egyetem már folytatott kutatásokat. A kutatás során a jövőbeli produktivitást három osztályba sorolták. A fejlesztésre szánt időt pedig az Eclipse fejlesztő környezetbe épített bővítmény szolgáltatta, ezenkívül felhasználtak termék és folyamat metrikákat. A becslés meghatározása gépi tanulási algoritmusokkal történt, melyeket a Weka nevű könyvtár biztosított. Több Java nyelvű projekt mérése után, az így kapott tanuló halmaz alapján betanított osztályozó megközelítőleg 60%-os teljesítményt nyújtott. Figyelembe véve a véletlenszerű választás esélyét – mely közelítőleg 33% — ez biztató, de nem elégséges eredmény.

Ebben a fázisban csatlakoztam a kutatáshoz, melynek fő célja a kapott eredmények javítása volt. Több lehetőség közül az osztályozás során használt célfüggvény, vagyis a produktivás meghatározásának pontosítását választottam. A módosítások egységes kezelése helyett, azokat csoportosítottam, és e csoportokat egy egyszerű evolúciós algoritmus segítségével becsült súlyokkal láttam el. Két új metrikát hoz-

Fejlesztői produktivitás megbízható mérése és előrejelzése szoftverfejlesztési projekteken

tam létre a produktivitás mérésére e módszer segítségével: a módosítás típusára és az érintett rendszer komponensre épült. Mind kettő alapját egy-egy létező csoportosítás adja, melyet átalakítottam a rendelkezésre álló adatok figyelembe vételével. A saját metrikáimat használva az osztályozó közelítőleg 80% teljesítményt ért el, mely a mérés során használt súlyok pontosításával valószínűleg tovább javítható. Ezért a későbbiekben szeretném összevetni az algoritmus által meghatározott súlyokat a fejlesztők által javasoltakkal.

Kulcsszavak

- produktivitás mérés
- produktivitás becslés
- genetikus algoritmus
- előrejelzés
- szoftverfejlesztés
- projektmenedzsment

1. fejezet

Bevezetés

A szoftverfejlesztők már a 1960-as évek óta próbálnak becsléseket tenni a fejlesztésre szánt erőforrások jövőbeli mértékére. A kutatások többsége formális modellek létrehozására irányult. Ez kezdetekben más szakterületek modelljeinek átemelését jelentette. Azóta több különböző modell létrehozási megközelítést tanulmányoztak a témában. Ezek közt voltak az *osztályozás*, *neurális hálózat*, *fuzzy logika*, ill. *szimuláció alapú modellek*, vagy ezek kombinációi.

A napjainkban elterjedt formális modell a COCOMO, mely az 1970-es években kezdte meg a pályafutását. A modell a *regresszióanalízisre* épül. Felhasználja a projekt történetéből származó adatokat és a jelenlegi tulajdonságait. Az alapmodell a fejlesztési költségeket a program méretének függvényeként értelmezi. Méretmeghatározáshoz pedig a SLOC (*forráskód sorok számát*) használja.

A *funkció pont* alapú modellek szintén az 1970-es években születtek. A rendszer nyújtotta szolgáltatások mennyiségének meghatározásával becslést tesz a fejlesztési költségekre. A funkciókat a rendszer történetéből gyűjti be. A szubjektív értékelés mellett létezik mérnöki változata is, mely az egyes rendszer elemek számosságán alapszik.

Ezekon kívül több más modell is született, azonban a költségbecslés legelterjedtebb módszere továbbra is a *szakértői vélemény* alapú becslés.[3] A módszer nyilvánvaló hiányossága a szubjektív véleményalkotás, mely nehezen automatizálható. Ezenkívül problémát okozhat a megfelelő szaktudás hiánya is.

A produktivitás, vagyis befektetett erőforrás és a megtermelt nyereség aránya nagy segítséget nyújthat a projekt menedzsment számára. Meghatározása során bármelyik fenti módszer használható. Számításának automatizálása lényeges szempont a nagyobb projektek esetében. Ezért fontos lehet az egyes metrikák használatának kutatása a becslés során. A *termék és a folyamat metrikákon* alapuló becsléssel kapcsolatban az egyetem már folytatott kutatásokat. [9, 10]

A produktivitás becslés során fontos megérteni a változások és a rendszer fejlődése közti kapcsolatot. A módosítások és a rendszerre gyakorolt hatásuk vizsgálatával a *hatásanalízis* szakterülete foglalkozik. A hatásanalízis során központi szerepet betöltő módosítások osztályozása és megértése e területen is fontos szerepet kap, ezeket különböző hatás analízis modellek segítségével írják le és vizsgálják. Egy ilyen modellt ismertet a [1] cikk.

Jelen kutatás célja a produktivitás előrejelezésének pontosítása. Az előrejelzéshez jelenleg termék és folyamat metrikák halmazát használjuk, melyek alapján gépi tanulási módszerek segítségével becslést adunk a projektek produktivitásának jövőbeli mértékére.

Az eddigi eredmények biztatóak. Végző cél egy olyan eszköz kifejlesztése, ami valós környezetben is használható produktivitás becslésekkel segíti a projekt menedzserek munkáját. Ehhez a meglévő eredmények pontosítása szükséges, melyet a *produktivitás mérésére szolgáló metrika* újradefiniálásával szeretnék elérni.

2. fejezet

A probléma ismertetése

A projekt különböző tulajdonságainak előrejelzése nagy fontossággal bír a szoftverfejlesztés területén. A megbízható előrejelzés főleg a projekt menedzsment számára fontos, mivel ezek a becslések segítséget nyújthatnak az erőforrások hatékonyabb szétosztásában.

2.1. Projekt fejlődésének előre jelzése

A Mockus és társai [6] által ismertetett költség becslő modell lehetővé tette a karbantartási költségek mennyiségének és időbeli eloszlásának előjelzését. A modell felhasználta a fejlesztő azonosítóját, a módosítás típusát, méretét, a méretének nagyságrendjét és a bonyolultságát. Az alapkutatás szintén felhasználja a fejlesztő azonosítóját az előrejelzés során. A saját mérésen ezen túl figyelembe veszi a Mockus és társai által javasolt módosítással kapcsolatos egyéb jellemzőket is.

A termék metrikákat gyakran használják a költségek előre jelzéséhez. Egy jól ismert termék metrika halmazt határoz meg Chidamber és Kemerer [2]. Sok empirikus kutatás erősítette meg e halmaz használhatóságát. Kimutatták, hogy korreláció

Fejlesztői produktivitás megbízható mérése és előrejelzése szoftverfejlesztési projekteken figyelhető meg a metrikák értékei és a karbantartási költségek között [4].

A termék metrikák mellett, fejlesztők tapasztalatainak és más folyamat metrikák használatát már kutatták a hiba előrejelzés területén. Mockus és Weiss [5] egy olyan modellt használtak, mely figyelembe veszi a módosítás méretét és szórását, típusát és a fejlesztők szaktudását a rendszerrel kapcsolatban. Úgy találták, hogy a módosítások szórása és a fejlesztők szaktudása jelentős a hiba előrejelzés során. Moser és társai [7] rámutattak, hogy a legpontosabb hiba előrejelző tulajdonságok, a revíziók és a hibajavítások nagy száma, valamint a kis méretű komitok és a kevés újratervezések. Yuan és társai [11] fuzzy szubtraktív klaszterezést használt a hibák számának előrejelzéséhez, termék és folyamat metrikákon alapuló modellt hozott létre. A többi kutatóval szemben azt állította, hogy a folyamat metrikák nem javítják az előrejelzés pontosságát és nem nyújtanak elfogadható eredményeket.

2.2. A produktivitás

Az általános és rövid definíció szerint a produktivitás a termelés hatékonyságát méri. Számszerűsítve ez a nyereség, vagyis a termék és a nyújtott szolgáltatás mennyiségének ill. az előállítás során használt erőforrások mennyiségének a hányadosa.

$$\text{produktivitás} = \frac{\text{nyereség}}{\text{erőforrás}}$$

A definíció túl általános a közvetlen használathoz. Ezért az egyes szakterületek számára pontosítani kell a *nyereség* és az *erőforrás* fogalmát. Ezeknek jól meghatározható, objektív, lehetőleg számszerűsíthető mértékeknek kell lenniük. Mind három tulajdonság a mérések későbbi automatizálhatósága szempontjából fontos. Az így kapott pontosabb definíció adja az adott szakterület produktivitás modelljét.

