Visszatervező eszközök kiértékelése és továbbfejlesztése

Ph.D. értekezés tézisei

Fülöp Lajos Jenő

Témavezető:
Dr. Gyimóthy Tibor

Informatika Doktori Iskola
Informatikai Tanszékcsoport
Szegedi Tudományegyetem

Szeged
2011

Bevezetés

Örökölt szoftverrendszerek karbantartása jelentős költségekkel járhat. Legtöbbször egy ilyen rendszer újratervezése jobb választás mintsem újraírni azt a semmiből [9]. Az újratervezés két fázisból áll: a jelenlegi rendszerben található információk visszatervezéséből és - erre az információra alapozva - a jelenlegi rendszer egy új verziójának megtervezéséből. A sikeres újratervezés az örökölt rendszer egy nagyon megbízható visszatervezését követeli meg, mivel bármilyen döntés az új rendszer megtervezését illetően a visszatervezés során meghatározott információn alapszik. Ez motivált minket egy olyan módszer kifejlesztésére, amely bővíti és javítja az egyik visszatervező eszközünket, valamint arra, hogy visszatervező eszközök kiértékeléséhez benchmarkokat fejlesszünk ki és kísérleteket végezzünk el.

A munka során tervezési minta keresőkkel, duplikált kód detektorokkal és szabálysértés ellenőrzőkkel foglalkoztunk. A tervezési minta kereső eszközök segítik egy rendszer és annak komponenseinek megértését. A duplikált kód detektáló eszközök olyan veszélyes forráskód másolatokat azonosítanak, melyek potenciálisan ugyanazokat a hibákat tartalmazhatják és ezáltal a rendszer karbantartását megnehezítik. A szabálysértéseket ellenőrző eszközök tipikus programozási hibákat ellenőriznek a forráskódban. Az ilyen eszközök örökölt rendszerekről fontos információkkal szolgálnak, de az eredményeik tartalmazhatnak hamis találatokat is.

Továbbfejlesztettük a tervezési minta kereső eszközünket, amely a Columbus visszatervező keretrendszer [6] egyik komponense. A mintafelismerést tovább pontosítottuk úgy, hogy az eredeti algoritmus által visszaadott mintajelölteket igaznak vagy hamisnak jelöltük meg [34], majd gépi tanulási módszereket alkalmaztunk. Ezután három tervezési minta felismerő eszközt is összehasonlítottunk (Columbus, Maisa and CrocoPat) három szempontból: találatok közötti különbségek, idő- és memóriaigény [35]. Majd kísérleteket végeztünk el egy újonnan kifejlesztett benchmarkon (DEEBEE), amely tervezési minta kereső eszközök kiértékelését és összehasonlítását támogatja. A benchmark segítségével két tervezési minta kereső eszköz pontosságát értékeltül ki a tervezési minták referencia implementációján, valamint két szoftverrendszeren [36, 37, 38]. Bevezettük a DPDX-et is, amely egy közös csere formátum tervezési minta felismerő eszközök számára. Itt megadtunk egy jól definiált és bővíthető metamodellt, amely a jelenlegi eszközök számos hiányosságát kezeli. A metamodellt egy XML alapú nyelvben implementáltuk [39, 40]. Végül, de nem utolsó sorban bevezettük a DEEBEE rendszer egy új verzióját, a BEFRIEND-et, amely szélesebb körben használható, mivel általánosítottuk a kiértékelési szempontokat és a kiértékelhető eszközök típusát. A BEFRIEND-el a forráskód tetszőleges tulajdonságait felismerő visszatervező eszközök eredményei értékelhetőek ki és hasonlíthatóak össze egymással [41, 42].

Az értekezés öt tézist tartalmaz, melyek a következők:

1. **Egy tervezési minta kereső eszköz továbbfejlesztése.**
2. **Tervezési minta keresők performanciájának kiértékelése.**
3. **Tervezési minta kereső eszközök validációja.**
4. **Terezési minta kereső eszközök közös formátuma.**
5. **Visszatervező eszközök validációja.**

A következő fejezetekben röviden ismertetem ezeket a téziseket és minden fejezet végén kiemelem a saját hozzájárulásomat.

1. Egy tervezési minta kereső eszköz továbbfejlesztése

A legtöbb mintaillesztésen alapuló mintafelismerési megközelítéssel az a probléma, hogy természetüknél fogva számos hamis eredményt állítanak elő, ahol bizonyos kódrészek - melyek a mintaleírásnak csak strukturálisan felelnek meg - mintajelöltekként kerülnek azonosításra. Most bemutatunk egy gépi tanulási módszert és néhány kísérleti eredményt, melyek megmutatják, hogy hogyan javíthatóak tervezési minta kereső eszközök eredményei és hogy hogyan dönthető el, hogy az eredmények helyesek-e vagy sem. A munka során a Columbus keretrendszer tervezési minta felismerőjét alkalmaztuk [4] és kísérleteket végeztünk el a StarOffice [28] szövegszerkesztőjén, a StarWriteren, amely több mint 6000 osztályt tartalmaz. Itt a Columbus mintafelismerő algoritmusát alkalmazva számos mintajelöltet találtunk, melyeket tovább szűrtük gépi tanulási algoritmusokkal, így végül pontosabb eredményeket sikerült elérnünk.

A tanulási folyamat

A következőkben egy áttekintést adunk a kifejlesztett tanulási folyamat egyes lépéseiről.

1. *Prediktor értékek kiszámítása.* Az eredeti mintafelismerő folyamatban [4] a Columbus (CAN) analizálja a forráskódot és készít egy Absztrakt Szemantikus Gráf (ASG) reprezentációt. Ezután a Columbus tervezési minta felismerő komponense (CAN2Dpm) azonosítja a tervezési mintákat (jelölteket), melyek megfelelnek az aktuális DPML (Design Pattern Markup Language) fájlnak, ami az éppen keresett minta struktúráját írja le. Minden egyes tervezési minta rendelkezik olyan tulajdonságokkal (prediktorok), melyek nem kapcsolódnak annak szerkezeti leírásához. Ezeket az információkat az ASG-ből nyerjük ki és elmentjük egy CSV fájlba (prediktor tábla fájl), amit a tanulórendszereknek adunk meg.

2. *Kézi vizsgálat.* A forráskód kézi vizsgálata alapján eldöntjük, hogy egy adott mintajelölt hamis vagy sem. A prediktor tábla fájlt bővítjük egy új oszloppal ami a kézi vizsgálat (döntés) eredményét tartalmazza.

