

TDK dolgozat

Teszthalmazok minőségének javításához módszertan és eszközkészlet

Havas Dávid
III. évf. programtervező informatikus BSc

Témavezető:
Dr. Beszédes Árpád egyetemi adjunktus
Tengeri Dávid PhD hallgató

Szegedi Tudományegyetem
Informatikai Tanszékcsoport
Szoftverfejlesztés Tanszék

Tartalom

1. Bevezetés	3
2. Kutatási kérdések és motiváció	4
2.1. Kapcsolódó munkák	4
2.2. Motiváció.....	4
2.3. Kutatási kérdések.....	4
2.4. Dolgozat felépítése	5
3. Módszertan	5
3.1. GQM.....	5
3.2. Felhasználási lehetőségek.....	6
3.3. Funkcionális csoportok.....	6
3.4. Metrikák a modellhez	7
3.4.1. COV, PART és TpCE	7
3.4.2. EFFiciency	8
3.4.3. SPEC és UNIQ.....	8
3.5. Észrevételek.....	9
3.6. Használat	10
4. Eszközkészlet	10
4.1. Adatszerkezetek.....	11
4.2. Algoritmusok.....	12
4.2.1. Teszthalmaz metrikák.....	12
4.2.2. Teszt kiválasztás és prioritizálás	12
4.2.3. Hiba lokalizáció	12
4.2.4. Teszthalmaz minimalizálás	13
4.3. Eszközök.....	13
5. Teszthalmaz refaktorálása	15
5.1. Csoportok	15
5.2. Refaktorálás	17
5.2.1. Cluster.....	17
5.2.2. Metric	18
5.2.3. Prioritization	18
5.2.4. FI-technique	19
5.3. Tapasztalatok.....	19
6. Összegzés	21
Hivatkozások	23

1. Bevezetés

Komplex szoftverrendszerek fejlesztésénél fontos szempont a forráskód minőségének megőrzése annak érdekében, hogy a rendszerünk később is könnyen karbantartható és továbbfejleszthető legyen. Nagy komplexitású és gyorsan változó rendszereknél fontos, hogy időben észrevegyük a változások folyamán előforduló új hibákat, ami az ilyen jellegű rendszereknél nehéz feladat lehet, mert nem biztos, hogy ismerjük egy-egy változtatás, milyen kihatással van a rendszerünk más részeire. Ezért az ilyen rendszerekhez regressziós tesztek [1], [2], készítenek, amelyekkel a rendszer helyes működését lehet könnyedén ellenőrizni. Az ilyen tesztekkel könnyen detektálhatóak a különböző revíziók között megjelenő hibák és a rendszer rejtett függőségeire is fényt deríthetnek. Ezek a tesztek különböző kategóriákba sorolhatóak, mint pl.: egységtesztek, funkcionális tesztek. Az egyes tesztek összetettségét befolyásolhatják az adott rendszernél alkalmazott tervezési stratégiák. Léteznek olyan tervezési stratégiák is, amelyekben kifejezetten az előre specifikált tesztek köré fejlesztik a rendszert.

Az imént említett tulajdonságok nem minden teszhalmazra egyaránt jellemzőek. Tesztjeink strukturáltsága, komplexitása nagyban befolyásolhatja, a tesztjeink hatékonyságát, hiba-detektáló képességeit. Ezért érdemes lehet a teszhalmazunk minőségével is tisztában lenni és szükség esetén javításokat végezni. A gyakorlatban nem mindig sikerül a teszhalmazunk minőségének a fenntartásához szükséges erőforrásokat biztosítani. Ezen felül a refaktorálás nem is olyan egyszerű feladat, mert nehéz lehet meghatározni a tesztjeink azon halmazát, amelyeken változtatást kell elvégezni. A minőség meghatározásához érdemes lehet különböző lefedettség alapú mértékek alkalmazni. Annak érdekében, hogy ezeket a javításokat egyszerűbbé és gördülékenyebbé tegyük, érdemes lehet valamilyen módszertant követni a művelet során.

Az alábbiakban egy ilyen lefedettség alapú módszertan a TAIME és a hozzá tartozó eszközkészlet a SoDA kerül bemutatásra. A SoDA tartalmazza a szükséges adatszerkezetek és metrikák implementálását, amelyekkel hatékonyan tudjuk a teszhalmazunk minőségét mérni. Majd a bemutatott módszertan és a SoDA-hoz tartozó grafikus felület segítségével, amelynek fejlesztésében részt vettem, a SoDA teszhalmazra kerül refaktorálásra. Az eredmények kiértékelésével és elemzésével igazolom a módszertan alkalmazhatóságát teszhalmazok refaktorálására.

2. Kutatási kérdések és motiváció

2.1. Kapcsolódó munkák

Szoftverrendszerek minőségének ellenőrzése a fejlesztés folyamán egyre nagy szerepet kap. A forráskódunk minőségének a mérésére több elfogadott modell is meghatározásra került [12], [13]. Azonban a forráskódunk minőségének fenntartása önmagában nem elegendő a rendszerek tesztelése során.

Szoftverek tesztthalmazainak minőségének a mérése is hasonlóan kiemelt szempont. A tesztthalmazok minőségmérésének a fő célja, hogy a tesztthalmazunk hibadetektáló képességeit mérhetően ki tudjuk fejezni. Ez a terület széles körben kutatott főleg a teszt kiválasztásnál és rangsorolásnál. Gyakran a lefedettséget alkalmazzák a hibadetektáló képesség méréséhez. Például Rothermel és a társai [14], bemutattak hat lefedettség alapú rangsorolási technikát hibadetektáláshoz.

A tesztminőség egy másik szempontja a tesztthalmazunk hiba lokalizációs képességei. Hibalokalizálásban központi szerepet játszanak a lefedettség alapú technikák. Baudry és társai [15] szembeállítják a minimalizálási és hiba lokalizációs módszereket, mert a hiba lokalizáció pontosabb, ha tesztek több futási információt tartalmaznak.

A hibadetektáló és hiba lokalizációs képességeken túl, egyre nagyobb igény van más metrikákra, amelyekkel a tesztthalmazunk minőségét mérhetjük. Néhány kísérlet található, ahol olyan metrikákat definiáltak, amik tesztthalmazok minőségének mérésére használhatók. Azonban jelenleg nincs olyan általánosan elfogadott mérési módszer, amellyel tesztjeink minőségét mérhetjük.

2.2. Motiváció

Kísérletet az motiválta, hogy szoftverek fejlesztésénél érdemes lehet nagy hangsúlyt fektetni a tesztminőség ellenőrzésére és fenntartására is. Azonban jelenleg nincsenek általánosan elfogadott módszerek tesztminőség méréshez, így ez egy ismeretlen területet. Ezért egy ilyen módszertant érdemes lehet kipróbálni valamilyen gyakorlati példán.

2.3. Kutatási kérdések

Az alábbi kérdések fogalmazódtak meg a témával kapcsolatban:

1. Milyen eszközökkel lehetne hatékonyan mérni különböző rendszerek tesztthalmazának a minőségét?
2. Hogyan alkalmazhatóak ezek az eszközök a gyakorlatban?