2.3. Produktivitás a szoftver fejlesztésben

A szoftverfejlesztés területén a termék szerepét az elkészített program tölti be. Míg szolgáltatásnak tekinthetjük például a karbantartási és rendszer-üzemeltetési feladatokat. Jelen dolgozatban csak az előbbi megközelítéssel foglalkozom, vagyis csak az előállított szoftvert tekintem terméknek.

2.3.1. A nyereség értelmezésének lehetőségei

Ahhoz hogy a fejlesztés során meg tudjuk határozni az adott idő intervallum által létrehozott mennyiséget, szükségünk van a szoftver állapotának jól meghatározott és visszakereshető, múltbeli pillanatképeire. Ezen pillanatképek szerepét betölthetik az egyes verziók során kiadott bináris csomagok, az adott rendszer specifikációja vagy a program forráskódja. Ez utóbbi választása mellett szól a verziókövető rendszerek nagymértékű elterjedése. Célszerű a revíziószámot használni, mint időbeli egységet.

A forráskód számos tulajdonságát vagy ezek tetszőleges kombinációit választhatjuk mértéknek. Például az forráskódsorok számát, vagy az egyes elemek számosságát, mint az osztályok, metódusok és adattagok.

A nyereség kiszámítása során elégséges a változásokat tekinteni. Ehhez két állapot összehasonlítása szükséges. A változások meghatározásánál használhatunk egyszerű mennyiségbeli eltérést. Az összehasonlítás alapja lehet az első, a megelőző vagy bármely e kettő közötti mérföldkőnek tekintett állapot.

Lehetőségünk van a már létező tulajdonságok csoportosítására is. Például elkülöníthetjük az változott sorokat a szerint, hogy megjegyzést, műveletet vagy a kettőt elválasztó térközöket tartalmaznak.

2.3.2. Az erőforrás értelmezési lehetőségei

Az erőforrás lehetőségek száma sokkal korlátozottabb. Például nem lehetséges a más területeken megszokott fizikai alapanyag mértékének használata, mivel a programozás során általában pusztán szellemi tevékenységről beszélünk. Szintén nem, vagy csak nehézkesen használható a pénzbeli ráfordítás mértéke, mivel ez nagyban függ a felhasznált technológiáktól. Ezen technológiák fejlesztőnként is eltérhetnek: gondoljunk a használt operációs rendszerre vagy a fejlesztői környezetre.

Az előállítás során használt erőforrások közül, a szoftverfejlesztés során a fejlesztésre szánt tényleges idő mennyisége logikus választás lenne, azonban ennek mérése igen nehézkes, így kevesen használják. De például használhatjuk a termelés előző fázisa során előállított nyereség méretét, mint például a rendszertervben leírt funkcionalitások számát.

3. fejezet

Alap kutatások

Jelen fejezetben a egyetem által a témában folytatott kutatásokat mutatom be. Az itt közölt eredmények nem saját méréseken és kutatásokon alapulnak. Azonban fontos szerepet töltenek be a saját kutatásaim alapjaként, ezért a később tárgyalt részek megértéséhez elengedhetetlen az átfogó ismeretük. A teljes leírás az [9, 10] cikkekben található.

A kísérlet témája a produktivitás előrejelzés. Az előrejelzés során vizsgálták a különböző metrikákat, mint prediktorokat, melyek segítségével pontosítani lehet a produktivitás előrejelzését. Ezzel segítséget nyújtva a menedzsment számára az erőforrások kezelésében.

A legtöbb, eddig a témában megjelent munkával ellentétben, az alap kutatás nem a különböző metrika halmazok hatékonyságát vizsgálja a produktivitás előrejelzés során. Inkább a különböző típusú metrikák összehasonlító elemzésére helyezi a hangsúlyt.

3.1. A használt keretrendszer

A mérés környezete több egyedből épült fel. Ezek közt megtalálhatók az egyes fejlesztők munkaállomásai, a különböző egyetemi szerverek, a Weka rendszer és a projekt menedzser.

3.1.1. Az Eclipse plug-in ismertetése

Minden fejlesztői munkaállomáson telepítésre került a fejlesztői környezetbe épülő *Productivity Plug-in* ami alacsony szintű információkat gyűjt a fejlesztéssel kapcsolatban, mint például az aktív fájlok, Eclipse perspektívák, fejlesztő azonosítója, fejlesztési feladat neve, eltelt idő. A naplózott adatok feltöltésre kerülnek a *Productivity Log Server-re*.

3.1.2. A metrikák beszerzése

A projekt menedzser megbecsül néhány folyamat metrikát, mint a feladat elvégzéséhez szükséges időt, a fejlesztők szaktudását és meghatározza a fejlesztési feladatokat. A folyamat metrikák a *Productivity Log Server* által tárolt adatokból származnak. A termékmetrikákat pedig a *Columbus tool* biztosítja, ami a verziókövető rendszerből származó forráskódokat elemzi.

Az összegyűjtött adatokat a *Weka* gépi tanulást végző eszköz használja fel a produktivitás meghatározásához.

3.2. Termékmetrikák

- LLOC (*Logical Lines Of Code*) nem üres és nem megjegyzés sorok összege

- C&K metrikák
 - DIT (*Depth of Inheritance Tree*)
 - NOC (*Number Of Children*)
 - CBO (*Coupling Between Object Classes*)
 - RFC (*Response For a Class*)
 - WMC (*Weighted Methods per Class*)
 - LCOM (*Lack of COhesion in Methods*)
- ECLOC (*Effectively Changed Lines Of Code*) ténylegesen változott sorok száma. Azon sorok melyeket módosítottak, töröltek, vagy létrehoztak a két érintett verzió között.

3.3. Folyamatmetrikák

- TT (*Task Time*) a feladat elvégzésére szánt idő becslése három kategóriába sorolva: alacsony, közepes, magas
- DEP (*Developer's Experience in Programming*) a fejlesztő szaktudásának becslése három kategóriába sorolva: alacsony, közepes, magas
- NFA (*Number of File Access*) megmutatja hogy a fejlesztő, a módosítás során hányszor tért vissza az adott fájlhoz
- NDF (*Number of Developers of File*) megadja a fájl módosító fejlesztők számát
- DT (*Development Time*) a fájl módosítására fordított teljes idő.

3.4. A használt produktivitás fogalom

Az előző fejezetben ismertetett produktivitás fogalmak közül a módosított sorok számának hányadosát és a fejlesztésre szánt idő hányadosát használták.

Ez a fent ismertetett metrikák felhasználásával az alábbi képlettel adható meg.

$$\text{produktivitás} = \text{LMC} = \frac{\text{ECLOC}}{\text{DT}}$$

3.5. Osztályozó

Az előrejelzés során három gépi tanulási algoritmus teljesítményét vizsgálták: döntési fák, neurális hálózatok és regresszió.

3.5.1. Tanuló halmaz

A tanuló és teszhalmazok bemenetét több üzleti szféra számára kifejlesztett projekt adatai és forráskódja adta. Ezek mindegyike *Java nyelven* íródott. A forráskód változásait az SVN verzió követő rendszer segítségével kezelték.

A használt metrikák több különböző forrásból származnak. A C&K és az LLOC metrikákat a *Columbus tool* biztosította. Az ECLOC metrikát az SVN szervertől begyűjtött adatokból számították. A folyamat metrikák többségét pedig a bővítéstől származó adatokból nyerték.

Ezt követően a tanulóhalmaz a következő három csoport kombinációjaként állt össze.

- revízió szám
- C&K, LLOC és ECLOC metrikák

- folyamat metrikák: TT, DEP, NFA, NDF és DT

A célfüggvény az LMC metrika három kategóriába sorolt értékei adták.