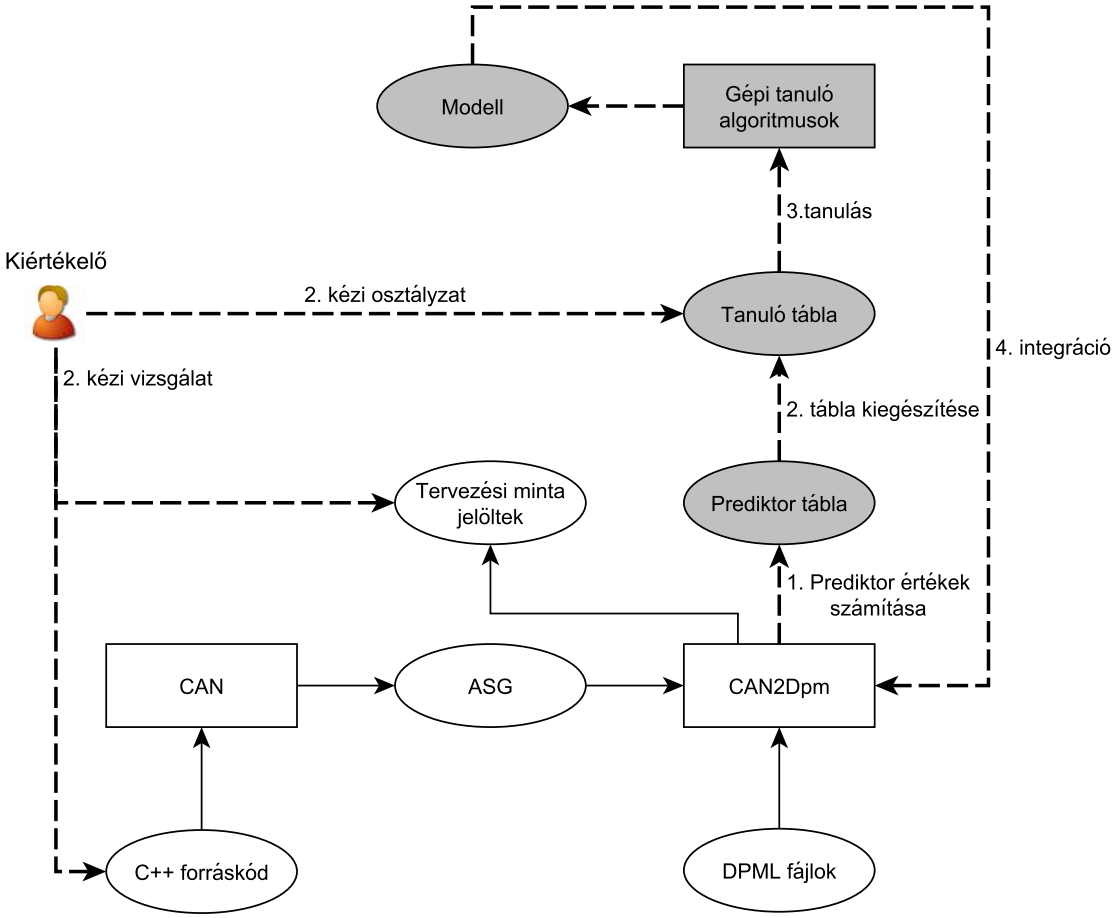
3. *Gépi tanulás.* Itt elvégezzük a gépi tanuló algoritmusok (döntési fa [25] és neurális háló [7]) betanítását. Ezek kimenetei a megtanult tudást tartalmazó modellek lesznek.

4. *Integráció.* Végül, az eredeti folyamatba integráljuk a gépi tanulás eredményeit.

Az 1. ábra grafikusan mutatja be ezt a folyamatot. A Columbus keresési folyamatának eredeti elemeit egyenes vonalak és üres dobozok jelölik, míg a jelen tanulmány által újonnan bevezetett részeket szaggatott vonalak és teli dobozok.

Kísérletek

Két tervezési mintával (Adapter Object és Strategy) végeztünk el kísérleteket. A tanulási folyamat sikerességének kiértékeléséhez a keresztvalidáció módszerét alkalmaztuk: a prediktor táblát felosztottuk három egyenlő részre és a tanulást-tesztelést három alkalommal végeztük el. A tanulási pontosságot minden esetben a tanulási rendszer helyes döntéseinek száma (a kézi osztályozáshoz képest) és



1. ábra. A tanulási folyamat

Tervezési minta	Döntési fa	Neurális háló
Adapter Object	66.70% (21.79%)	66.70% (23.22%)
Strategy	90.47% (4.13 %)	95.24% (4.12 %)

1. táblázat. Átlagos pontosság és szórás (zárójelben) a keresztvalidáció alapján

az összes jelöltek számának arányaként definiáltuk. Kiszámítottuk az átlagot és a szórást a három tesztelési eredményt felhasználva és a 1. táblázatban látható eredményeket kaptuk.

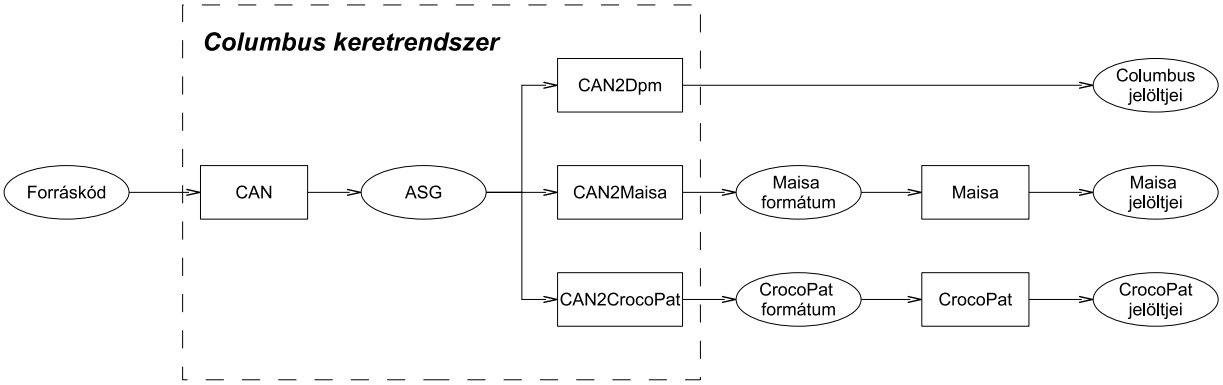
A célünk a struktúra alapú mintafelismerő eszközünk eredményeiből [4] a hamis jelöltek kiszűrése volt. A kísérleteink során 67–95%-os pontosságot értünk el, és az előállt modellel képesek voltunk a hamis Adapter Object jelöltek 86%-át, valamint a hamis Strategy jelöltek 94%-át kiszűrni.

Saját hozzájárulás

A szerző végezte el a kísérleteket a Strategy tervezési mintával, valamint kézzel felcímkézte az alap tervezési minta kereső eszköz Strategy mintára adott eredményeit. Ennek a tézisnek az eredményeit a [34] cikkben publikáltuk.

2. Tervezési minta keresők performanciájának kiértékelése

Ebben a tanulmányban három tervezési minta kereső eszközt hasonlítottunk össze, a Columbust, a Maisat és a CrocoPatet. Azért választottuk ezeket az eszközöket, mert a Columbussal közös bemenetet lehet gyártani mindhárom eszköz számára. Korábbi munkánk már biztosította a bemenetet a Maisa számára [12], míg a CrocoPat esetében kifejlesztettünk egy új Columbus plugint, amely képes a megfelelő formátumot előállítani. Ezt a folyamatot a 2. ábrán illusztráljuk.