2.4. Dolgozat felépítése

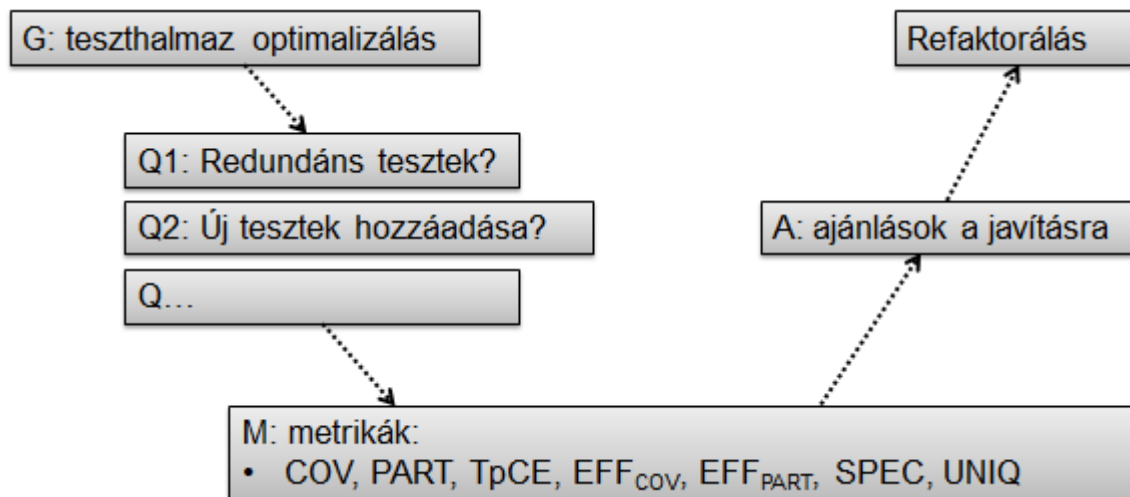
Dolgozatom a következők szerint épül fel. A következő fejezetben bemutatásra kerül egy olyan módszertan, amely az *1. kérdésben* megfogalmazottakhoz nyújt eszközöket és metrikákat a tesztalmazók minőségének a méréséhez. Ezek után a *4. fejezetben* bemutatásra kerül a módszertanhoz tartozó eszközkészlet és használati lehetőségei. Majd az *5. fejezetben* a *2. kérdésben* felmerültekhez egy gyakorlati példán alkalmazom a bemutatott eszközöket és ezek segítségével egy kiválasztott rendszer tesztalmazát próbálom refaktorálni. A javítás befejezése után összehasonlítom a tesztalmaz minőségének javítás előtti és utáni állapotát. Ezek után a módszertan alkalmazásával kapcsolatos tapasztalatok és fejlesztési lehetőségek kerülnek részletezésre.

3. Módszertan

A TAIME (Test Suite Assessment and Improvement Methodology) egy lefedettség alapú módszertan, ami irányelveket ad regressziós tesztalmazunk minőségének a mérésére és esetleges refaktorálására. A TAIME a SoDA függvénykönyvtár alkalmazásaira és szolgáltatásaira épül és úgynevezett Cél-Kérdés-Metrika (GQM) [9] megközelítést alkalmaz a probléma megközelítéséhez. Továbbá a módszertan útmutatást ad a különböző felhasználási helyzetekben alkalmazható eljárásokról, valamint a metrika eredményekhez nyújt észrevételeket.

3.1. GQM

A TAIME-ben a GQM megközelítés segítségével került definiálásra a mérési modell. Általánosságban a GQM a mérési modellt három szintre osztja szét: cél, kérdés és metrika. Ahhoz, hogy ez a megközelítés alkalmazható legyen, először az elérni kívánt célt kell megfogalmazni a meghatározott objektumhoz, jelen esetben a tesztalmazhoz. Ezek után kérdések egy halmaza kerül meghatározásra, amelyek modelleket definiálnak vizsgálandó objektumhoz, aminek segítségével az elérni kívánt cél mértékét jellemezhetjük. Végül metrikák egy csoportja kerül meghatározásra, amelyek mérhető módon adnak választ a korábban feltett kérdésekre. A TAIME környezetében ez a megközelítés az *1. ábrán* láthatóan épül fel:



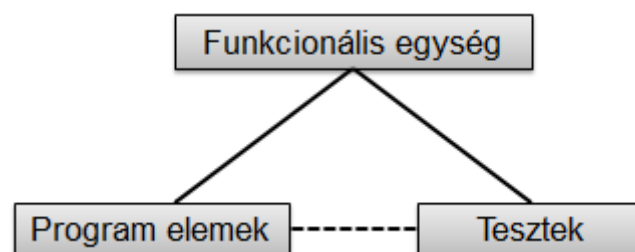
1. ábra GQM megközelítés a TAIME esetén

3.2. Felhasználási lehetőségek

A módszertan különböző helyzetekben alkalmazható adott program teszthalmaz minőségének a javítására vagy ellenőrzésére. A felhasználási lehetőségek lehetnek: teljes teszthalmaz refaktorálás, változás alapú tesztfejlődés vagy „white-box” tesztelés [10], [11]. Az első esetben a szoftver fejlesztésének egy adott pontjában a teszthalmaz teljes refaktorálásra kerül metrikák meghatározása után tesztek módosításával, törlésével vagy hozzáadásával. A második esetben a szoftver fejlesztése közben folyamatosan figyelemmel kísérjük a metrikák változását revíziók között és adott esetben új teszt kerül hozzáadásra vagy törlésre. A harmadik esetben pedig a tesztek tervezése során monitorozzuk a teszthalmazunk minőségét és megvizsgáljuk, hogy az egyes tesztek hozzáadása milyen hatást gyakorol a metrikákra és ez alapján korrigáljuk az adott tesztet, ha szükséges.

3.3. Funkcionális csoportok

Ahhoz, hogy a teszthalmazunk minőségéről pontos képet kapjunk, érdemes lehet teszteket és a program elemeket funkcionális csoportokba sorolni, 2. ábra. Ezeket a csoportokat úgy érdemes kialakítani, hogy a kiválasztott tesztekhez azok a program elemek kerülnek kiválasztásra, amelyeket a kiválasztott tesztek hivatottak tesztelni. Ezek a csoportok lehetnek például az elemzendő szoftver egyes moduljai.



2. ábra Funkcionális egység és elemeinek a kapcsolata

3.4. Metrikák a modellhez

A modellhez alkalmazott metrikák a SoDA függvénytárban kerültek implementálásra. A következőkben a felhasznált metrikák részletes bemutatásra kerülnek.

3.4.1. COV, PART és TPCE

A *code coverage* metrika megmutatja, hogy a program elemek hány százaléka került végrehajtásra a tesztek által. A lefedettség ismerete segíthet a tesztek kiválasztásánál vagy prioritizálásánál. A metrikát egyszerűen lehet kiszámítani a lefedettségi mátrix segítségével. Legyen T tesztek egy halmaza és C a program elemek egy halmaza. Akkor,

$$\text{cov}(T, C) = \frac{|\{c \in C \mid T \text{ - ben lefedésre került } c\}|}{|C|}.$$

Egy másik metrika a *partition*, amely korábban már alkalmazásra került hiba lokalizációs célokra [5]. A partícionáltság azt fejezi ki, hogy átlagosan a program elemek mekkora részét lehet megkülönböztetni más program elemektől lefedettség segítségével. Azok a program elemek, amelyek ugyanazokban a tesztekben voltak lefedve ugyanabba a partícióba kerülnek. Pontosabban, legyen T tesztek halmaza és C program elemek halmaza egy ehhez tartozó partícionálás kifejezhető $\Pi \subseteq \mathcal{P}(P)$ hatványhalmazként. A $\pi_c \in \Pi$ minden $c \in C$ -re definiáljuk, ahol:

$$\pi_c = \{c' \in C \mid c' \text{ pontosan ugyanazokban a tesztekben van lefedve } T \text{ - ben mint } c \text{ is} \}.$$

A hiba lokalizációs alkalmazhatóságot szem előtt tartva, $|\pi_c| - 1$ lesz a c -hez hasonló program elemek száma, ahhoz hogy c megtalálható legyen π_c -ben legfeljebb $|\pi_c| - 1$ -szer kell megvizsgálni [5]. Ezen megfigyelést alapul véve PART a következőképpen formalizálható:

$$\text{PART}(T, C) = 1 - \frac{\sum_{c \in C} (|\pi_c| - 1)}{|P|(|P| - 1)}.$$

A partition és a coverage metrikának az értékkészlete $[0, 1]$ közé esik, ahol a nagyobb érték jelenti a jobbat.