3.5.2. Teljesítmény mérés

A fent leírt tanuló halmazban betanított osztályozók teljesítményét keresztvalidációs eljárással mérték. A pontosság kiértékelése során a precision, recall és az F-measure értékeket használták.

A precision és recall mértékeket az elfogadott-igaz ($t_{\oplus}$), elutasított-igaz ($t_{\ominus}$), elfogadott-hamis ($f_{\oplus}$) és az elutasított-hamis ($f_{\ominus}$)¹ esetek számával definiálják.

$$\text{precision} = \frac{t_{\oplus}}{t_{\oplus} + f_{\oplus}}$$
$$\text{recall} = \frac{t_{\oplus}}{t_{\oplus} + f_{\ominus}}$$

Ezen két mérték harmonikus közepeként az F-measure a következőképpen adható meg.

$$\text{F-measure} = 2 \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Mind az osztályozó betanítását és a keresztvalidációt a *Weka* eszköz segítségével végezték el.

¹true-positive, false-positive, true-negative, false-negative

4. fejezet

Típusos módosítás használata

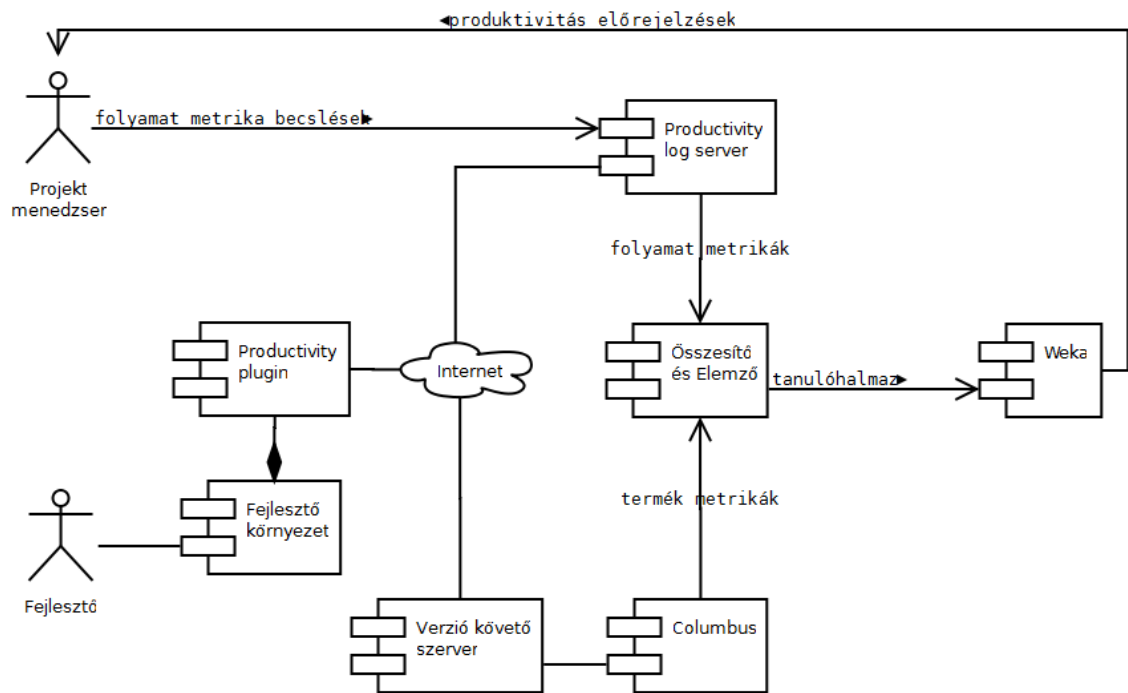
A kutatásba az előző fejezetben leírt eredmények elérése után kapcsolódtam be. Céloom a kezdeti biztató eredmények pontosítása volt.

Több lehetőség közül a célfüggvénykén módosítását választottam. Feladatokat a produktivitás használt fogalmának újragondolásával kezdtem. Ennek során pontosítottam a nyereség mérésére szolgáló metrikát. A metrika kiszámításához saját programot készítettem, mely elemzi a projektek egyes állapotairól készült pillanatképeket.

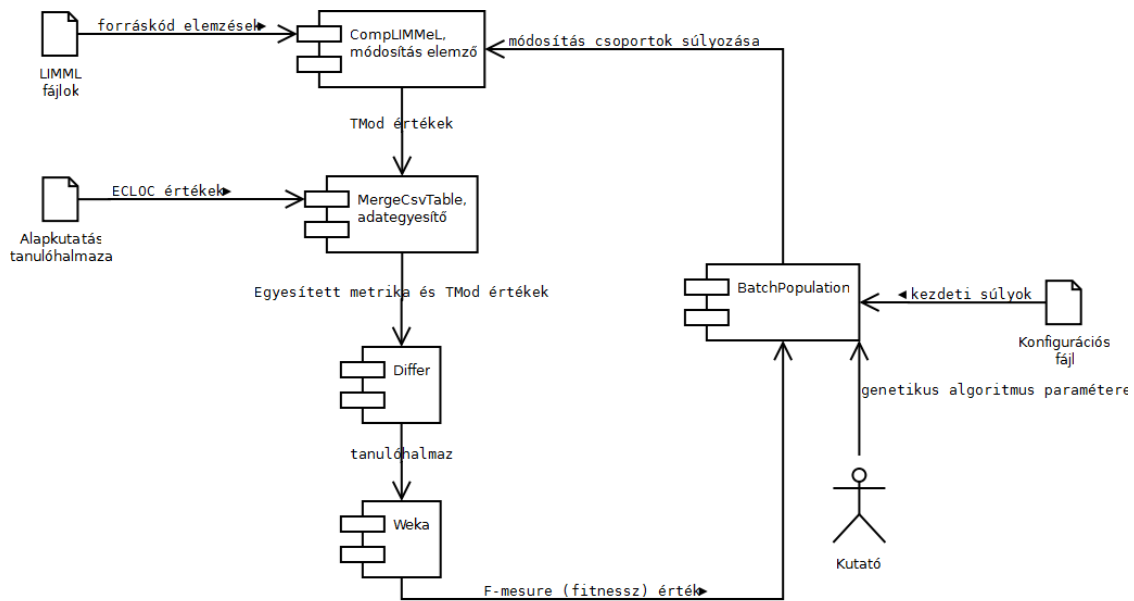
A 4.1 ábra mutatja a kutatás során használt keretrendszer elemeit és az azok közötti főbb adatáramlást. Az általam írt szoftver az összesítő és elemző komponens része. A 4.2 ábra ezt a komponenst mutatja be, az egyes elemek ismertetése a fejezet további részeiben található.

4.1. A módosítások csoportosítása

A korábban részletezett ECLOC metrikából indultam ki. Ezt a korábbi kutatások során a produktivitás fogalom szempontjából nyereségként értelmezték. Alapfelte-



4.1. ábra. A méréshez használt keretrendszer



4.2. ábra. Az összesítő és elemző komponens áttekintése

vésem, hogy a változtatás során előteremtett érték nem egyenletesen oszlik meg a változtatott sorok között. Vagyis egyes sorok nagyobb jelentőséggel bírnak.

Ezt jól szemlélteti az alábbi két példa. Az első esetben a változó átnevezése sokkal kisebb fejlesztői erőfeszítést igényel mint a második példában látható típus változtatás. Azonban az eddig használt metrika szempontjából mindkettő azonos változásnak tűnik.

Példa

`int a;` $\Rightarrow$ `int b;`

Példa

`int a;` $\Rightarrow$ `double a;`

Ez a megközelítés nyilván jobb becslését adja a programozói munka során előállított szellemi terméknek. Megfigyelhető azonban, hogy egyes változtatások nem fejezhető ki csak az érintett sor megfigyelésével és létezik olyan sormódosítás, mely nem jelent számottevő változást a rendszer szempontjából. Ezért teljesen elhagytam a kódsorok vizsgálatát és rendszer egészéről készült elemzéseket hasonlítottam össze.