2. ábra. Keretrendszer

Összehasonlítási módszer

Tervezési minta kereső eszközökhöz kialakítottunk egy összehasonlítási módszert, mely a következő szempontokat tartalmazza.

- *Az azonosított tervezési minta jelöltek közötti különbségek.* Az eszközök néha ugyanazt a mintajelöltet eltérően közlik. Ezeknek a különbségeknek számos oka lehet. Az eszközök különböző technikákat alkalmazhatnak a tervezési minták definiálására, valamint az eredmények reprezentálása is eltérhet. Kísérletek során megpróbáltuk felderíteni a különbségek lehetséges okait.
- *Sebesség.* A sebesség az eszköz által az adott tervezési minta felismeréséhez szükséges időt jelenti.
- *Memória használat.* A tervezési minta felismeréséhez szükséges maximális memóriaigényt mértük.

Kísérletek

Négy nyílt forráskódú kisebb-nagyobb rendszeren (DC++, WinMerge, Jikes and Mozilla) végeztünk összehasonlítást, ezzel elértük azt, hogy az eredmények függetlenek legyenek olyan rendszerjellemzőktől mint például a méret, a komplexitás és az alkalmazás típusa. Minden tesztet ugyanazon a számítógépen végeztük, ezáltal a mért értékek függetlenek a hardvertől és az eredmények összehasonlíthatóak. A tesztszámítógép 3GHz-es Intel Xeon processzorral és 3GB memóriával rendelkezett.

Tervezési minta jelöltek közötti különbségek.

Az eszközök által azonosított mintajelöltek az esetek többségében lényegében megegyeztek volna, ha a különbségek következő gyakori okaitól eltekintünk.

Tervezési minták eltérő definiálása. Megfigyeltük, hogy néhány tipikus oka volt annak, hogy az eszközök ugyanazon mintajelölteket eltérően jelentették. A fő ok az volt, hogy bizonyos esetekben a tervezési minta leírás nem vett figyelembe egy résztvevőt, ahogyan a Maisa a Builder minta esetében. Itt a Maisa által adott definíció nem tartalmazta a Director résztvevőt, így a Maisa jelöltjei nem egyeztek a másik két eszköz jelöltjeivel. Ilyen különbségek persze lehetnek szándékosak de véletlenszerűek is, emiatt ezeket nehéz egységesíteni és kiküszöbölni.

Minta leírások pontossága. Egy másik különbség a mintaleírások pontossága volt. Például a Jikes esetében az Adapter Class jelöltek száma azért tért el, mert a CrocoPat és a Columbus a Target szereplőt absztraktnak definiálta, míg a Maisa nem.

Eltérő algoritmusok. Különbségeket találtunk a tervezési minta felismerő algoritmusokban is. Például a Columbus és a Maisa bizonyos osztályokkal közösen rendelkező jelölteket egyszer jelentik, de a CrocoPat minden előfordulást külön-külön közöl. A jelöltek helyességének korrekt összehasonlításakor az ilyen jellegű különbségeket kezelni és egységesíteni kell.

Sebesség.

Összességében arra jutottunk, hogy a legjobb eszköz a sebességet tekintve a CrocoPat, habár bizonyos esetekben a Columbus gyorsabb volt. A Columbust kis és közepes méretű rendszerek és bonyolultabb tervezési minták (pl. Visitor) esetében célszerű használni. Ami a CrocoPatet illeti, nagyobb rendszerek esetén vagy egyszerűbb tervezési mintáknál (pl. Template Method) előnyös alkalmazni.

Memória használat.

Az eredmények azt mutatták, hogy a memóriafogyasztás jelentősen függ az elemzett projekt méretétől, viszont független az adott tervezési mintától. A Columbus esetében a szükséges memória igen nagy volt a másik kettőhöz képest. Ez annak a következménye, hogy a Columbus egy általános visszatervező keretrendszer és a tervezési minta felismerés csak egy képessége a sok közül. Ebből adódóan ASG reprezentációt használ, ami minden információt tartalmaz a forráskódról. Megjegyezzük, hogy a CrocoPat és Maisa memóriaigénye kisebb volt, mivel a bemeneteik csak a mintafelismeréshez szükséges információt tartalmazták a forráskódról. Összességében, a memóriaigényt tekintve a Maisa teljesített a legjobban.

Saját hozzájárulás

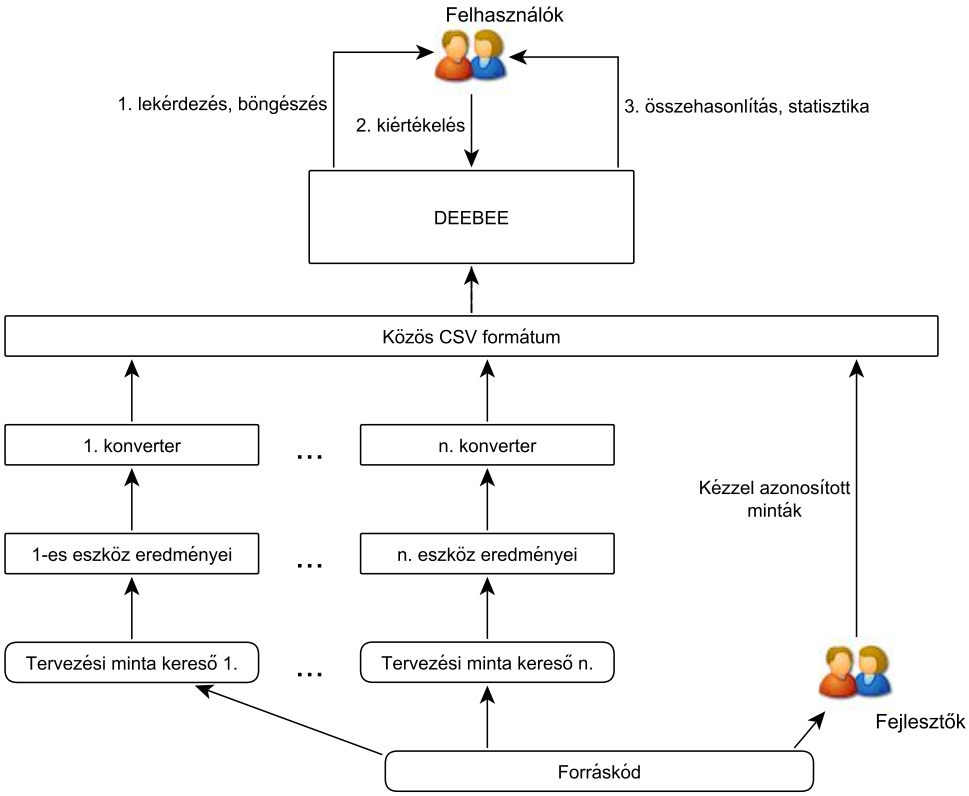
A szerző fejlesztette ki a Columbus-CrocoPat exportert és integrálta a Columbus keretrendszerbe. Számos tervezési mintát is definiált a CrocoPat reprezentációs nyelvén (RML). Továbbá, a szerző végezte el a tézisben bemutatott kísérleteket, és részt vett az összehasonlítási és kiértékelési módszer meghatározásában is.