A SoDA-ban megtalálható még néhány származtatott metrika, melyek a korábban bemutatottakra épülnek, ezek segítségével például megvizsgálhatjuk a teszthalmazaink optimalizáltságát. Ebben a környezetben optimalizáltságon a tesztek számának valamilyen más paraméterhez való viszonyát értjük, mint a program elemek vagy más metrikák. A TPCE metrika a tesztek és program elemek számának az aránya és azt mutatja meg, hogy átlagosan hány tesztesettel van egy program elem tesztelve:

$$\text{TPCE}(T, C) = \frac{|T|}{|C|}.$$

3.4.2. EFFiciency

Az *efficiency* metrikák azt mutatják meg, hogy átlagosan egy teszt mekkora mértékben változtat meg más metrikákat. A coverage (EFF_{COV}) és partition (EFF_{PART}) metrikákra az *efficiency* a következő:

$$\text{EFF}_{\text{COV}}(T, C) = \frac{|\{c \in C \mid T - \text{ben lefedésre került } c\}|}{|T|},$$

$$\text{EFF}_{\text{PART}}(T, C) = \frac{\text{PART}(T, C)|C|}{|T|}.$$

A 3. ábrán és az 1. táblázatban egy példa mátrix és a hozzá kiszámított metrika értékek és partíciók találhatók:

$$C = \begin{matrix} & c_1 & c_2 & c_3 & c_4 & c_5 & c_6 \\ \begin{matrix} t_1 \\ t_2 \\ t_3 \\ t_4 \\ t_5 \\ t_6 \\ t_7 \\ t_8 \end{matrix} & \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 \end{pmatrix} \end{matrix}$$

3. ábra Példa lefedettség mátrix

1. táblázat Példa mátrixra kiszámított metrikák és partíciók

Teszthalmaz	Partíciók	COV	PART	TPCE	EFF _{COV}	EFF _{PART}
$\{t_1, t_2\}$	$\{\{c_1\}, \{c_2\}, \{c_3, c_4, c_5, c_6\}\}$	0,33	0,60	0,33	1,00	1,80
$\{t_3, t_4\}$	$\{\{c_1, c_2, c_3, c_4, c_6\}, \{c_5\}\}$	1,00	0,33	0,33	3,00	1,00
$\{t_5, t_6\}$	$\{\{c_1, c_2\}, \{c_3\}, \{c_4, c_5, c_6\}\}$	0,50	0,73	0,33	1,50	2,19
$\{t_7, t_8\}$	$\{\{c_1, c_2\}, \{c_3, c_4\}, \{c_5, c_6\}\}$	1,00	0,80	0,33	3,00	2,40
$\{t_1 \dots t_8\}$	$\{\{c_1\}, \{c_2\}, \{c_3\}, \{c_4\}, \{c_5\}, \{c_6\}\}$	1,00	1,00	1,33	0,75	0,75

3.4.3. SPEC és UNIQ

További metrikaként egyediségi metrikák is megtalálhatók a SoDA-ban, amelyek arról nyújtanak információt, hogy egyes teszt csoportok mennyire egyediek a többi a teszthez képest, azaz ezek a csoportok mekkora mértékben járulnak egyedien hozzá a program elemek lefedettségéhez a többi teszthez képest vagy mennyire átfedőek másokkal. Ahhoz hogy ez kifejezhető legyen a SoDA-ban két metrika van implementálva. A *Specialization* és *Uniqueness* metrikák. Azt a célt szolgálják, hogy megállapítható legyen, melyek azok a tesztek amelyek csak a kiválasztott funkcionális egységet fedik le vagy más csoport is hozzájárul ehhez. Vegyük figyelembe, hogy a teszt csoport vagy funkcionális csoport a teszt halmaznak egy részhalmaza. A funkcionális csoportok a teszthalmaznak azon részhalmazai, amelyekben a tesztek és a program elemek között olyan összefüggés van, miszerint az egy

csoportba kerülő tesztek hivatottak tesztelni az egy csoportba kerülő program elemeket, így alkotva funkcionális egységet. Ahhoz, hogy az egyediségi metrikák kifejezőek legyenek, szükségünk van információra a többi funkcionális egységről is.

Legyen $T \subseteq \mathcal{T}$, ahol $\mathcal{T}$ az összes teszt halmaza. Az első metrika, SPEC megmutatja, hogy T mennyire specializált C programelemek csoportjához. Ami az aránya a C -t lefedő T -beli tesztek számának és a C -t lefedő $\mathcal{T}$ -beli tesztek számának:

$$\text{SPEC}(T, C) = \frac{|\{t \in T \mid t \text{ lefed } C - t\}|}{|\{t \in \mathcal{T} \mid t \text{ lefed } C - t\}|}.$$

Alacsony érték azt mutatja, hogy kevés olyan teszt van, ami lefed a program elemeket az összes többihez hasonlóan, míg egy magas érték azt mutatja, hogy kevés teszt fed le a program elemeket a megadott csoporton kívül.

Egy kapcsolódó metrika a UNIQ, azt mutatja meg, hogy program elemek mekkora részét fed le egyedülállóan egy meghatározott teszt csoport:

$$\text{UNIQ}(T, C) = \frac{|\{c \in C \mid c - t \text{ csak } T \text{ fed le}\}|}{|\{c \in C \mid c - t \text{ lefed } T\}|}.$$

Itt a számlálóba kerülő program elemek azok, amelyeket egy $\mathcal{T} \setminus T$ -beli teszt sem fed le csak T . Hasonlóan, mint az előző esetben jelen esetben is alacsony érték azt jelenti, hogy kevés program elem van egyedülállóan lefedve a megadott teszt csoport által és sok más tesztet fed le ugyanazt. Ugyanakkor a magas metrika értéknél a megadott teszt csoport egyedülálló és kevés más teszt fed le ugyanazokat a program elemeket.

3.5. Észrevételek

A metrikák értékeinek elemzése segít meghozni a megfelelő döntést annak érdekében, hogy a teszthalmaz minőségén javítani tudjunk. A módszertanban néhány észrevétel került rögzítésre, a metrikák értékeire vonatkozóan, hogy mi lehet ezeknek a hatása a teszthalmazra nézve és milyen megoldásokat alkalmazhatunk. Az 2. táblázatban ezek az észrevételek és lehetséges megoldásaik találhatók. A táblázatban semmilyen küszöbérték nem került rögzítésre, mert ezek az értékek nagyban különbözhetnek más teszthalmazokon.

2. táblázat Észrevételek a metrikák értékeivel kapcsolatban

Észrevétel	Hatáskör	Hatás	Megoldás
Alacsony COV	Teszthalmaz	Rossz hiba felderítés	Tesztek hozzáadása
Alacsony PART	Teszthalmaz	Rossz hiba lokalizáció	Tesztek hozzáadása, módosítása
Túl magas TpCE	Teszthalmaz	Nagyobb erőforrás igény	Tesztek törlése
Túl alacsony TpCE	Teszthalmaz	Alacsony függvény lefedettség	Tesztek hozzáadása
Alacsony EFF _{COV}	Teszt	Nagyobb erőforrás igény	Tesztek törlése
Alacsony EFF _{PART}	Teszt	Nagyobb erőforrás igény	Tesztek törlése

Alacsony SPEC	Csoport	Nehezebb teszt kiválasztás	Tesztek hozzáadása, specializálása
Alacsony UNIQ	Csoport	Nehezebb célzott tesztelés	Új egyedi tesztek

3.6. Használat

A TAIME útmutatást nyújt a refaktorálási műveletek elvégzésének megkönnyítésére. Bizonyos helyzetekben változhatnak ezek a lépések, de általánosságban az alábbi lépéseket követhetjük, amennyiben tesztalmazunk refaktorálása a cél:

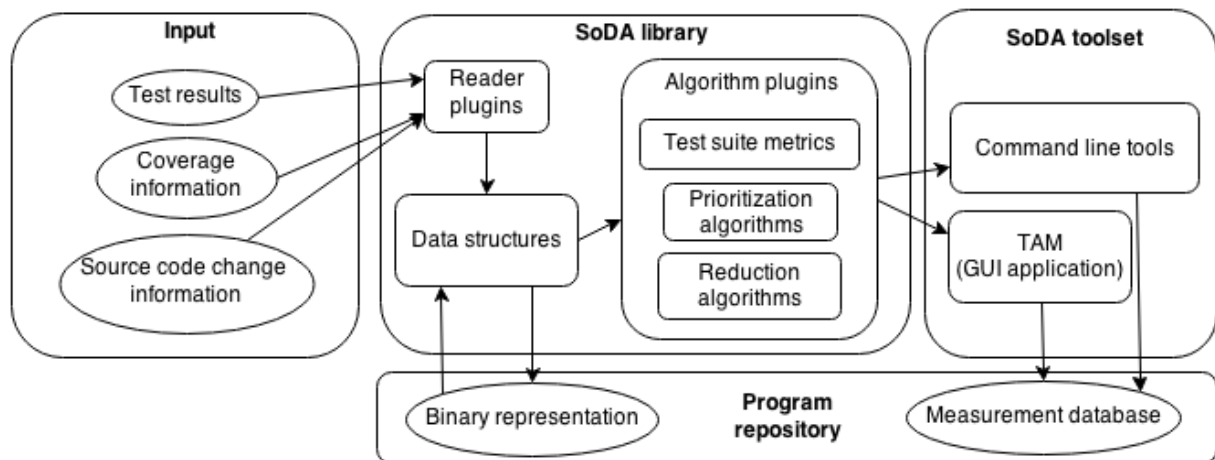
1. A lefedettségi mátrix előállítása
2. A funkcionális egységek meghatározása
3. A lefedettségi mátrix és a funkcionális egységek betöltése a TAM-ba
4. Metrikák kiszámítása
5. Eredmények alapján a további lépések meghatározása
6. A tesztek refaktorálása
7. Metrikák újraszámolása
8. Javulás megerősítése

Különböző lépések elvégzésének az ideje nagyban változhat. Például amennyiben a lefedettségi mátrixhoz szükséges adatok előállításához nem áll rendelkezésre megfelelő eszköz a mátrix első előállítása több időt vehet igénybe. Funkcionális egységek meghatározásánál alkalmazhatunk valamilyen heurisztikát az adatok előállításához vagy a rendszer ismeretében a felhasználók maguk is előállíthatják ezeket. Miután ezek az előkészületek megtörténtek a következőkben a tesztalmaz metrikáit kell meghatározni TAM segítségével. A TAIME-ben használt metrikák a SoDA függvénykönyvtárban vannak implementálva. Amennyiben, a metrika értékek a rendelkezésre állnak akkor az eredmények megvizsgálásával kiválasztásra kerül a megfelelő megoldás, amivel a tesztalmaz minősége javítható. Ezek után a tesztalmazunkat módosítjuk az eredmények alapján, majd a metrikákat a módosított tesztalmazra újramérjük és ellenőrizzük, hogy a megfelelő megoldás került kiválasztásra. Ezek a lépések használhatók az összes korábban bemutatott alkalmazási lehetőség esetében, annyi különbséggel, hogy adott helyzetben ezeket a lépéseket érdemes lehet iteratívan végrehajtani, mint például a változás alapú tesztfejlődésnél, annak érdekében, hogy folyamatos képet tudjunk kapni a metrikák értékének változásáról.

4. Eszközkészlet

A SoDA (Software Development Analysis) [3], [4], egy olyan nyílt forráskódú platform független keretrendszer, amely segítségével a fejlesztők és kutatók egyszerűen

tudnak forráskód lefedettség alapú vizsgálatok végezni, tetszőleges méretű szoftverrendszereken bármilyen megkötések nélkül. A SoDA alkalmazása nem függ a választott programozási nyelvtől sem.



4. ábra SoDA működése

A SoDA keretrendszer különböző csoportokra osztható. A függvénykönyvtár C++ nyelven lett megvalósítva. A függvénykönyvtár tartalmazza az alap adatszerkezeteket, amelyekben lefedettségi adatokat, teszteredményeket és revíziók közötti változásokat tárolhatunk. Továbbá tartalmaz még különböző algoritmusokat, amelyek plugin jellegűen lettek megvalósítva ez által egyszerűen bővíthető a funkcionalitása. Az eszközkészlethez tartozik több parancssoros program, amelyek különböző alkalmazási funkciókat valósítanak meg, ezen kívül egy grafikus felület a TAM (Test suite Analysis and Manipulation) is megvalósításra került. A TAM segítségével könnyedén használhatjuk a SoDA függvénykönyvtár által nyújtott szolgáltatásokat. Az imént említett csoportok részletesen bemutatásra kerülnek az alábbiaknak.

4.1. Adatszerkezetek

A SoDA egyedi formátumban tárolja a lefedettségi információkat. Ez a formátum úgy lett megtervezve, hogy hatékonyan tudja kezelni a nagy mennyiségű adatokat nagy programok és teszthalmazok esetén is. A függvénykönyvtárban három fő adatszerkezet található: lefedettségi-, változás-, és eredmény-mátrixok. Ezekben az adatszerkezetekben tetszőleges részletezettségű adatokat tárolhatunk, mert olyan egyszerű komponensek vannak tárolva, mint a tesztek és a program alkotóelemek (függvények, utasítások...).

A *lefedettségi mátrix* gyakorlatilag egy bit-mátrix, ami a tesztek és a program alkotóelemeinek a lefedettségét mutatja. Ahhoz, hogy a mátrix elemei könnyen azonosíthatók legyenek tárolásra kerülnek ezen elemek nevei. A függvénykönyvtár lehetőséget ad a mátrix

elemeinek könnyű elérésére a nevük alapján. Lehetőség van a meghatározott tesztek és program alkotóelemek szűrésére is.

Az *eredmény mátrix* tárolja a tesztek futásának az eredményét a különböző revíziók között. A tesztek futásának eredményén felül az is eltárolásra kerül, hogy az adott teszt futtatva lett-e a különböző revíziókon.

SoDAban külön adatszerkezet került implementálásra annak érdekében, hogy adott revíziók közötti *változások* nyomon követhetők legyenek. Ezeket az adatokat a különböző verziókövető rendszerekből gyűjthetjük össze. Ezek az adatok hasznosak lehetnek tesztek prioritizálásánál vagy kiválasztásánál különböző szempontok szerint. Az adatszerkezetben tetszőleges számú program elemet és revíziót tárolhatunk el.

4.2. Algoritmusok

Különböző algoritmusok kerültek implementálásra, amelyek használhatóak különböző tesztelési tevékenységeknél, amelyek hasznosítani tudják a lefedettségi információkat, mint például teszt prioritizálás és kiválasztás, hibalokalizálás stb. Az algoritmusok plugin jellegű felépítettsége miatt könnyedén tudjuk bővíteni a jelenlegi kategóriákat, de új elemzési kategóriákkal is bővíthető szükség esetén.

4.2.1. Teszthalmaz metrikák

SoDA-ban több mérték is megvalósításra került, amelyekkel a teszthalmazaink karakterisztikájáról kaphatunk információt különböző nézőpontokból. Hasonlóan, mint a szoftvereknél használt forráskód metrikák. A megvalósított metrikák korábban ismertetésre kerültek a módszertannál.

4.2.2. Teszt kiválasztás és prioritizálás

Ezek a módszerek lehetőséget adnak regressziós tesztek kiválasztásos és rangsorolt újratestelésére, ami által csökkenthető a teszteknek a futási ideje. A SoDA-ban megvalósításra került néhány népszerű rangsoroló algoritmus, amik a lefedettség alapján működnek: *general coverage* és *additional coverage* [6], ezen kívül néhány hasznos módszer is implementálásra került, mint például *véletlenszerű rangsorolás*.

4.2.3. Hiba lokalizáció

A hiba lokalizáció egy olyan módszer, amelynek célja a tesztelők és a fejlesztők segítése a tesztelésbe, hibakeresésbe, hibajavításban, úgy hogy megpróbálja azonosítani a hiba a helyét a programban. Gyakran alkalmazott módszer a hiba lokalizációra a Spectrum-Based Fault Localization (SBFL) megközelítés [7]. A SBFL működése a program spektrumon

alapul, ami a program viselkedésének egy lenyomata és azt mutatja meg, hogy a program mely részei kerültek végrehajtásra a futtatás során [8]. A program spektrum gyakorlatilag a lefedettségi mátrix ebben a környezetben. A módszer figyeli a különbségeket a sikeres és sikertelen tesztek lefedettsége között és megpróbálja meghatározni azt a program elemet, amelyik okozhatta a teszt elbukását. A SBFL kockázati értéket használ, hogy előre jelezze a relatív kockázatát minden egyes program elemnek, ami a hibát tartalmazza. A kockázati értékek meghatározására kockázati formulával történik. Több különböző módszernél definiáltak ilyen formulákat, mint például Tarantula, Ochiai és Jaccard [7]. A SoDA-ban az Ochiai és a Tarantula SBFL van implementálva.