Az így kapott módosítás halmaz elemeit különböző tulajdonságaik alapján csoportokba soroltam. A csoportosításra két fő szempont alapján került sor. Mivel minden eddigi mérés objektum orientált paradigmát használó nyelveken írt rendszereken történt, ezért a két szempontot e területen keresztül mutatom be.

Rendszer elemnek tekintem az következő elemeket az objektum orientált környezetben.

- névterek
- osztályok
- objektumok
- osztály adattagok
- osztály metódusok

A csoportosítás az [1] cikkben ismertetett hatás analízis modell alapján dolgoztam ki.

4.1.1. Művelet alapú csoportosítás

Művelet alatt bármelyik rendszerszintű komponensen elvégzett változtatást értek. Jelenleg a következőket észleltem és különítettem el az összehasonlítás során.

Fejlesztői produktivitás megbízható mérése és előrejelzése szoftverfejlesztési projekteken

elem létrehozása Például ha egy új osztályt adunk hozzá a rendszerhez.

elem törlése Például ha egy osztályban szereplő metódust törlünk.

elem láthatóságának módosítása Például egy adattag publikussá tétele.

elem típusának módosítása Például egy adattag típusának megváltoztatása egész számról lebegő pontos számra. Mivel minden adattag felfogható egy lekérdező (getter) és egy beállító (setter) metódusként, ezért ide sorolom a metódusok paraméterlistájának és visszatérési értékének módosítását is.

elem méretének megváltoztatása Minden elem esetén meghatározható az elemet leíró forráskód. E kódrészlet mennyiségében történő változás tartozik ide. Ez nem fedi le a kódrészlet tartalmát érintő változásokat.

4.1.2. Érintett elem alapú csoportosítás

Az érintett elem az, melyre a változtatás művelete irányul. Jelenleg a következő elemek szerint csoportosítom a változtatásokat.

- Osztály
- Metódus
- Adattag

4.2. A módosítások súlyozása

A fent definiált módosítás osztályokra elmondható – hasonlóan a módosított sorokhoz – hogy nem minden módosítás elvégzéséhez szükséges azonos fejlesztői erőfeszítés, és így az egyes módosítások kisebb arányban járulnak hozzá az előállított

Fejlesztői produktivitás megbízható mérése és előrejelzése szoftverfejlesztési projektekben szellemi termék értékéhez. E különbség kifejezése érdekében célszerű a módosításokhoz súlyokat rendelni. A hozzárendelést csoportok segítségével valósítottam meg.

A későbbiekben leírt csoportosításokat egy általam írt program végzi el. Ez több jól elkülönülő részből áll, melyek programkönyvtárakba szerveztem. A fő belépési pontot tartalmazó modul a `BatchPopulation(lib).NET` nevet kapta. A mérés ennek futtatásával indítható.

4.3. A módosítások meghatározása

A módosítások meghatározását a `CompLIMel(lib).NET` nevű könyvtár végzi. Ez hasonlítja össze az egymás után következő SVN revíziók forráskódja alapján készült elemzéseket. A teljes elemzés során kizárólag a statikus forráskódból indulok ki.

4.3.1. Statikus forráskód elemzés

A forráskód elemzését egy korábbi projekt keretén belül megvalósított eszköz végzi. Az egyes állapotok leírásához az egyetemen kifejlesztett sémát használom. A séma egységes módon tárolja a forráskód elemzés során felderített nyelvi elemeket és rendszer komponenseket, valamint ezek közti kapcsolatokat. Előnye, hogy így nem szükséges a forráskód újrafeldolgozása a mérés lépései során. Ezenkívül információkat szolgáltat a teljes rendszerre vonatkozó esetleg forrásfájlokon átívelő összefüggéseiről, melynek felderítését leveszi a mérést végző szoftver válláról.

A felhasznált séma szerkezete

A séma belső szerkezetének ismertetése túlmutat ezen dolgozat keretein, így csak röviden ismertetem a főbb jellemzői. A séma gráf reprezentációként tárolja a rend-

Fejlesztői produktivitás megbízható mérése és előrejelzése szoftverfejlesztési projekteken
szer felépítését. Az egyes csomópontok a rendszer elemeit, míg a köztük lévő élek az elemek közti kapcsolatokat jelképezik. A rendszer különböző szintjein található elemek (pl.: osztályok, adattagok, utasítások) szintén csomópontok segítségével kerülnek tárolásra. A pontok közötti öröklődés biztosítja a hasonló tulajdonságú elemek kezelését. A séma lehetőséget biztosít a típus és objektum hierarchia ill. a vezérlési szerkezetek tárolására is. A kapcsolatok kifejezhetnek például tartalmazást, öröklődést és az asszociációk különböző fajtáit is.

LIMML fájlok

A kutatás során a sémának csak egy szűk részhalmazát használtam fel, az ún. LIM (*nyelv független modell*) részét. Ez független az elemzett program forrásnyelvétől. A modell lehetővé teszi az objektum orientált rendszerek típus hierarchiájának feltérképezését. A definiált osztályok és azok belső szerkezete is elérhető. Az egyes elemek revíziókénti és rendszer szintű egyedi azonosítóval is rendelkeznek, így a változások detektálása könnyebben elvégezhető.

A modell tárolására több lehetőség is adott. Én az XML formátum feldogozását választottam, a könnyű kezelhetősége és a széles körben elterjedt támogatottsága miatt. XML formátum esetén az elemzés során egyetlen, szintaktikailag helyes XML fájl keletkezik LIMML kiterjesztéssel. Miután kutatás során vizsgált projektek minden egyes revíziójához elkészült a LIMML fájl, a forráskód nem került többé felhasználásra.

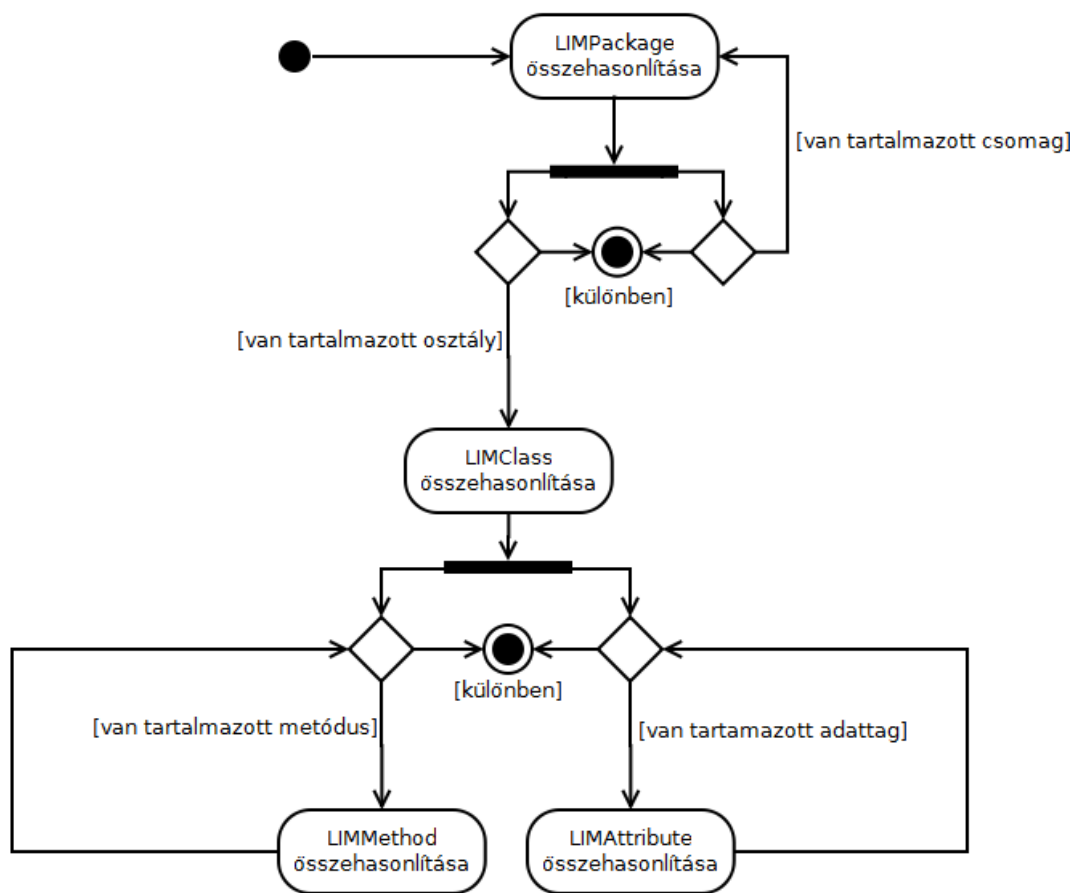
4.3.2. Revíziók közötti módosítások meghatározása