3. Tervezési minta kereső eszközök validációja

Kifejlesztettünk egy publikusan elérhető benchmarkot, a DEEBEE-t (DEsign pattern Evaluation BEnchmark Environment), tervezési minták kiértékelésére és összehasonlítására. A benchmarkunk általános: nyelv-, szoftver-, eszköz- és mintafüggetlen. Ezzel a benchmarkkal az eszközök pontossága (precision) és teljessége (recall) bárki által validálható.

A benchmark

A 3. kép a benchmark folyamatát mutatja be alulról felfelé. Első lépésként a tervezési minta kereső eszközök és a fejlesztők tervezési mintákat azonosítanak a forráskódban. Ezután a tervezési minta kereső eszközök a saját, specifikus formátumukban kiírják eredményeiket, melyeket ezután konvertálni kell a DEEBEE bemeneti formátumára (ami egy CSV fájl).



3. ábra. DEEBEE folyamat

A fejlesztők időközben kézzel azonosítanak tervezési mintákat és az eredményeiket közvetlenül a DEEBEE CSV fájlformátumában tárolják el. A következő lépésben, a CSV fájlba összegyűjtött jelölteket feltöltjük a benchmarkba. Ezután a DEEBEE online felületén keresztül lehetőség nyílik a jelöltek lekérdezésére, kiértékelésére és összehasonlítására. Sőt, a benchmark képes automatikusan statisztikákat generálni (pl. pontosság és teljesség) a felhasználók korábbi szavazatai alapján. Ezek a funkcionalitások szignifikánsan megkönnyítik a tervezési minta kereső eszközök kiértékelésének és összehasonlításának folyamatát.

Ahogyan a második tézispontban bemutattuk, a tervezési minta kereső eszközök eredményei számos ok miatt eltérhetnek. Ennek kezelésére DEEBEE-ben bevezettünk egy módszert. A lényegében megegyező, de eltérően közölt mintajelölteket *testvéreknek* neveztük el. A testvérek azonosítása a mintajelöltek *alapvető résztvevőin* alapul. Például a State [13] minta esetében az alapvető résztvevő a State osztály szerepét játsza el.

A benchmark 1,274 tervezési minta jelöltet tartalmaz három C++ szoftverrendszerből (Mozilla [21], NotePad++ [22] és FormulaManager [29]), három Java szoftverrendszerből (JHotDraw [17], JRefactory [18] és JUnit [19]) és tervezési minták C++ referencia implementációiból. A feltöltött mintajelölteket három tervezési minta kereső eszköz azonosította: a Columbus (C++) [4], a Maisa (C++) [23] és a Design Pattern Detection Tool (Java) [30].

Kísérletek

A Gamma és társai által publikált könyv [13] alapján a tervezési minták referencia implementációit készítettük el az eszközök alapos összehasonlításához. Ezekkel a referencia implementációkkal a C++ tervezési minta kereső eszközök alapvető képességei kiértékelhetőek és összehasonlíthatóak. Mivel a referencia implementációk a tervezési minták egy összefüggéstelen és mesterkélt megvalósításai, emiatt kifejlesztettünk a FormulaManager programot, ahol minden tervezési minta legalább egyszer előfordul valós környezetben. Továbbá szoftverfejlesztők által a NotePad++-ban azonosított minta példányokat is hozzáadtunk a benchmarkhoz.

Két tervezési minta kereső eszközt - *Maisat* és *Columbust* - értékeltünk ki és hasonlítottunk össze a DEEBEE segítségével. Az eszközöket a *referencia implementációkon*, a *FormulaManageren*, és a *NotePad++-on* értékeltük ki. Az eredményeket a 2. táblázat mutatja be.

Rendszer	Referencia impl.		NotePad++		FormulaManager	
Eszköz	Columbus	Maisa	Columbus	Maisa	Columbus	Maisa
Pontosság	100%	80.00%	62.50%	16.67%	52.27%	80%
Teljesség	58.33%	33.33%	29.41%	11.76%	71.88%	25%

2. táblázat. Eredmények

Saját hozzájárulás

A szerző fejlesztette ki a benchmarkot az instance view kivételével. Továbbá definiálta és implementálta a feltöltési formátumot, a testvér mechanizmussal együtt. A szerző végezte el a kísérleteket a Maisa és Columbus eszközökkel, valamint feltöltötte az eredményeiket a benchmarkba. Részt vett a benchmark architektúrájának megtervezésében is, a kiértékelési szempontok meghatározásában, az eszközök eredményeinek kézi kiértékelésében és a benchmark használati eseteinek megtervezésében.

4. Tervezési minta kereső eszközök közös formátuma

A jelenlegi tervezési minta kereső eszközök (DPD) kimeneti formátumainak hiányosságait próbáltuk korrigálni, pótolni egy jól definiált és bővíthető metamodellen alapuló, közös csereformátum (DPDX) bevezetésével. Ez a formátum segíti a különböző mintakereső eszközök kimeneteinek összehasonlítását [35], fúzióját [20], vizualizációját [10], és validációját [37].

Követelmények

A következő alapvető követelményeket támasztjuk a közös csereformátummal szemben azért, hogy a jelenlegi mintakereső eszközök kimeneti formátumainak hiányosságait pótolja és az eszközök "szövetségének" az alapja legyen.