4.2.4. Teszthalmaz minimalizálás

A teszthalmaz minimalizálásnak az a célja, hogy találjunk tesztek egy olyan minimális halmazát, amely kielégíti a meghatározott feltételeket. Ilyen feltétel lehet, hogy maximalizáljuk a kiválasztott tesztek hibadetektáló képességét (gyakran összekapcsolódik a forráskód lefedettséggel). Egy másik lehetséges alkalmazása hiba lokalizáció, ahol a cél az, hogy olyan tesztek kerüljenek kiválasztásra, amelyek segítenek meghatározni egy hiba helyét (ehhez a problémához a partíciós metrika társítható). A problémák a minimális lefogó pontthalmaz probléma esetei, ami NP-teljes. Ezért az algoritmusok valamilyen heurisztikát alkalmaznak a megadott feltételek eléréséhez. A legtöbb algoritmus a teszt rangsorolásra épül, amelyből kiválasztásra kerül a megfelelő számú teszt adott sorrendben.

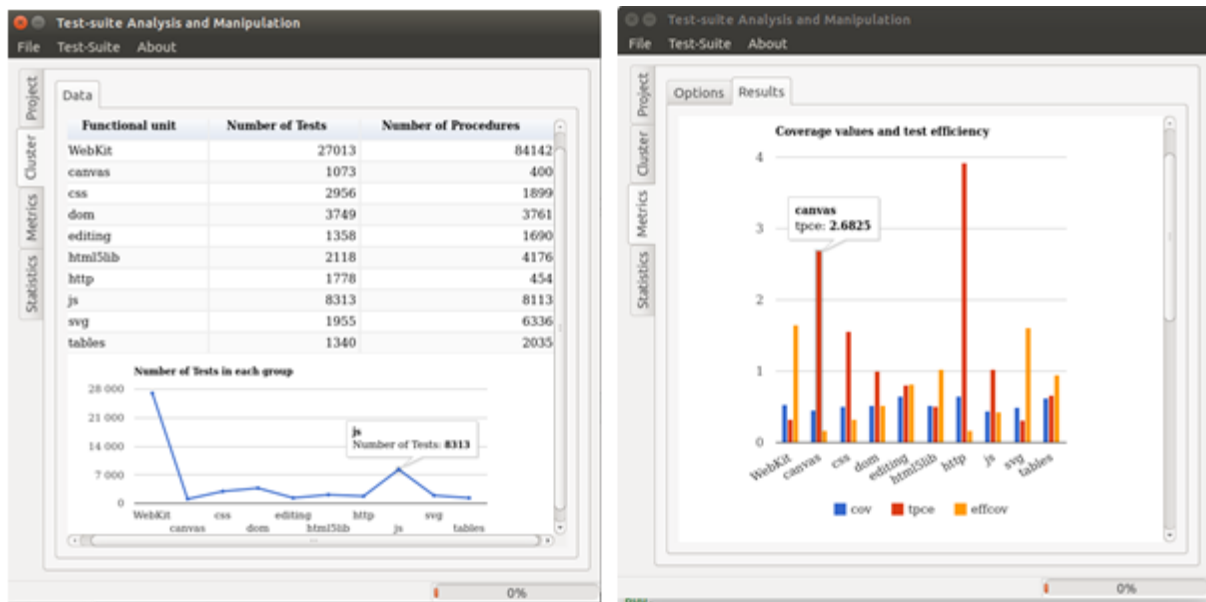
A SoDA-ban három minimalizáló algoritmus lett megvalósítva. Ezek közül kettő algoritmus hibadetektálásra épül, a *Naiv lefedettség alapú* és *Kiegészített lefedettség alapú* módszerek [5]. A harmadik algoritmus hiba lokalizációra épül, aminek az a célja, hogy a teszthalmaz hiba lokalizációs képességét a lehető legmagasabb legyen, minél kevesebb teszt bevonásával. Ez a partíció alapú módszer [4].

4.3. Eszközök

A SoDA szolgáltatásainak alkalmazásához két fajta felhasználói program érhető el: parancssoros programok és grafikus felület. A parancssoros programok a SoDA összes funkciójához elérhetőséget biztosítanak egységesített felületen keresztül. Így ezek az eszközök egyszerűen integrálhatóak sokkal összetettebb elemző scriptekbe és környezetekbe.

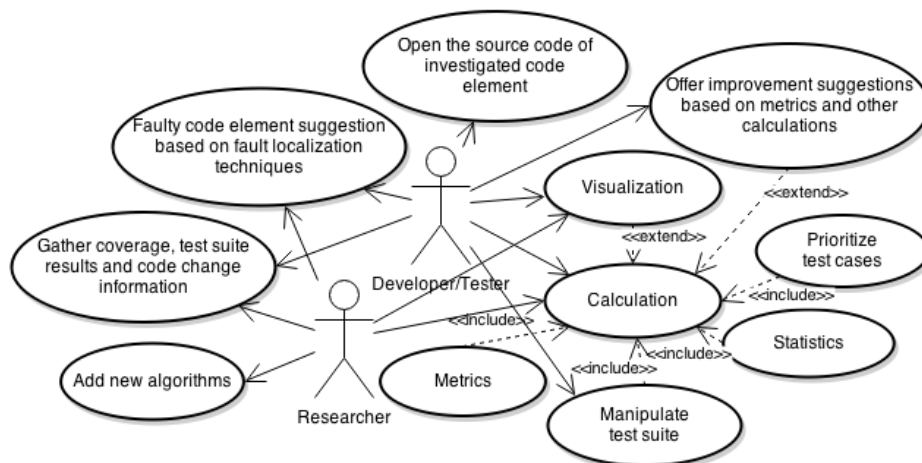
A TAM egy grafikus felülettel rendelkező alkalmazás, amit azzal a céllal terveztünk, hogy a SoDA könyvtár összes funkcióját könnyedén használni tudják kutatók, fejlesztők és tesztelők is. A felhasználóknak lehetőséget biztosít, hogy egyszerűen tudjanak teszthalmazokat elemezni és jelentéseket készíteni. A korábban bemutatott SoDA

függvénykönyvtár adatszerkezeteit és algoritmusait használja a különböző mérésekhez. Ezen felül számos hasznos funkciót is megvalósítottunk. Különböző mérések konfigurációit menthetjük el, grafikusan vizsgálhatjuk a tesztalmazunk részeit, betölthetjük különböző verzióit a programunknak és különböző módokon ábrázolva vizsgálhatjuk az eredményeinket. Abban az esetben, ha méréseinkre később is szükség lenne, vagy újra el szeretnénk ezeket a méréseket végezni a TAM lehetőséget ad az adatok elmentésére, amit szükség esetén mozgathatunk a munkaállomások között.



5. ábra TAM grafikus felülete

A legtöbb funkció elérhető a TAM-ból és a parancssoros alkalmazásokból is. A 6. ábrán látható a SoDA és a TAM alkalmazási lehetőségei. Kutatók és fejlesztők számára egyaránt hasznosak lehetnek ezek a szolgáltatások. Kutatók egyszerűen adhatnak hozzá új algoritmusokat és hasonlíthatják össze a jelenlegi megoldásokkal. Fejlesztők számára pedig a teszt rangsorolása segíthet regressziós teszteknel a megfelelő tesztek kiválasztásában, ezen kívül a hiba lokalizációs módszerekkel meghatározott lehetséges hibás program elemek listája is hasznos lehet adott esetben. A TAM külön felületet biztosít ezeknek a program elemeknek a listázására és attól függően milyen nevezési konvencióval látjuk el őket könnyedén megtalálhatjuk ezeket forráskódunkban. Hatékony felület található a funkcionális egységek szerkesztésére és vizsgálatára is, aminek segítségével a tesztelési folyamat hatékonyabbá tehető.