Két egymást követő revízió közötti különbségek meghatározásához, első lépésben beolvasom mindkét revíziót leíró LIMML fájlt. A mérés során elemzett programok objektum orientált szemlélete miatt ezek egy-egy LIMPackag^(obj)_{.NET}-ként kerülnek

Fejlesztői produktivitás megbízható mérése és előrejelzése szoftverfejlesztési projektekben reprezentálásra. Ezek összehasonlíthatóságát az $IChangeable_{NET}^{(interface)}$ biztosítja.

Minden módosítást az ún. $IDelta_{NET}^{(interface)}$ leszármazottjaként tárolok. A módosítás csoportokat pedig az $AggregatedDelta_{NET}^{(class)}$ fogja össze.

Minden összehasonlítás során egy $AggregatedDelta_{NET}^{(class)}$ jön létre. Ez rekurzív algoritmussal kerül feltöltésre. Az algoritmus végighalad az elem tartalmazási kapcsolatain a $LIMPackage_{NET}^{(class)}$ -től a $LIMClass_{NET}^{(class)}$ -on keresztül egészen a $LIMMethod_{NET}^{(class)}$ és $LIMAttribute_{NET}^{(class)}$ elemekig. A folyamatot a 4.3.2 ábra mutatja be.



4.3. ábra. A LIM elemek összehasonlítása

A módosításokat két interfész fogja össze: a már említett $IDelta_{NET}^{(interface)}$ és az $IChange_{NET}^{(interface)}$. Előbbi tetszőleges változást ír le, ide tartozik a létrehozás és a törlés. Utóbbi az érintett elem valamelyik tulajdonságának módosítását jelképezi.

4.3.3. Módosítás szabályok

A módosítások (delták) csoportosítását módosítás szabályok segítségével végzem. A szabályok írják azokat a közös jellemzőket, melyek alapján a delta adott csoportba kerül. Vagyis a szabályok ekvivalencia relációt definiálnak a delták halmaza fölött. Mindegyik szabály tartalmazza az általa definiált delta csoporthoz rendelt súlyt.

A szabályok két csoportra oszthatók attól függően hogy melyik delta interfész által leírt tulajdonságokra tartalmazznak megszorításokat. A $IDelta_{NET}^{(interface)}$ -hoz a $WeightRule_{NET}^{(class)}$, a $IChange_{NET}^{(interface)}$ -hez a $ChangeRule_{NET}^{(class)}$ szabály tartozik.

A $WeightRule_{NET}^{(interface)}$ a következő tulajdonságokat képesek előírni.

- módosítás típusa
- érintett elem típusa
- érintett elem láthatósága

Ezeket kibővítve a $ChangeRule_{NET}^{(class)}$ a következők korlátozására is alkalmas.

- megváltozott tulajdonság típusa
- megváltozott tulajdonság változás előtti állapota
- megváltozott tulajdonság változás utáni állapota

Szabályok illesztése

A szabályok az egyes megszorításokat reguláris kifejezések segítségével írják le, így megfelelő szabadságot adva a későbbi metrika finomhangolásához vagy új metrika bevezetéséhez. A reguláris kifejezések a .NET által leírt szabályokat követik, melyekre itt nem térek ki részletesen.

Az érintett elem láthatósága és típusa egyazon kifejezés illesztésével történik. Ehhez az érintett elem láthatóságát és típusát a `[.]` (pont) elválasztásával fűzöm össze.

A példában látható osztály a következő leírot adja az összefűzés után:

```
public.LIMClass.
```

Példa

```
public A{}
```

A módosítások illesztése során teljes illesztést használok. *Egy szabály akkor és csakis akkor illeszkedik egy deltához, ha a delta minden tulajdonsága illeszkedik a deltában szereplő az adott tulajdonságot leíró reguláris kifejezésre.*

Módosítások összegzése

A módosításokat osztályok szerint szétválogatom. Minden olyan módosítást is az adott osztály alá sorolok ami annak elemeit érinti. Ezt követően a szabályok illesztése segítségével meghatározom az adott deltához tartozó súlyt. Végül ezen súlyok összegét rendelem hozzá az adott osztály adott revízióban szereplő állapothoz, mint változás mértékét. A folyamat a következő pszeudokóddal írható le.

minden r_i és r_{i+1} revíziókra

minden c_j r_{i+1} -ben létező osztályra

legyen $w_{c_j, r_{i+1}} = 0$ a c_j -hez és r_{i+1} -hez tartozó változás mértéke

minden d_k módosítására, mely a c_j osztályhoz tartozik

minden p_m szabályra

ha p_m illeszkedik d_k -hoz akkor

növeljük meg $w_{c_j, r_{i+1}}$ értékét a p_m -hez rendelt súllyal

különben

ne tegyünk semmit

4.4. Módosítások összekapcsolása a metrikákkal

Következő lépésben a 3.2 és a 3.3 részben ismertetett metrikákat összekapcsolom a kapott súlyokkal. Ezt két tábla egyesítésével végzem. A folyamat és termékmetrikákat az alapkutatás részeredményeit tartalmazó tábla, míg a típusos módosítást az általam, a fenti módon leírt tábla tartalmazza. A táblázat sorait a revíziószám és az osztály nevének egyezése esetén tekintem azonosnak.

Az egyesítést a saját `MergeCsvTable(lib).NET` végzi el. Ez tetszőleges CSV (*vesszővel elválasztott értékek*) vagy `DataTable(obj).NET` formában tárolt táblát képes egyesíteni. A keletkező tábla fejléce és az összekapcsoló oszlopnevek is meghatározhatók, ill. lehetőséget biztosít az értékek egyszerű kombinálására is, mint például a súlyozott összegzésre.

4.4.1. Sorszám egyesítése a módosítás értékével

Mivel a fent leírt LIMML formátum statikus kód elemzés eredménytárolására volt kifejlesztve és nem változás követésre, ezért található benne néhány hiányosság a jelen kutatással kapcsolatban. Például az egyes elemek helyét csak a kezdő és a végpozíciójuk sorának és oszlopának számával jelzi; méretét pedig csak az LLOC

Fejlesztői produktivitás megbízható mérése és előrejelzése szoftverfejlesztési projekteken metrika mutatja. Ez elégséges az elemek pontos azonosításához, de nem ad információt a belső elemek cseréjéről, amennyiben azok nem okoznak jelentős méretbeli változást. Ezért a következő példában leírt változtatást nem észleli a program.

Példa

```
public int Calc(      =>  public int Calc(
    int a,              int a,
    int b )             int b )
{
    return a + b;
}                      {
    return a - b;
}
```

A probléma miatt több olyan osztályhoz is 0 értéket rendelt a program, melyek változáson estek át. A hiba kiküszöbölése érdekében az alap kutatás méréseiből származó ECLOC metrika értékét nem dobtam el, hanem egy nagyságrendileg kisebb szorzóval hozzá adtam az általam meghatározott változás mértékéhez.