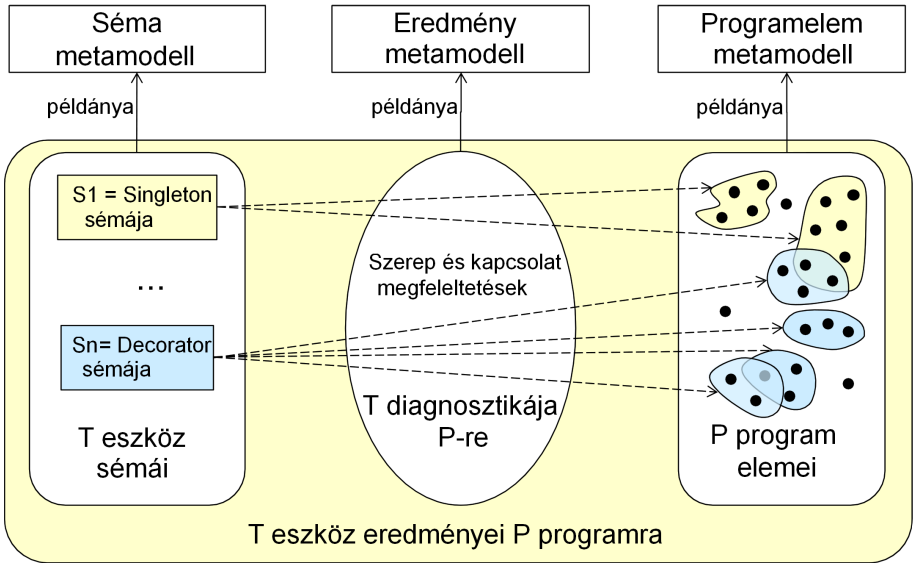
1. **Specifikáció.** A csereformátumot formálisan kell specifikálni ahhoz, hogy a DPD eszközfejlesztők könnyen tudjanak generátorokat, elemzőket és konvertereket implementálni.
2. **Reprodukálhatóság.** Az eszközt és az elemzett programot expliciten kell közölni ahhoz, hogy a kutatók reprodukálni tudják az eredményeket.
3. **Indoklás.** A formátumnak tartalmaznia kell magyarázatokat és pontokat az eszköz által közölt eredmények megbízhatóságára vonatkozóan.
4. **Teljesség.** A formátumnak képesnek kell lennie a programrésztvevőket oly mértékben reprezentálni, ahogyan az a tervezési minták irodalmában szerepel.
5. **Szerepjátékosok azonosítása.** Minden egyes programrésztvevőt, amely egy tervezési minta szerepet játszik el, egyértelműen kell azonosítani.
6. **Jelöltek azonosítása.** Minden jelöltet egyértelműen kell azonosítani és csak egyszer lehet közölni.
7. **Összehasonlíthatóság.** A formátumnak meg kell engednie az eszközben megadott minta definíciók és az alkalmazott módszerek közlését ahhoz, hogy az eredmények összehasonlíthatóak legyenek.
8. **Nyelvfüggetlenség.** (opcionális) A közös csereformátumnak el kell vonatkoztatnia nyelvspecifikus fogalmaktól, így használható tetszőleges imperatív programozási nyelven írt programban azonosított jelöltek közlésére is.
9. **Szabványosság.** (opcionális) A formátum specifikációja a létező szabványokkal konzisztens kell legyen azért, hogy könnyen adaptálható, karbantartható és fejleszthető maradjon.

Metamodellek

Három metamodell közösen definiálja a DPDX formátumot: a tervezési minták sémáinak metamodellje, a programelem azonosítók metamodellje és a DPD eredmények metamodellje. A séma metamodell lehetővé teszi azt, hogy az eszközök közöljék a keresett mintákat, azok sémáit; a programelem metamodell teszi lehetővé, hogy az eszközök azonosítsák azokat a programelemeket a forráskódban,

melyek valamilyen szerepet játszanak egy mintában; valamint az eredmény metamodell írja le az azonosított tervezési minta jelölteket.

A modellek kapcsolatát a 4. ábra mutatja be, ahol a eredmények az eredmény metamodell példányai. A legfontosabb rész a sémában megadott szerepek és kapcsolatok megfeleltetése a programelemekkel (a programelem modellben). Egy jelölt lényegében egy ilyen megfeleltetés. Megjegyezzük azt, hogy a jelöltek átfedhetik egymást; vagyis bizonyos programelemek különböző minta sémákban is eljátszhatnak szerepeket, ahogyan az egyik Singleton jelölt átfedi az egyik Decorator jelöltet a 4. ábrán.



4. ábra. Kapcsolat a séma, az eredmények és a példányok között

Implementáció

A hosszútávú karbantarthatóság miatt a metamodellek implementációja amennyire csak lehet szabványos kell hogy legyen. Emiatt a közös csereformátumot XML-re alapoztuk. A DPDX implementációja a három metamodell megvalósítását tartalmazza. A következő általános alapelvekhez ragaszkodtunk a metamodellek és az XML formátum közötti leképezés során azért, hogy az implementáció a lehető legegyszerűbb maradjon: (1) a metamodell osztályai az XML tageknek felelnek meg; (2) a metamodell elemek attribútumai az XML tagek attribútumainak felelnek meg; (3) a metamodell elemei közötti aggregációkat az XML szülő-gyerek beágyazási technikája reprezentálja; (4) egy hivatkozható elem rendelkezik egy 'id' attribútummal, és az az elem amelyik szeretne erre az elemre hivatkozni, egy attribútummal teheti meg; (5) az egynél nagyobb multiplicitású asszociáció egy csoportosító elemben megadott hivatkozó elemekkel kerül reprezentálásra.

Saját hozzájárulás

A szerző fejlesztette ki a séma metamodell kezdeti implementációját és bemutatta a Maisa eszközt. Részt vett a kezdeti ötletek (alapvető elképzelések, metamodell, implementáció) lényeges továbbfejlesztésében és véglegesítésében is.

5. Visszatervező eszközök validációja

A DEEBEE rendszert továbbfejlesztettük a kiértékelési szempontok és a kiértékelendő adatok általánosításával, ezzel segítve azt, hogy szélesebb körben is alkalmazható legyen. Az új rendszer a BEFRIEND (BEnchmark For Reverse engInEering tools workiNg on source coDe). A BEFRIEND öt szempontban jelentősen eltér az elődjétől (DEEBEE).

1. **Területek (domains).** A DEEBEE a tervezési minta kereső eszközök kiértékelését és összehasonlítását támogatja. A BEFRIEND-el a forráskód tetszőleges tulajdonságait azonosító visszatervező eszközök eredményei értékelhetőek ki és hasonlíthatóak össze egymással. Ilyen eszközök a tervezési minta keresők, a duplikált kód azonosítók és a szabálysértés ellenőrzők. A BEFRIEND-ben a felhasználó képes beállítani az aktív területet (domaint), amelyre minden további művelet (pl. jelöltek listázása) vonatkozik.