6. ábra TAM és SoDA lehetséges alkalmazásai

5. Teszthalmaz refaktorálása

Miután megismerkedtünk a SoDA és TAIME nyújtotta lehetőségekkel a továbbiakban egy gyakorlati példán keresztül vizsgáljuk meg a módszertan az alkalmazhatóságát. Ennek elvégzéséhez segítségünkre lesz a TAM és a TAIME-ben megadott irányelvek. A kiválasztott rendszer, amin a korábban ismertetett eszközök lesznek használva pedig a SoDA.

A SoDA egy közepes méretű rendszer, aminek a függvénykönyvtárhoz és a megvalósított algoritmusaihoz tartozik egy teszthalmaz. A teszthalmazban unit tesztek, és integrációs tesztek találhatók. Az integrációs tesztek a megvalósított algoritmusok teszteléséhez szükségesek, míg az adatszerkezetekhez és az egyéb modulokhoz unit tesztek kerültek megvalósításra.

A refaktorálás elkezdéséhez mindenekelőtt a lefedettség mátrix előállítására volt szükségünk. Ahhoz, hogy a gyorsan változó teszthalmazból könnyen elő tudjuk állítani az újabb lefedettség mátrixokat olyan eljárást kellett megvalósítani, aminek segítségével ez gördülékenyen megvalósítható, de ennek a megoldását most nem részletezzük. Természetesen ez a lépés különböző rendszereknél más és más folyamatokkal járhat. Miután a lefedettség mátrixot sikeresen előállítottuk a következő lépés a funkcionális egységek/csoportok meghatározása. Az jelenlegi környezetben a mátrix program elemeit SoDA függvényeinek tekintjük.

5.1. Csoportok

Ahhoz hogy a funkcionális csoportokat jól határozzuk meg, mindenképpen szükséges a rendszer belső felépítésének az ismerete. Amennyiben ilyen információkkal nem rendelkező valamilyen heurisztika alapján is kialakíthatóak ezek a csoportok. SoDA esetében például a

könyvtárstruktúra alapján. A SoDA függvényeit és tesztjeit összesen hét funkcionális csoportba osztottuk be. A 3. táblázatban a csoportok nevei és méretei találhatók a refaktorálást megelőző állapotban.

3. táblázat SoDA funkcionális egységeinek adatai

Csoport	Teszttek száma	Függvények száma
SoDA	111	411
data	86	213
io	13	56
reader	4	27
metric	3	42
prioritization	2	15
fl-technique	2	12
cluster	1	21

A *data* csoportba a különböző adatszerkezeteket implementáló függvények kerültek. Az *io* csoportba a különböző fájlműveleteket végző metódusok kerültek. A *reader* csoportba olyan egységek kerültek, amelyek SoDA-val kompatibilis fájllokká alakítják a különböző formátumú adatokat. A *metric* csoportba tartozik több metrika megvalósítása, amik egy teszt halmaz tulajdonságait határozhatják meg. A *prioritization* csoportba kerültek a teszt prioritizáló algoritmusok. A *fl-technique* csoportban kaptak helyet a hiba lokalizációs eljárások. Végezetül a *cluster* csoportban kaptak helyet különféle csoportosító algoritmusok.

A legtöbb csoport mérete meglehetősen kicsi, egy kivétellel a *data* csoport. Ennek több oka is van, egyrészt a *data* csoportba tartoznak az adatszerkezeteket implementáló függvények és a SoDA-nak a legnagyobb részét az adatszerkezetek teszik ki. Továbbá ez a legnagyobb és legrészletesebben tesztelt csoport is. Többi csoport azért tartalmaz arányaiban kevesebb tesztet azon felül, hogy méretüket nézve sokkal kisebbek, mert a funkcionalitás bővítése folyamán nem fordítottunk kellő erőforrást a tesztek karbantartására. Ezen kívül számos csoport, hasonló karakterisztikával rendelkezik ezért a javításuk során e csoportok hasonló irányú változásokat fognak mutatni. Ezek a csoportok a *reader*, *metric*, *prioritization*, *fl-technique*, *cluster*. A csoportok közös tulajdonsága, hogy mindegyik csoport dinamikusan betöltődő osztályokat valósít meg. A különböző típusú plugineknek kötött a formátuma, ezért az 1 plugin által minimálisan hozzáadott függvények száma egy csoportra nézve kötött. Valamint a pluginokhoz tartozó függvények meghívási sorrendje is kötött, ennek okán nem lehet annyira részletesen tesztelni egy-egy plugint és ezért lettek integrációs tesztek megvalósítva a tesztelésükhöz. Legtöbb esetben minden algoritmushoz egy teszt valósítható meg hatékonyan.

5.2. Refaktorálás

Miután a csoportokat meghatároztuk és lemértük a kezdeti értékek a különböző eredmények kiértékelésével eldöntjük, hogy melyek azok a csoportok, amelyeket javítani szeretnénk. A SoDA csoportjainak metrikáinak javítási előtti értékei a 4. táblázatban láthatók.

4. táblázat Refaktorálás előtti metrika értékek

Csoport	COV	EFF _{COV}	PART	EFF _{PART}	TpCE	SPEC	UNIQ
SoDA	0,73	2,63	0,92	3,33	0,28	-	-
reader	0,96	6,50	0,83	5,63	0,15	1,00	1,00
io	0,88	3,77	0,95	4,10	0,23	0,35	0,37
fl-technique	0,83	5,00	0,80	4,82	0,17	1,00	1,00
data	0,82	2,02	0,96	2,38	0,4	0,87	0,57
prioritization	0,67	5,00	0,74	5,55	0,13	1,00	1,00
metric	0,50	7,00	0,69	9,66	0,07	1,00	1,00
cluster	0,38	8,00	0,50	10,5	0,05	0,14	0,00

A javításra kiválasztott csoportokat a COV mérték alapján választottuk, továbbá figyelembe vettük, hogy hány teszt hiányzik egy-egy kisebb csoportból. Ezért a kiválasztott csoportok a *metric*, *cluster*, *prioritization* lett. Ezeken a csoportokon új tesztek hozzáadásával vagy módosításával próbáltam javítani. Továbbá kiválasztásra került még az *fl-technique* csoport is, amelynek, annak ellenére, hogy a COV értéke magas azért került be a javításra szánt csoportok közé, hogy megvizsgáljam van-e valamilyen módja a csoport metrikáinak javítására.

5.2.1. Cluster

A *cluster* csoportban nagyon alacsony volt a COV mérték ezért az elsődleges cél között a lefedettség javítása volt. A csoporthoz összesen négy új teszt került hozzáadásra és minden egyes teszt után méréseket végeztünk, hogy megállapítsuk az egyes mérések utáni változást. Az 5. táblázatban a metrikák értékei hasonlíthatóak össze tesztek hozzáadása között.

5. táblázat Cluster csoport metrika változásai a refaktorálás során

Tesztek száma	COV	EFF _{COV}	PART	EFF _{PART}	TpCE	SPEC	UNIQ
1	0,38	8	0,5	10,5	0,048	0,14	0,00
2	0,48	5	0,66	6,93	0,095	0,29	0,30
3	0,57	4	0,66	4,62	0,14	0,29	0,30
4	0,71	3,8	0,66	3,47	0,19	0,29	0,30
5	0,81	3,4	0,87	3,64	0,24	0,5	0,59

A táblázatban jól látható hogy egyes tesztek hozzáadása mekkora mértékben járult hozzá a különböző mértékek javulásához vagy romlásához. Megfigyelhető, hogy a második, harmadik és negyedik teszteknel, bizonyos mértékek nem változtak, míg a többi mérték követte az eredeti tendenciát. A négy új teszt hozzáadása után a javítást befejeztük, mert a

kitűzött célt sikeresen elértük és minden egyes a csoportba tartozó csoportosító algoritmushoz lett teszt megvalósítva.