4.5. Tanulóhalmaz létrehozása

Következő lépésként, az előbbi egyesített táblából létrehoztam a *Weka* által kezelhető tanuló halmazt. A létrehozás a `MergeCsvTable(lib).NET` ismételt használatával, valamint a `Differ(lib).NET` saját fejlesztésű könyvtár segítségével történik. A halmaz tartalmazta a 3.2 és a 3.3 részben ismertetett metrikákat ill. a revíziók számát, valamint az egységnyi időre eső típusos módosítás mértékét. Az osztályok, és a *Productivity Plug-in* által gyűjtött feladat nevek törlésre kerültek, elkerülve ezzel az irreleváns szöveges adat felhasználását a tanulás során. Az fájl szerkesztési ideje, valamint a típusos módosítás értékek szintén törlésre kerültek, mivel ezek közvetlenül felhasznál-

Fejlesztői produktivitás megbízható mérése és előrejelzése szoftverfejlesztési projekteken
nálásra kerültek a célfüggvény értékének kiszámítása során.

4.6. Célfüggvény értékek címkézése

A célfüggvény értéként használt egységnyi időre eső típusos módosítás mértékét három csoportba soroltam: alacsony, normál, és magas címkékkal. A tanulóhalmaz elemeihez egyenletes elosztással rendeltük hozzá a megfelelő címkéket, így a címkék által meghatározott halmazok elemszámai megközelítőleg azonosak.

4.7. Osztályozó betanítása és értékelése

Végül a *Weka* által nyújtott osztályozó algoritmust betanítottam a tanuló halmaz segítségével. Az osztályozó pontosságát kereszt validációs eljárással mértem, melyet szintén a *Weka* rendszer biztosított.

5. fejezet

A genetikus algoritmus

A 4.3.3 részben ismertetett szabályokhoz rendelt súlyok meghatározása két lépésben történt. A kezdeti súlyokat a kutatásban részt vett fejlesztők szaktudása alapján állítottam be. Ez a súlyozás szolgált alapul egy egyszerű genetikus algoritmus számára. Az evolúció célja a kézzel definiált súlyok pontosítása.

Az algoritmus futása során minden evolúciós lépés után rögzítjük a legjobb, a legrosszabb és az átlagos egyed fitness értékét. A futás végén a legjobb egyedhez tartozó súlyozás és az összes, élete során felhalmozott részeredmény mentésre kerül.

5.1. Egyedek

Az algoritmus során egyednek egy adott súlyozással, az adott rendszeren betanított osztályozót tekintettem. Az egyedek genomját a szabályokhoz rendelt súlyok vektora adta.

Keresztezés. Mindig két egyed szolgáltat bemenetként és egy utód született. A súlyokból alkotott vektoron párhuzamosan végighaladva minden érték 50-50 százalékos eséllyel öröklődött az anya ill. az apa oldaláról.

Mutáció. A művelet megváltoztatja az egyedek genomját. Két paramétert vesz át: a mutáció gyakoriságát és mértékét. A súlyvektor meghatározott gyakorisággal kiválasztott elemeit a meghatározott mértékkel növeljük vagy csökkentjük. A növelés és a csökkentés esélye egyaránt 50-50 százalék.

5.1.1. Az egyedek élet ciklusa

Az egyedek életciklusa a születéssel kezdődik, amikor a megkapják a LIMML fájlok helyét és beállításra kerül a genomjuk kezdeti értéke. Életük során végighaladnak a 4. fejezetben részletezett lépéseken. Az utolsó lépés után értelmezhető az életképességüket jelző fitness érték.

Fitness érték. Az utolsó lépésben használt osztályozó F-measure értékét használom az egyedek életképességének meghatározásához. Ez a precision és recall értékek kombinációja, jól jellemzi az osztályozó pontosságát.

5.2. Populáció

A populáció egymástól független egyedek halmaza, vagyis az egyedek közt – a szaporodást leszámítva – nincs információ csere.

Evolúciós lépés. Az evolúció lépések sorozata. Egy lépés során a két legéletképesebb egyed meghatározott számú keresztezésen megy át, vagyis adott számú új

Fejlesztői produktivitás megbízható mérése és előrejelzése szoftverfejlesztési projektekben
utód születik. Gyorsított evolúciós algoritmust használok. Az utódok születésük után rögtön mutáción esnek át és kiértékelésre kerül az életképességük, majd hozzá adódnak a populációhoz. A keresztezések után, meghatározott számú, leggyengébb fitness értékkel rendelkező egyed elpusztul, vagyis törlésre kerül a populációból.

Kezdeti népesség. Az evolúciós ciklus előtt a kezdeti súlyozás alapján adott számú egyed keletkezik. Ezek mindegyike egy kezdeti mutációs gyakoriság és érték szerinti mutáción esik át, megteremtve a lehetőséget a kellően változatos kezdeti népesség létrehozásához.

6. fejezet

Eredmények

A következőkben ismeretem az eddigiekben leírt módszerek pontos paraméterezését, melyet a mérés során használtam, majd bemutatom a kapott eredményeket.

6.1. Mért projektek ismertetése

A méréseket 5 különböző ipari K+F projekten végeztük el. Mindegyik *Java* nyelven íródott és több technológiát is felhasználtak a fejlesztés során. Méretüket tekintve eltérőek, megtalálható köztük kis, közepes, és nagy kódbázisú rendszerek is. A mérést nem a fejlesztés kezdetén kezdtük, hanem az alapvető szerkezetek felépülése után, valamint a végső verzió létrehozása előtt befejeztük.

Időtartamot tekintve összesen 4246 revízió került megvizsgálásra. A revíziók száma nem egyenletesen oszlik meg a projektek között, de megközelítőleg ezer revízióról beszélhetünk rendszerenként.

A megvalósításon dolgozó fejlesztők száma sem egyenletes az egyes projektek között. A fejlesztők személye is változott a fejlesztés ideje alatt, de megközelítőleg 30 fejlesztő dolgozott a mért időtartamban. A legkisebb projekten átlagosan 2, míg

Fejlesztői produktivitás megbízható mérése és előrejelzése szoftverfejlesztési projekteken a legnagyobbban 10.

A továbbiakban a projekteket a mérés során használt belső azonosítóval fogom megkülönböztetni, mely a „Projekt” szóból és a mérés során használt sorszámból áll. Ezt az azonosítót fogom használni a táblázatokban és a diagramokon is.

6.2. A használt metrikák

A mérés során a 3.2 és a 3.3 részben ismertetett folyamat és termék metrikákat használtam prediktorként az osztályozás során.

6.3. A TMOD, típusos módosítás kiszámítása

Az általam használt változás mennyiségét mérő metrikát TMOD (*típusos módosítás*) néven neveztem el, a továbbiakban ezzel a névvel fogok hivatkozni rá.

6.3.1. A vizsgált módosítás típusok

A mérés során kigyűjtöttem a következő módosításokat, melyek a későbbi csoportosítás bemeneteként szolgáltak.

- osztály
 - létrehozás
 - törlés
 - láthatóság változás
 - méret változás
 - hely változás

- metódus
 - létrehozás
 - törlés
 - láthatóság változás
 - paraméter lista változás
 - visszatérési érték változás
- adattag
 - létrehozás
 - törlés
 - láthatóság változás
 - típus változás
 - hely változás

6.3.2. A csoportosítás során használt reguláris kifejezések

A csoportosítás során használt szabályokat és a hozzájuk tartozó reguláris kifejezéseket a 6.1 táblázat tartalmazza. A szabályok és a csoportok közti szoros kapcsolat miatt, a táblázat megadja a kutatás során használt módosítás csoportokat is. Az így definiált csoportok sorrendjét használok fel a későbbiekben a súlyvektorok megadása során.