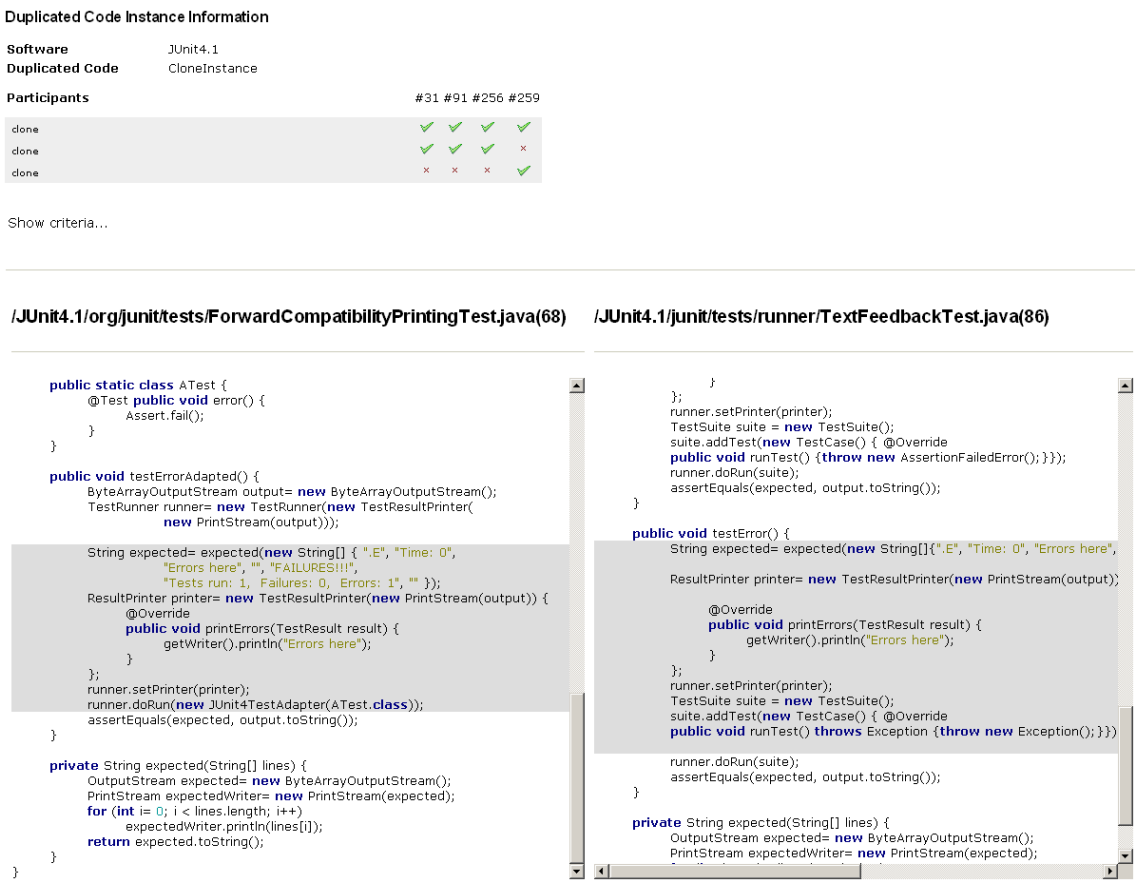
2. **Kiértékelési szempontok.** A DEEBEE előre rögzített kiértékelési szempontokkal rendelkezik, viszont a BEFRIEND lehetővé teszi az eredmények kiértékelési szempontjainak tetszőleges felvételét és törlését. A feltöltött jelölteket ezek alapján lehet kiértékelni. Egy kiértékelési szempontnál meg kell adni egy kérdést, amelyre tetszőleges számú válasz definiálható. Minden egyes válaszra be kell állítani egy százalékos arányt, amely azt mutatja, hogy az adott kérdést milyen mértékben válaszolták meg. A felhasználók visszajelzéseire alapozva a benchmark ezeket az arányszámokat felhasználva képes különböző statisztikákat számolni.

3. **Felhasználói felület.** A DEEBEE felhasználói felületét is továbbfejlesztettük a BEFRIEND kialakításakor. Például, az instance view csak egy kódrészletet mutat DEEBEEben, míg a BEFRIEND két kódrészletet jelenít meg (lsd. 5. ábra). Ez a bővítés többek között egyszerűsíti a duplikált kód kereső eszközök által azonosított másolt kódrészletek összehasonlítását.
4. **Csoportosító mechanizmus.** A BEFRIEND általánosítja a csoportosító mechanizmust és a testvér kapcsolatokat az egyéb területek problémáinak kezelése miatt. Például a duplikált kód kereső eszközök számára a meghatározó résztvevők nem használhatóak fel a csoportosítás alapjaként úgy, mint a DEEBEE esetében. A BEFRIEND-ben három tényező határozza meg a testvér kapcsolatot két jelölt között. Ezek a forráskód pozíciók megfeleltetése, az egyező résztvevők minimális száma, és a terület függő névillesztés. A testvér relációk beállításai pedig minden egyes területhez külön állíthatóak.

5. **Plug-in architektúra.** A DEEBEE-nek egy speciális CSV feltöltési formátuma van, egy tervezési minta kereső eszköz kimenetét erre a formátumra kell konvertálni. A BEFRIEND viszont lehetővé teszi különböző formátumok feltöltését egy plug-in architektúra bevezetésével. A megfelelő plug-in implementálásával biztosítható egy új eszköz eredményeinek feltöltése.

Kísérletek

A BEFRIEND-et három visszatervező területen alkalmaztuk: tervezési minta keresők, duplikált kód detektorok és kódolási szabálysértés azonosító eszközök esetén. A duplikált kód terület esetén végeztünk kísérleteket a benchmarkkal. Öt duplikált kód kereső eszközt értékeltünk ki két különböző nyílt forráskódú rendszeren, a JUnit-on és a NotePad++-on. A kiértékeléshez három kiértékelési szempontot használtunk. A *Correctness* (helyesség) kritériumot arra használtuk, hogy eldöntsük, hogy egy



5. ábra. Group instance nézet

klón csoport milyen mértékben tartalmaz másolt kódrészleteket; a pontosság (precision) és teljesség (recall) pontok ezen a kritériumon alapultak. A második szempont a *Procedure abstraction* volt, a kapcsolódó kérdés pedig: *'Megéri-e a duplikált kódrészleteket kicserélni egy új függvénnyel és függvényhívásokkal?*. A harmadik kritérium a *Gain* volt, a kapcsolódó kérdés pedig: *'Mekkora a becsült haszon a függvények bevezetésének?'*. A JUnit esetében kapott eredményeket mutatja a 3. táblázat.

Eszköz	Bauhaus clones	CCFinder	Columbus	PMD	Simian
Precision	62.79%	54.84%	100.0%	100.0%	100.0%
Recall	84.38%	53.13%	12.5%	15.63%	6.25%
Proc. abstr.	48.31%	44.23%	79.0%	73.0%	66.25%
Gain	29.36%	30.98%	62.5%	62.5%	62.5%

3. táblázat. Eredmények a JUnit rendszeren

Saját hozzájárulás

A szerző adaptálta és általánosította a testvér kapcsolatok elméletét és megadta a kapcsolódó implementációt. Továbbá részt vett a terminológia lefektetésében, az eredmények benchmarkban történő kézi kiértékelésében, valamint a benchmark architektúrájának bemutatásában.

Összefoglalás

Jelen munka fő hozzájárulásai a következőkképpen foglalhatóak össze. Először gépi tanuló algoritmusokat alkalmaztunk a tervezési minta kereső eszközünk eredményeinek pontosításához. Ebben a tanulmányban megmutattuk, hogy egy gépi tanuláson alapuló módszer sikeresen alkalmazható egy visszatervező eszköz hamis találatainak kiszűrésére. Korábban számos érdekes megközelítést is publikáltak [32, 33, 16, 1] tervezési minta kereső eszközök eredményeinek javítására, de egyik sem próbálta az eredményeket úgy javítani, ahogyan azt mi tettük.

A második tanulmányban kísérleteink során három tervezési minta kereső eszközt értékeltünk ki és hasonlítottunk össze. A kísérletek közben az eszközök eredményei közötti gyakori eltéréseket is összegyűjtöttük, és megmutattuk, hogy mely eszköz használható adott körülmények között a sebességet illetve a memóriaigényt tekintve.