5.2.2. Metric

A *metric* csoportban az elérni kívánt célok között szerepel a lefedettség javítása valamint új tesztek létrehozása olyan metrika algoritmusokhoz, melyek nem rendelkeznek jelenleg teszttel. A csoporthoz hat új teszt lett implementálva és a méréseket az összes új teszt mátrixba kerülése után mértük le, ami miatt a SPEC és UNIQ értékek az első új teszt hozzáadása után lecsökkentek, majd szépen lassan visszatértek az eredeti állapotba, ahogy egyre több új és új teszt került a csoportba. Ez a jelenség azért fordulhatott elő, mert valójában a lefedettségi mátrixban megtalálhatóak voltak az új tesztek lefedettségi adatai, de a funkcionális csoportba csak a függvények voltak megtalálhatók, az ezekhez tartozó még nem. A 6. táblázatban ezek az eredmények találhatók.

6. táblázat Metric csoport metrikáinak változása a refaktorálás során

Tesztek száma	COV	EFF _{COV}	PART	EFF _{PART}	TpCE	SPEC	UNIQ
3	0,5	7	0,69	9,66	0,071	1	1
4	0,55	5,8	0,73	7,67	0,095	0,44	0,17
5	0,6	5	0,77	6,47	0,12	0,56	0,32
6	0,64	4,5	0,81	5,67	0,14	0,67	0,37
7	0,64	3,9	0,81	4,86	0,17	0,78	0,67
8	0,69	3,6	0,84	4,41	0,19	0,89	0,69
9	0,74	3,4	0,87	4,06	0,21	1	1

Hat új teszt hozzáadása után a javítást abbahagytuk, mert már minden pluginhoz készült teszt és a korábbi pontban említettek miatt ennél a csoportnál már nem lehet részletesebb tesztek készíteni. A mértékek az előzőekben látottakhoz hasonló tendenciákat vázolnak fel azzal a különbséggel, hogy jelen esetben a COV és PART mértékeknél vehető észre stagnálás.

5.2.3. Prioritization

A *prioritization* csoportban három rangsoroló algoritmus található, amelyből kettőhöz már létezik teszt. Így a cél a lefedettség maximalizálása egy új teszt létrehozásával és a jelenlegi tesztek módosításával.

7. táblázat Prioritization csoport metrika változásai a refaktorálás során

Tesztek száma	COV	EFF _{COV}	PART	EFF _{PART}	TpCE	SPEC	UNIQ
2	0,67	5	0,74	5,55	0,13	1	1
3	1,00	5	0,77	3,85	0,20	1	1

A teljes lefedettség eléréséhez önmagában nem lett volna elég egy új teszt létrehozása.

Ezért a TAM segítségével megvizsgáltuk a jelenlegi csoportban lefedetlen függvényeket és ez alapján módosítottuk a létező teszteket.

5.2.4. Fl-technique

Az *fl-technique* csoportban minden pluginhoz tartozott teszt így ebben a csoportban ezen tesztek módosításával próbáltam javítani a lefedettségen.

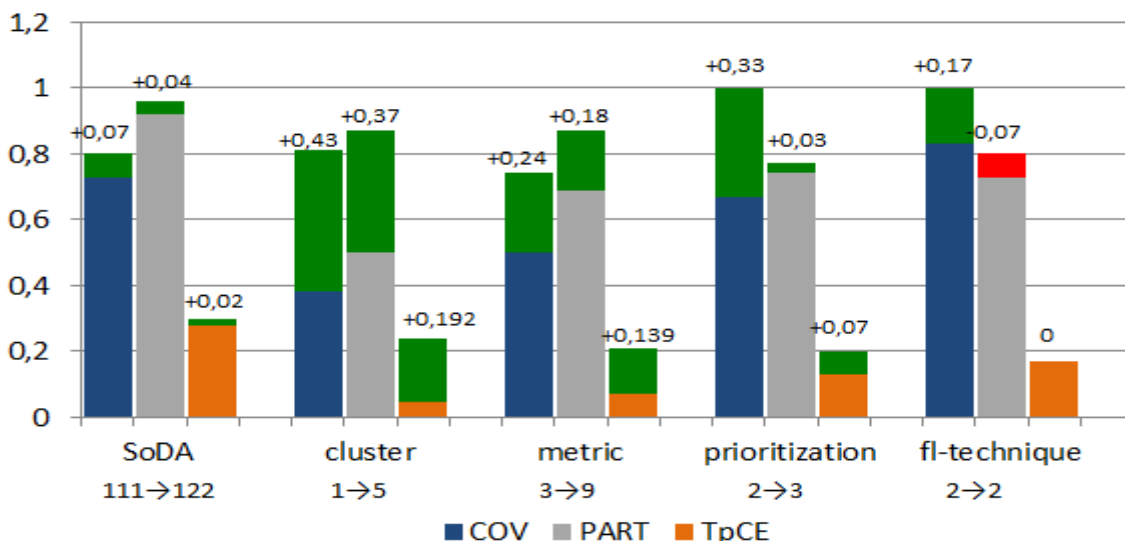
8. táblázat A fl-technique csoport metrika változásai a refaktorálás során

Teszt szám	COV	EFF _{COV}	PART	EFF _{PART}	TpCE	SPEC	UNIQ
2	0,83	5	0,80	4,82	0,17	1	1
2	1,00	6	0,73	4,36	0,17	1	1

Ugyanazzal a módszerrel, mint az előző csoport esetén sikerült javulást elérni a lefedettséggel, habár jelen esetben a PART értéke romlott.

5.3. Tapasztalatok

Az alábbi táblázatban megfigyelhetők a javítás utáni mérések eredményei, amelyben jól látható, hogy a teljes SoDA metrikái csak kis mértékben változtak a különböző csoportokban végrehajtott módosítások hatására. A javítások eléréséhez csak létező teszteket módosítottunk és új teszteket adtunk hozzá a tesztalmazhoz. Összesen tizenegy új teszt lett létrehozva. Az alábbi 7. ábrán leolvasható a metrikák változása a refaktorálás hatására és információk a tesztek számának változásáról.



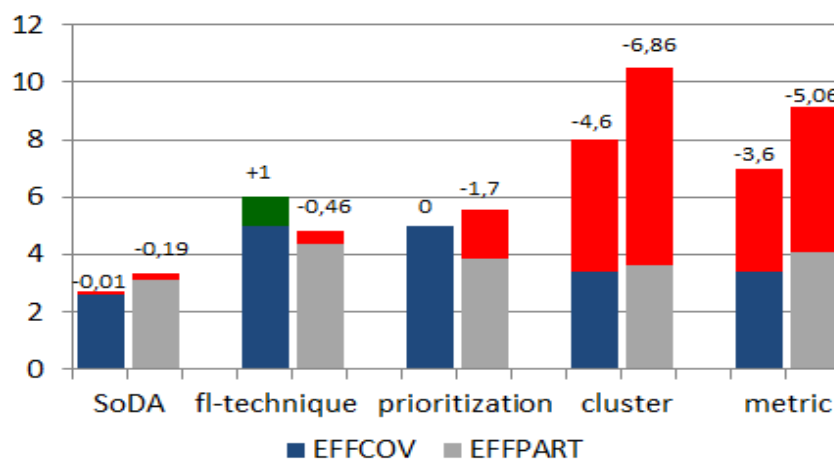
7. ábra COV, PART, TpCE metrikák változása refaktorálás előtt és után

Az összes érintett csoportban a lefedettség maximalizálása volt a cél és ezt sikerült is elérni a *fl-technique* és a *prioritization* csoportban. Sajnos a *cluster* és *metric* csoportban ezt nem sikerült elérni külső tényezők miatt. Sajnálatos módon az alkalmazott módszer a

lefedettségi mátrixok előállítására nem pontos így bizonyos kódrészleteket nem tud azonosítani.