6.3.3. A csoportok súlyainak meghatározása

A 6.1 táblázatban felsorolt szabályok által definiált csoportokhoz tartozó súlyok megadása során az említett táblázatban szereplő sorrendet követem, vagyis csak a

Szabály típusa	Delta típusa	Elem típusa
Weight Rule ^(obj) _{.NET}	Creation	Attribute
Weight Rule ^(obj) _{.NET}	Deletion	Attribute
Weight Rule ^(obj) _{.NET}	TypeChange	Attribute
Weight Rule ^(obj) _{.NET}	AccessibilityChange	Attribute
Weight Rule ^(obj) _{.NET}	Creation	Method
Weight Rule ^(obj) _{.NET}	Deletion	Method
Weight Rule ^(obj) _{.NET}	PrototypeChange	Method
Weight Rule ^(obj) _{.NET}	ReturnTypeChange	Method
Weight Rule ^(obj) _{.NET}	AccessibilityChange	Method
Weight Rule ^(obj) _{.NET}	SizeChange	Method
Weight Rule ^(obj) _{.NET}	Creation	Class
Weight Rule ^(obj) _{.NET}	Deletion	Class
Weight Rule ^(obj) _{.NET}	AccessibilityChange	Class

6.1. táblázat. TMOD definiálásához használt reguláris kifejezések

Fejlesztői produktivitás megbízható mérése és előrejelzése szoftverfejlesztési projekteken
súlyvektorokat adom meg a 6.2 táblázatban.

A kezdeti súlyozás a kutatásban résztvevők szakmai tapasztalata alapján került meghatározásra. Mivel a kutatásban résztvevők száma alacsony, ezért a szubjektív vélemények kiküszöbölése érdekében a súlyokat genetikusan algoritmus segítségével is meghatároztam, kiindulva a kezdeti súlyokból.

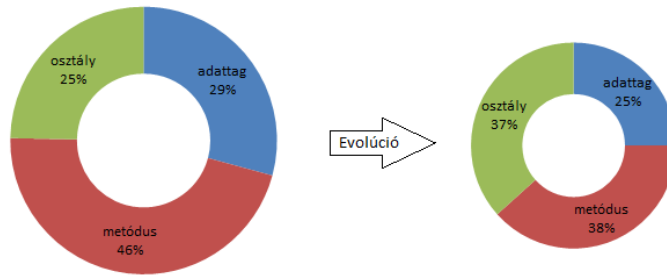
A genetikusan algoritmus során 10 generáció fejlődött ki 10 százalékos mutációs hajlammal és 1 értékű mutációs egységgel, a kezdeti 20 fős népeségből, mely 50 százalékos mutációs hajlammal és 10 értékű mutációs egységgel keletkezett. Minden generáció során 2 utód született és a 2 leggyengébb életképességű egyed pusztult el.

Súlyozás típusa	Projekt neve	Súlyvektor												
Kezdeti	(összes)	4	4	2	3	4	4	3,5	2	3	4	4	4	3
Evolúciós	Projekt1	4	-6	2	-7	4	-6	12,5	12	3	4	4	14	3
	Projekt2	4	-6	-8	-7	-6	-7	2,5	2	3	-6	14	14	-8
	Projekt3	-5	4	11	2	14	4	-6,5	2	3	-6	4	15	-7
	Projekt4	-6	-6	-9	-7	-6	4	3,5	12	2	5	4	4	3
	Projekt5	4	-5	2	3	4	4	3,5	2	13	-6	3	-5	13

6.2. táblázat. TMOD definiálásához használt súlyok

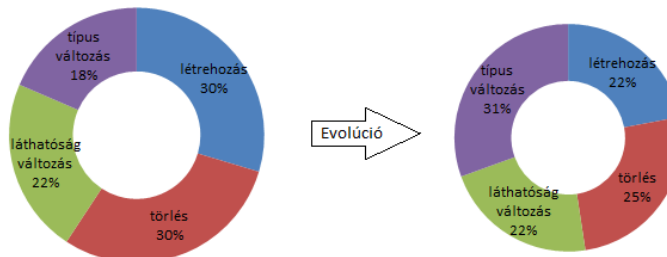
A súlyokat legegyszerűbben az adott csoportba rendelt módosítások fontosságaként értelmezhetjük a produktivitás szempontjából. A projektek közötti összehasonlítás érdekében a kapott súlyokat 0 és 1 közé normáltam, majd súlyoztam a projekt elemzése során használt tanuló halmaz nagyságával. Végül vettem az egyes módosítás csoportokra kapott súlyok átlagát. Az így kapott értékeket két szempont szerint vizsgáltam. Az összehasonlítás alapjaként mindig a kezdeti súlyvektor szolgált.

A 6.1 diagramon látható, hogy a genetikusan algoritmus fontosabbnak ítélte az



6.1. ábra. Az érintett elemhez rendelt súlyok fejlődése

osztályokat és a metódusokat érintő változásokat, az adattagokat érintőkkel szemben. Az osztályok és a metódusok összetettségét és az adattagok hozzájuk viszonyított egyszerűségét figyelembe véve a fontossági sorrendet jogosnak tartom.

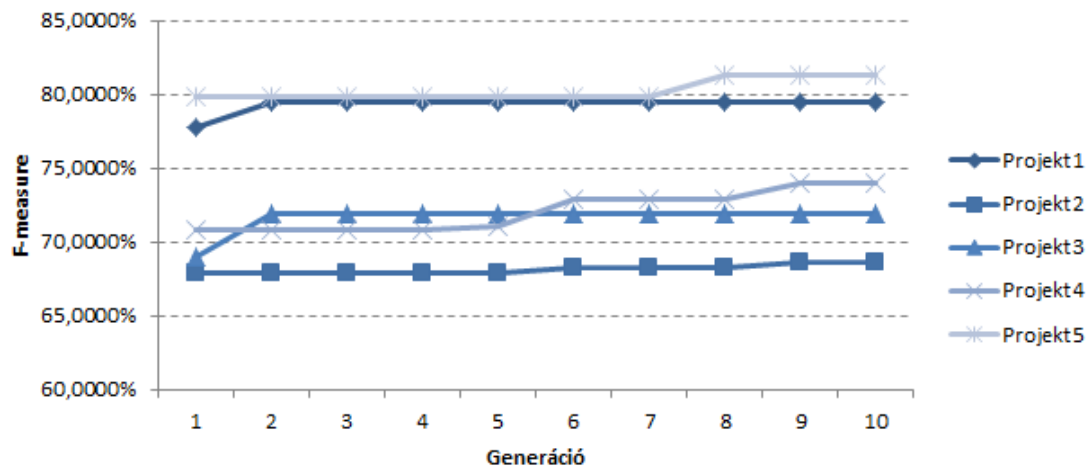


6.2. ábra. A műveletekhez rendelt súlyok fejlődése

A különböző műveleteket közelítőleg azonos fontosságúnak ítélte az algoritmus, amint ezt a 6.2 diagram is mutatja. Ez valószínűleg a projektek jellege és a fejlesztésük során használt technológiák közti eltérésből fakad. Ha az egyes projekteket külön-külön vizsgáljuk a műveletek fontossága is jobban eltér.

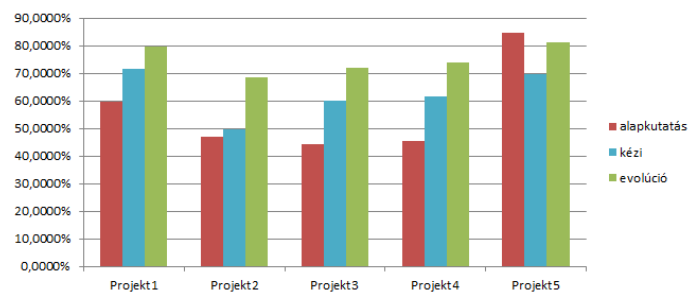
Az egyes projektek mérése során keletkező populációk legéletképesebb egyedeinek fitness értékét a 6.3 diagram mutatja. A generációk számát a időhiány miatt korlátoztam, azonban a fejlődés ilyen rövid evolúció során is megfigyelhető. A mérést végző program optimalizálásával további generációk és szélesebb populációk

Fejlesztői produktivitás megbízható mérése és előrejelzése szoftverfejlesztési projekteken mérését tervezem.