A második tanulmány eredményeire alapozva kifejlesztettük a DEEBEE-t, egy benchmarkot tervezési minta eszközök kiértékelésére és összehasonlítására, és elvégeztünk néhány kísérletet. A benchmark általános, nyelv-, szoftver-, eszköz- és mintafüggetlen. A benchmarkkal az eszközök pontossága bárki által validálható. Korábban számos tanulmányt publikáltak [24, 15, 11, 2, 14] a tervezési minta kereső eszközök kiértékeléséről, összehasonlításáról, áttekintéséről, de egy olyan benchmark, mint a DEEBEE korábban nem létezett (pl. a jelöltek automatikus csoportosításával).

Szintén bevezettünk a tervezési minta kereső eszközök számára egy XML alapú közös formátumot (DPDX). A javasolt formátum egy jól definiált és bővíthető metamodellen alapul, amely a korábbi tervezési minta kereső eszközök formátumainak hiányosságait pótolja. Ez a formátum segítheti a különböző tervezési minta kereső eszközök kimeneteinek összehasonlítását [35], fúzióját [20], vizualizációját [10], és validációját [37].

Végül, de nem utolsó sorban kifejlesztettük a BEFRIENDet, egy benchmarkot, amely visszatervező eszközök kiértékelésére és összehasonlítására használható. A BEFRIEND-et három visszatervező területre alkalmaztuk: tervezési minta kereső eszközökre, duplikált kód kereső eszközökre, és kódolási szabálysértéseket azonosító eszközökre. A duplikált kód területen kísérleteket is elvégeztünk a benchmarkkal öt duplikált kód kereső eszközt felhasználva. Számos cikket publikáltak [5, 27, 8, 31, 3, 26] a különböző visszatervező eszközök kiértékeléséről és összehasonlításáról de egy olyan általános benchmark, mint a BEFRIEND korábban nem létezett.

A 4. táblázat összefoglalja a tézis eredményeit és a kapcsolódó publikációkat.

$\mathcal{N}o.$	[34]	[35]	[36]	[37]	[38]	[39]	[40]	[41]	[42]
1.	•								
2.		•							
3.			•	•	•				
4.						•	•		
5.								•	•

4. táblázat. A tézispontok és a kapcsolódó publikációk közötti kapcsolat.

Köszönetnyílvánítás

Mindenek előtt szeretném megköszönni témavezetőmnek, Dr. Gyimóthy Tibornak a számos ötletet és tanácsot amivel munkámat segítette, valamint az érdekes kutatási irányokat melyeket számomra elérhetővé tett. Szeretném megköszönni szerzőtársam és mentorom, Dr. Ferenc Rudolf segítségét is, aki irányította a munkámat és számos nélkülözhetetlen és alapvető dolgot tanított meg a kutatásról. Értékes tanácsai és segítsége nélkül nem tudtam volna kutatói szemléletet elsajátítani. További köszönettel tartozom kollegáimnak és társszerzőimnek, Dr. Beszédes Árpádnak, Bakota Tibornak, Dr. Siket Istvánnak, Dr. Jász Juditnak, Siket Péternek, Hegedűs Péternek, Dr. Schrettner Lajosnak, Dr. Gergely Tamásnak, Dr. Vidács Lászlónak, Hegedűs Györgynek, Dr. Günter Knieselnek, Alexander Binunnak, Dr. Alexander Chatzigeorgiounak, Dr. Yann-Gaël Guéhéneucnek, Dr. Nikolaos Tsantalisnak, Kakuja-Tóth Gabriellának, Demeter Hunornak, Nagy Csabának, Fischer Ferencnek, Ilia Árpádnak, Végh Ádám Zoltánnak, Rácz Róbertnek, Farkas Lórántnak, Szokody Fedornak, Sógor Zoltánnak, Lóki Gábornak, Lele Jánosnak, Gyovai Tamásnak, Horváth Tibornak és Pánczél Jánosnak. Szintén szeretnén megköszönni a cikkeim anonim bírálóinak az értékes megjegyzéseiket és javaslataikat. Továbbá köszönöm David P. Curleynek a munka nyelvi helyességének ellenőrzését és javítását. Szeretném megköszönni édesanyámnak folyamatos támogatását és bátorítását. Végül, de nem utolsó sorban, szívből köszönöm feleségemnek, Mártinak, a nélkülözhetetlen, szeretetteljes és támogató hátteret.

Fülöp Lajos Jenő, 2011

Hivatkozások

[1] Giuliano Antoniol, Roberto Fiutem, and L. Cristoforetti. Using Metrics to Identify Design Patterns in Object-Oriented Software. In *Proceedings of the Fifth International Symposium on Software Metrics (METRICS98)*, pages 23–34. IEEE Computer Society, November 1998.

[2] Francesca Arcelli, Stefano Masiero, Claudia Raibulet, and Francesco Tisato. A Comparison of Reverse Engineering Tools based on Design Pattern Decomposition. In *Proceedings of the 15th Australian Software Engineering Conference (ASWEC'05)*, pages 677–691. IEEE Computer Society, February 2005.

[3] Nathaniel Ayewah, William Pugh, J. David Morgenthaler, John Penix, and YuQian Zhou. Evaluating static analysis defect warnings on production software. In *PASTE '07: Proceedings of the 7th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*, pages 1–8. ACM, 2007.

[4] Zsolt Balanyi and Rudolf Ferenc. Mining Design Patterns from C++ Source Code. In *Proceedings of the 19th International Conference on Software Maintenance (ICSM 2003)*, pages 305–314. IEEE Computer Society, September 2003.

[5] Stefan Bellon, Rainer Koschke, Giuliano Antoniol, Jens Krinke, and Ettore Merlo. Comparison and Evaluation of Clone Detection Tools. In *IEEE Transactions on Software Engineering*, Volume 33, pages 577–591, September 2007.

[6] Árpád Beszédes, Rudolf Ferenc, and Tibor Gyimóthy. Columbus: A Reverse Engineering Approach. In *Proceedings of the 13th IEEE Workshop on Software Technology and Engineering Practice (STEP 2005)*, pages 60–69. IEEE Computer Society, September 2005.

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

[8] Elizabeth Burd and John Bailey. Evaluating Clone Detection Tools for Use during Preventative Maintenance. In *Proceedings of the 2th International Workshop on Source Code Analysis and Manipulation (SCAM 2002)*, pages 36–43. IEEE Computer Society, 2002.

[9] Serge Demeyer, Stéphane Ducasse, and Oscar Nierstrasz. Object-Oriented Reengineering Patterns. Square Bracket Associates, 2008.

[10] Jing Dong, Sheng Yang, and Kang Zhang. Visualizing Design Patterns in Their Applications and Compositions. *IEEE Trans. Softw. Eng.*, 33:433–453, July 2007.

[11] Jing Dong, Yajing Zhao, and Tu Peng. A Review of Design Pattern Mining Techniques. *the International Journal of Software Engineering and Knowledge Engineering (IJSEKE)*, pages 823–855, 2008.

[12] Rudolf Ferenc, Juha Gustafsson, László Müller, and Jukka Paakki. Recognizing Design Patterns in C++ programs with the integration of Columbus and Maisa. *Acta Cybernetica*, 15:669–682, 2002.