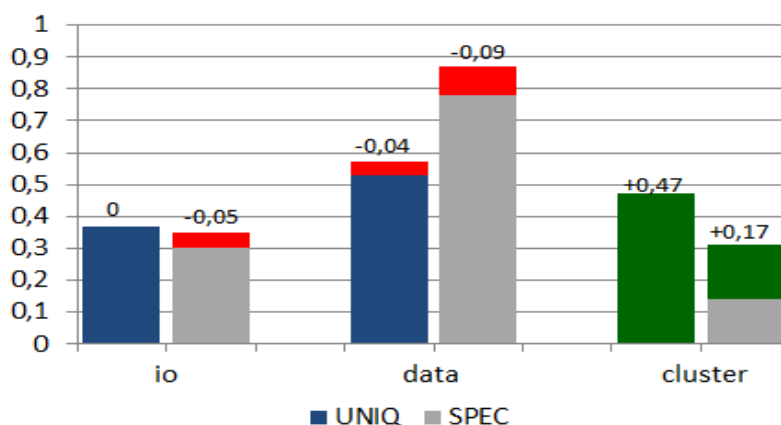
Ezen kívül észrevehető, hogy az összes csoportban alacsonyok a TpCE értékek annak ellenére a COV értékek magasak. Ez azért lehetséges, mert egyes tesztek nagyszámú függvényt érintenek. A tesztek száma nagyban függ a tesztelés részletességétől és az adott rendszer korlátozásaitól, funkcióinak megvalósításától.

A 8. ábrán látható, hogy a refaktorálás hatására az EFFiciency metrikák értékei milyen irányba változtak. Kiemelendő a *cluster* és *metric* csoportok, nagymértékű romlás következett be. Ennek az az oka, hogy több új teszt került a csoportba, mint amennyi teszt eredetileg volt, így nagymértékben megváltoztatva a csoport lefedettségi mátrixát és ennek hatására a csoport a csoport karakterisztikája nagymértékben megváltozott.



8. ábra EFFiciency értékek változása a refaktorálás hatására

SPEC és UNIQ értékeket nagyban befolyásolja az, hogy egyes tesztek mennyire fednek le olyan függvényeket, melyek másik csoportban találhatók. A 9. ábráról megfigyelhető, hogy a kiválasztott csoportokon végzett refaktorálás, milyen mértékben befolyásolhatja bizonyos csoportok metrikáit. A kiválasztott csoportok közül csak a *cluster* csoport SPEC és UNIQ tudtuk javítani a maradék három csoportban nem változott ez a maximális érték. Továbbá észrevehető, hogy a refaktorálásból kihagyott *data* és *io* csoportok metrika értékei romlottak. Ennek az oka az, hogy a többi csoportba tartozó funkciók nagyban támaszkodnak *data* és az *io* csoport függvényeire



9. ábra SPEC és UNIQ metrikák változása a refaktorálás hatására

A módszertan alkalmazása a SoDA-ra könnyen kivitelezhető volt. A szükséges adatfájlok létrehozása és a funkcionális egységek meghatározása, amennyiben nem rendelkezünk előkészített eszközökkel további tervezést igényelhet, de ezek után a tesztthalmaz refaktorálása gyorsan haladt. A teszteket és a csoportokat a TAM-ban egyszerűen lehet módosítani és gyorsan újra lehet mérni a változásokat.

A refaktorálás hatására a tesztekkel így új eddig ismeretlen hibákat lehet felderíteni adott esetben.

6. Összegzés

Részletesen bemutatásra került egy olyan lefedettség alapú módszertan és eszközkészlet, mely segítségével hatékonyan tudjuk szoftverrendszerek tesztthalmazának minőségét mérni és javítani a fejlesztés során. A módszertan esetén részletezésre kerültek a módszertan legfontosabb elemei, mint például a funkcionális egységek, a felhasznált metrikák és ezek lehetséges jelentése, különböző alkalmazási lehetőségek.

Továbbá ismertettem a TAIME-hez tartozó eszközkészletet a SoDA-t és a hozzá tartozó grafikus környezet a TAM-ot, amely az 1. kérdésre ad választ. Majd ezen eszközök segítségével a SoDA tesztthalmazán a gyakorlatban is ellenőriztem a bemutatott eljárások használhatóságát, amely a 2. kérdésre ad választ. A SoDA tesztthalmazához elemzéséhez szükséges fájlok elállításához szükséges eszközök megvalósítása után meghatároztam a funkcionális egységeket, majd a metrikák meghatározása után kiválasztottam azokat a csoportokat, amelyeket javítani szeretnék. Ezek után a részletesen mértem a csoportok metrikáinak változását minden egyes változtatásnál. A kiválasztott csoportok javítása után összehasonlítottam a javított metrikákat a kezdeti állapottal.

Végül pár szóban összegeztem a módszertan alkalmazásával kapcsolatos tapasztalatokat. A korábban részletezett tapasztalatok alapján a bemutatott módszertan hatékonyan alkalmazható tesztalmazók minőségének az elemzésére és javítására.

TAM használata során sok új tapasztalatot gyűjtöttünk és több új fejlesztési lehetőség is felmerült, amelyek hasznosak lehetnek ilyen típusú tevékenységeknél, mint például több tesztalmaz hozzáadásának lehetősége, felület a különböző tesztalmazok közötti változások egyszerű összehasonlítására, dinamikusan alakítható felület az eredmények ábrázolására, a tesztalmaz módosításának lehetősége grafikus felület segítségével. A módszertan kis és közepes méretű rendszereken hatékonyan használható azonban a nagy rendszereken való alkalmazáshoz jelenleg nem rendelkezünk ismeretekkel így ez még egy vizsgálandó alkalmazási terület.

Hivatkozások

- [1] L. Copeland, A Practitioner's Guide to Software Test Design. Artech House, 2004.
- [2] A. Bertolino, "Software testing research: Achievements, challenges, dreams," in 2007 Future of Software Engineering. IEEE Computer Society, 2007, pp. 85-103.
- [3] Tengeri, D., Beszédes, Á., Havas, D. and Gyimóthy, T. Toolset and Program Repository for Code Coverage-Based Test Suite Analysis and Manipulation. In Proceedings of the 14th IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM 2014), pages 47-52.
- [4] "Soda framework and repository," <http://www.sed.inf.u-szeged.hu/soda>.
- [5] L. Vidács, Á. Beszédes, D. Tengeri, I. Siket, and T. Gyimóthy, "Test suite reduction for fault detection and localization: A combined approach," in Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on. IEEE, 2014, pp. 204–213.
- [6] G. Rothermel, R. H. Untch, C. Chu, and M. J. Harrold, "Prioritizing test cases for regression testing," Software Engineering, IEEE Transactions on, vol. 27, no. 10, pp. 929–948, 2001.
- [7] S. Yoo, "Evolving human competitive spectra-based fault localisation techniques," in Search Based Software Engineering. Springer, 2012, pp. 244–258.
- [8] M. J. Harrold, G. Rothermel, R. Wu, and L. Yi, "An empirical investigation of program spectra," in ACM SIGPLAN Notices, vol. 33, no. 7. ACM, 1998, pp. 83–90.
- [9] V. Basili, G. Caldiera, and H.D. Rombach, "Goal Question Metric Approach," Encyclopedia of Software Engineering, pp. 528-532, John Wiley & Sons, Inc., 1994.
- [10] Ehmer Khan, Mohd (July 2011). „Different Approaches to White Box Testing Technique for Finding Errors”. International Journal of Software Engineering and Its Applications 5: 1-6.
- [11] International Software Testing Qualification Board. Certified Tester Foundation Level Syllabus – version 2011. ISQTB 2011.
- [12] Bakota, T., Hegedűs, P., Körtvélyesi, P., Ferenc, R. and Gyimóthy, T. A probabilistic software quality model. In Software Maintenance (ICSM 2011) 27th IEEE International Conference on, pp 243 -252, Sept. 2011.
- [13] S. Wagner, F. Deissenboeck: An Integrated Approach to Quality Modelling. In Proc. 5th Workshop on Software Quality (5-WoSQ). IEEE Computer Society Press, 2007.

- [14] G. Rothermel, R. J. Untch, and C. Chu, “Prioritizing test cases for regression testing,” *IEEE Trans. Softw. Eng.*, vol. 27, no. 10, pp. 929–948, Oct. 2001.
- [15] B. Baudry, F. Fleurey, and Y. Le Traon, “Improving test suites for efficient fault localization,” in 28th international conference on Software engineering, ser. ICSE ’06. ACM, 2006, pp. 82–91.