6.3. ábra. A populációk fejlődése

6.4. Eredmények ismertetése és elemzése



6.4. ábra. Az egyes kutatási fázisok során mért eredmények

A kutatás három fázisa – alap kutatás, kézi és genetikus algoritmussal meghatározott súlyok – során kapott F-measure értékeket a 6.4 diagram mutatja. A produktivitás előrejelzésének javulása szinte minden projekt esetében jól látható. Egyetlen

Fejlesztői produktivitás megbízható mérése és előrejelzése szoftverfejlesztési projektekben

kivételt az ötös számú projekt képez, melynél a tanulóhalmaz mérete a legkisebb, így kérdéses, hogy releváns adatnak tekinthető-e. A többi projekt esetében sikerült a kezdeti 60-40 százalékos eredményeket, a kezdeti súlyozás során előbb 50-70, majd a genetikus algoritmus felhasználásával 65-80 százalékra javítani.

6.4.1. Alap kutatás

A fázis során több különböző osztályozó algoritmust pontosságát mérték, melyek eredményei közül a legjobb szerepel a 6.3 táblázatban. Szintén vizsgálták a precision és a recall értékeket. Ezeket figyelmen kívül hagytam, mert a saját kutatásom során e két mérték harmonikus közepét, az F-measure értékét használtam a pontosság mérésére.

6.4.2. Kézi súlyválasztás

A kézi súlyokat a második fázisban résztvevők által kitöltött kérdőív által határoztam meg. Mivel körülbelül 10 százalékos javulást okozott a TMOD bevezetése ezekkel a súlyokkal, így jó kiindulási alapot jelentett a genetikus algoritmus számára.

6.4.3. Genetikus algoritmus által számolt súlyok

Az utolsó evolúciós fázisban kapott eredmények ismét megközelítőleg 10 százalékos pontosság javulást mutattak, mely egyes projektek esetében, a 6.3 diagramon látható fejlődési görbék alapján tovább növelhető az algoritmus hosszabb futtatásával.

Projekt1	Projekt2	Projekt3	Projekt4	Projekt5
59,8000%	47,0000%	44,2000%	45,3000%	84,5000%

6.3. táblázat. Az alapkutatás eredményei

Projekt1	Projekt2	Projekt3	Projekt4	Projekt5
71,3920%	49,8020%	60,0687%	61,5666%	69,5481%

6.4. táblázat. Az kezdeti súlyvektor eredményei

Projekt1	Projekt2	Projekt3	Projekt4	Projekt5
79,5047%	68,6554%	71,8602%	73,9870%	81,2386%

6.5. táblázat. Az genetikus algoritmus eredményei

7. fejezet

Összefoglalás és további kutatási lehetőségek

A mérések eredményéből egyértelműen leolvasható a produktivitás becslés pontosságának javulása. A kezdeti biztató eredményekhez képest tapasztalt jelentős javulás igazolja a terület átfogóbb kutatását. A továbbiakban néhány jövőbeli kutatási irányt ismertetek.

7.1. Kapott súlyok kiértékelése

Az első fő irány a kapott súlyvektorok értékének vizsgálata, mely lehetőséget ad a projektek, a fejlesztők, a technológiák és a költségek közti kapcsolatok mélyebb megértésére.

7.1.1. Nagyobb méretű tanulóhalmaz

Az elemzéshez fontos a lehető legnagyobb tanulóhalmaz vizsgálata. Ezért az eddigi projektek folyamatos mérése mellett továbbiakat is szeretnék bevonni. Valamint a

Fejlesztői produktivitás megbízható mérése és előrejelzése szoftverfejlesztési projektekben

nyelvi paletta szélesítése érdekében az ismertetett bővítményt biztosítanánk a *Visual Studio* fejlesztői környezethez is. Ez lehetővé teszi a C#, VB, C++, és C nyelven írt programok mérését.

7.1.2. Szakértői vélemény bevonása

A kezdeti súlyvektor meghatározásához hasonlóan, tervezzük szélesebb fejlesztői kör tapasztalatait kikérni és összegezni. A kapott súlyok összehasonlítása betekintést adhat a fejlesztők és fejlesztett rendszer viszonyára.

7.1.3. Projektek közötti mérések

A projektek közti különbségek megértése szintén fontos a menedzsment és a programozó szempontjából. A rendszerek azonos vagy átlagolt súlyvektorokkal történő vizsgálata alapján felderíthetjük ezeket az eltéréseket.

7.1.4. Technológiák feladat specifikus produktivitásának becslése

Szintén fontos az egyes technológiák használhatóságának vizsgálata. Szeretnénk összehasonlítani a különböző – akár azonos, akár eltérő nyelven megvalósított – technológiákat a produktivitás szempontjából.

7.1.5. Micro Productivity Profile bevezetése

Az alap kutatásban már használt ún. *Micro Productivity Profile*, lehetőséget biztosít az idő felosztásától függő produktivitás változások feltárására. [10]

7.2. További csoportosítás létrehozása

A jelenlegi produktivitás fogalom pontosítása és elemzése mellett, fontos megvizsgálni más csoportosítási modellek hatékonyságát.

7.2.1. Érintett rendszer komponens alapú csoportosítás

Egy ilyen új csoportosítás lehet például az *érintett rendszer komponens alapú csoportosítás*.^[8] Lényege, hogy a rendszer komponenseit három szintre osztja: rendszer, osztály, és adattag szintre. Minden szint meghatározott elemekből áll. A szintenként megkülönböztet belső vagy elemek megvalósítását érintő (intra) és külső vagyis az elemek közötti (inter) módosításokat.

7.3. Projekt menedzser eszköz megvalósítása

Az elméleti eredmények éles környezetben való használta fontos a szoftverfejlesztéssel kapcsolatos kutatások során. Ezért célunk egy olyan eszköz biztosítása a projekt menedzsment számára, mely segíti a választást a különböző fejlesztési forráskönyvek és használt technológiák közül. Az eszköz kezdeti fejlesztése már elkezdődött. A cél a már bemutatott *ProductivityForward* eszköz első prototípusának továbbfejlesztése.

Irodalomjegyzék

- [1] MK Abdi, H. Lounis, and H. Sahraoui. Using coupling metrics for change impact analysis in object-oriented systems. In *QAOOSE 2006 Proceedings*, page 61. Citeseer, 2006.
- [2] S.R. Chidamber and C.F. Kemerer. A metrics suite for object oriented design. *Software Engineering, IEEE Transactions on*, 20(6):476–493, 1994.
- [3] M. Jørgensen. A review of studies on expert estimation of software development effort. *Journal of Systems and Software*, 70(1):37–60, 2004.
- [4] W. Li and S. Henry. Object-oriented metrics that predict maintainability. *Journal of systems and software*, 23(2):111–122, 1993.
- [5] A. Mockus and D.M. Weiss. Predicting risk of software changes. *Bell Labs Technical Journal*, 5(2):169–180, 2000.
- [6] A. Mockus, D.M. Weiss, and P. Zhang. Understanding and predicting effort in software projects. In *Proceedings of the 25th International Conference on Software Engineering*, pages 274–284. IEEE Computer Society, 2003.
- [7] R. Moser, W. Pedrycz, and G. Succi. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In *Software*

- Engineering*, 2008. *ICSE'08. ACM/IEEE 30th International Conference on*, pages 181–190. IEEE, 2008.
- [8] D.P. Tegarden, S.D. Sheetz, and D.E. Monarchi. A software complexity model of object-oriented systems. *Decision Support Systems*, 13(3):241–262, 1995.
- [9] Gabriella Tóth, Ádám Zoltán Végh, Árpád Beszédes, and Tibor Gyimóthy. Adding process metrics to enhance modification complexity prediction. In *Proceedings of the 19th IEEE International Conference on Program Comprehension (ICPC'11)*, pages 201–204, June 2011.
- [10] Gabriella Tóth, Ádám Zoltán Végh, Árpád Beszédes, Lajos Schrettner, Tamás Gergely, and Tibor Gyimóthy. Adjusting effort estimation using micro-productivity profiles. In *Proceedings of the 12th Symposium on Programming Languages and Software Tools (SPLST'11)*, pages 207–218, October 2011.
- [11] X. Yuan, T.M. Khoshgoftaar, E.B. Allen, and K. Ganesan. An application of fuzzy clustering to software quality prediction. In *Application-Specific Systems and Software Engineering Technology, 2000. Proceedings. 3rd IEEE Symposium on*, pages 85–90. IEEE, 2000.