[13] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Pub Co, 1995.

[14] Yann-Gaël Guéhéneuc. P-MARt: Pattern-like Micro Architecture Repository. `http://www-etud.iro.umontreal.ca/~ptidej/yann-gael/Work/Publications/Documents/EuroPLoP07PRa.doc.pdf` .

[15] Yann-Gaël Guéhéneuc, Kim Mens, and Roel Wuyts. A Comparative Framework for Design Recovery Tools. In *Proceedings of the 10th Conference on Software Maintenance and Reengineering (CSMR'06)*, pages 123–134. IEEE Computer Society, March 2006.

[16] Yann-Gaël Guéhéneuc, Houari Sahraoui, and Farouk Zaidi. Fingerprinting Design Patterns. In *Proceedings of the 11th Working Conference on Reverse Engineering (WCRE 2004)*, pages 172–181. IEEE Computer Society, 2004.

[17] The JHotDraw Homepage.
`http:/www.jhotdraw.org`.

[18] The JRefactory Homepage.
`http:/jrefactory.sourceforge.net/`.

[19] The JUnit Homepage.
`http:/www.junit.org`.

[20] Günter Kniesel and Alexander Binun. Standing on the Shoulders of Giants – A Data Fusion Approach to Design Pattern Detection. In *17th IEEE International Conference on Program Comprehension (ICPC'09)*. IEEE Computer Society, 2009.

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

[22] The NotePad++ Homepage.
`http://notepad-plus.sourceforge.net/`.

[23] J. Paakki, A. Karhinen, J. Gustafsson, L. Nenonen, and A.I. Verkamo. Software Metrics by Architectural Pattern Mining. In *Proceedings of the International Conference on Software: Theory and Practice (16th IFIP World Computer Congress)*, pages 325–332, 2000.

[24] Niklas Pettersson, Welf Löwe, and Joakim Nivre. On Evaluation of Accuracy in Pattern Detection. In *First International Workshop on Design Pattern Detection for Reverse Engineering (DPD4RE'06)*, October 2006.

[25] John Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[26] Nick Rutar, Christian B. Almazan, and Jeffrey S. Foster. A Comparison of Bug Finding Tools for Java. In *ISSRE '04: Proceedings of the 15th International Symposium on Software Reliability Engineering*, pages 245–256. IEEE Computer Society, 2004.

[27] Filip Van Rysselberghe and Serge Demeyer. Evaluating Clone Detection Techniques from a Refactoring Perspective. In *19th International Conference on Automated Software Engineering (ASE'04)*, pages 336–339. IEEE Computer Society, 2004.

[28] The StarOffice Homepage.
`http://www.sun.com/software/star`.

[29] The source code of FormulaManager.
`http://www.sed.hu/src/FormulaManager/`.

[30] Nikolaos Tsantalis, Alexander Chatzigeorgiou, George Stephanides, and Spyros T. Halkidis. Design Pattern Detection Using Similarity Scoring. In *IEEE Transactions on Software Engineering*, Volume 32, pages 896–909, Nov 2006.

[31] Stefan Wagner, Jan Jurjens, Claudia Koller, and Peter Trischberger. Comparing Bug Finding Tools with Reviews and Tests. In *Proceedings of 17th International Conference on Testing of Communicating Systems (TestCom'05)*, pages 40–55. Springer, 2005.

[32] Lothar Wendehals. Improving Design Pattern Instance Recognition by Dynamic Analysis. In *Proceedings of the ICSE 2003 Workshop on Dynamic Analysis (WODA), Portland, USA*, May 2003.

[33] Lothar Wendehals. Specifying Patterns for Dynamic Pattern Instance Recognition with UML 2.0 Sequence Diagrams. In *Proceedings of the 6th Workshop Software Reengineering (WSR2004)*, pages 63–64, May 2004.

Felhasznált publikációk

[34] Rudolf Ferenc, Árpád Beszédes, Lajos Fülöp, and János Lele. Design Pattern Mining Enhanced by Machine Learning. In *Proceedings of the 21th International Conference on Software Maintenance (ICSM 2005)*, pages 295–304. IEEE Computer Society, September 2005.

[35] Lajos Jenő Fülöp, Tamás Gyovai, and Rudolf Ferenc. Evaluating C++ Design Pattern Miner Tools. In *Proceedings of the 6th International Workshop on Source Code Analysis and Manipulation (SCAM 2006)*, pages 127–136. IEEE Computer Society, September 2006.

[36] Lajos Jenő Fülöp, Árpád Ilia, Ádám Zoltán Végh, and Rudolf Ferenc. Comparing and Evaluating Design Pattern Miner Tools. In *Proceedings of the 10th Symposium on Programming Languages and Software Tools (SPLST 2007)*, pages 372–386. Eötvös Loránd University, Faculty of Informatics, June 2007.

[37] Lajos Jenő Fülöp, Rudolf Ferenc, and Tibor Gyimóthy. Towards a Benchmark for Evaluating Design Pattern Miner Tools. In *Proceedings of the 12th European Conference on Software Maintenance and Reengineering (CSMR 2008)*, pages 143–152. IEEE Computer Society, April 2008.

[38] Lajos Jenő Fülöp, Árpád Ilia, Ádám Zoltán Végh, Péter Hegedűs, and Rudolf Ferenc. Comparing and Evaluating Design Pattern Miner Tools. *Journal of ANNALES Universitatis Scientiarum Budapestinensis de Rolando Eötvös Nominatae, Sectio Computatorica*, 31:167–184, 2009. Department of Computer Algebra, Eötvös Loránd University.

[39] Günter Kniesel, Alexander Binun, Péter Hegedűs, Lajos Jenő Fülöp, Alexander Chatzigeorgiou, Yann-Gaël Guéhéneuc, and Nikolaos Tsantalis. DPDX – A Common Exchange Format for Design Pattern Detection Tools. In *Proceedings of the 14th European Conference on Software Maintenance and Reengineering (CSMR 2010)*, pages 232–235. IEEE Computer Society, March 2010.

[40] Günter Kniesel, Alexander Binun, Péter Hegedűs, Lajos Jenő Fülöp, Alexander Chatzigeorgiou, Yann-Gaël Guéhéneuc, and Nikolaos Tsantalis. A common exchange format for design pattern detection tools. Technical report IAI-TR-2009-03, ISSN 0944-8535, CS Department III, University of Bonn, Germany, October 2009.

[41] Lajos Jenő Fülöp, Péter Hegedűs, Rudolf Ferenc, and Tibor Gyimóthy. Towards a Benchmark for Evaluating Reverse Engineering Tools. In *Tool Demonstrations of the 15th Working Conference on Reverse Engineering (WCRE 2008)*, pages 335–336. IEEE Computer Society, October 2008.

[42] Lajos Jenő Fülöp, Péter Hegedűs, and Rudolf Ferenc. BEFRIEND - a Benchmark for Evaluating Reverse Engineering Tools. *Journal of Periodica Polytechnica, Electrical Engineering*, 52/3-4:153–162, 2008. Budapest University of Technology and Economics.