UNIVERSITATEA BABEŞ-BOLYAI
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ

# Dezvoltare de soft bazată pe instruire automată

**Rezumat Extins**

Doctorand: Zsuzsanna-Edit Marian
Conducător de doctorat: Prof. Dr. Gabriela Czibula

Septembrie 2014

# Lista publicaţiilor

## Publicaţii indexate în ISI Web of Knowledge

### Publicaţii indexate în ISI Science Citation Index Expanded

1. [CMC14b] Gabriela Czibula, **Zsuzsanna Marian** şi Istvan-Gergely Czibula, Software defect prediction using relational association rule mining. *Information Sciences* publicat de *Elsevier*, Vol. 264, pp. 260-278, DOI: 10.1016/j.ins.2013.12.031, 2014. (**IF = 3.643**)

2. [CMC14a] Gabriela Czibula, **Zsuzsanna Marian** şi Istvan-Gergely Czibula. Detecting software design defects using relational association rule mining. *Knowledge and Information Systems* publicat de *Springer*, DOI: 10.1007/s10115-013-0721-z, publicat on-line în Ianuarie, 2014. (**IF = 2.225**)

3. [MCC12] **Zsuzsanna Marian**, Gabriela Czibula şi Istvan-Gergely Czibula. Using software metrics for automatic software design improvement. *SIC Journal, Studies in Informatics and Control*, România, Vol. 21, Number 3, pp. 249-258, 2012. (**IF = 0.578**)

4. [MCC14] **Zsuzsanna Marian** Gabriela Czibula şi Istvan-Gergely Czibula, Software packages refactoring using a hierarchical clustering-based approach. *Fundamenta Informaticae*, În curs de recenzie, 2014. (**IF = 0.399**)

### Publicaţii indexate în ISI Conference Proceedings Citation Index

5. [SMV09] Adrian Sterca, **Zsuzsanna Marian** şi Alexandru Vancea. Distortion-based media-friendly congestion control. *Proceedings of the International Conference on Knowledge Engineering, Principles and Techniques*, Cluj-Napoca, România, pp. 265-267, 2009.

6. [ŢIM09] Radu Ţurcaş, Oana Iova şi **Zsuzsanna Marian**. The autonomous robotic tank (ART): an innovative lego mindstorm NXT battle vehicle. *Proceedings of the International Conference on Knowledge Engineering, Principles and Techniques*, Cluj-Napoca, România, pp. 95-98, 2009.

## Articole publicate în jurnale internaţionale şi la conferinţe internaţionale

7. [Mar12a] **Zsuzsanna Marian**. Aggregated metrics guided software restructuring. *Proceedings of 8th IEEE International Conference on Intelligent Computer Communication and Processing*, Cluj-Napoca, România, pp. 259-266, 2012. (**Indexat IEEE**)

8. [Mar12c] **Zsuzsanna Marian**. A study on hierarchical clustering based software restructuring. *Studia Universitatis "Babes-Bolyai", Informatica*, România, Vol. LVII, Number 2, pp. 20-31, 2012. (**Indexat MathSciNet**)

9. [Mar13b] **Zsuzsanna Marian**. A study on Relational Association Rule Mining Based Software Design Defect Detection. *Studia Universitatis "Babes-Bolyai", Informatica*, România, Vol. LVIII, Number 1, pp. 42-57, 2013. (**Indexat MathSciNet**)

10. [Mar13a] **Zsuzsanna Marian**. On the software metrics influence in Relational Association Rule-based Software Defect Prediction. *Studia Universitatis "Babes-Bolyai", Informatica*, România, Vol. LVIII, Number 4, pp. 35-48, 2013. (**Indexat MathSciNet**)

11. [Mar14b] **Zsuzsanna Marian**. On evaluating the structure of software packages. *Studia Universitatis "Babes-Bolyai", Informatica*, România, Vol. LIX, Number 1, pp. 46-58, 2014. (**Indexat MathSciNet**)

12. [Mar14a] **Zsuzsanna Marian**. FAOS - A framework for analyzing object-oriented software systems. *Studia Universitatis "Babes-Bolyai", Informatica*, România, Under review, 2014. (**Indexat MathSciNet**)

13. [MS10] **Zsuzsanna Marian** şi Christian Săcărea. Using contextual topology to discover similarities in modern music. *Proceedings of the IEEE International Conference on Automation Quality and Testing, Robotics*, Cluj-Napoca, România, vol. 3, pp. 1–6, 2010. (**Indexat IEEE**)

14. [TLM11] Doina Tatar, Mihaiela Lupea şi **Zsuzsanna Marian**. Text summarization by formal concept analysis approach. *Studia Universitatis "Babes-Bolyai", Informatica*, Romaânia, Vol. LVI, Number 2, pp. 7-12, 2011. (**Indexat MathSciNet**)

15. [MCB11] **Zsuzsanna Marian**, Cosmin Coman şi Attila Bartha. Learning to play the guessing game. *Studia Universitatis "Babes-Bolyai", Informatica*, România, Vol. LVI, Number 2, pp. 119–124, 2011. (**Indexat MathSciNet**)

# Articole publicate la conferinţe naţionale

16. [Mar12b] **Zsuzsanna Marian**. Software metrics based refactoring: a case study. *Proceedings of the National Symposium ZAC 2012*, Cluj-Napoca, pp. 59-64, 2012.

17. [Mar10] **Zsuzsanna Marian**. Solving the subset sum problem with DNA computation, *Proceedings of the National Symposium ZAC 2010*, Cluj-Napoca, pp. 25-29, 2010.

În cazul publicaţiilor cu mai mulţi autori, am contribuit la: conceperea, proiectarea şi execuţia experimentelor; analiza datelor; dezvoltarea structurii şi argumentelor din articole; scrierea manuscrisului; revizuirea articolelor şi aprobarea variantei finale.

# Cuvinte cheie

- Instruire Automată
- Clustering
- Metrici Soft
- Reguli de Asociere Relaţionale
- Dezvoltare de Soft
- Restructurare Soft
- Detectarea Defectelor în Sisteme Informatice
- Detectarea Defectelor de Proiectare
- Predicţia Defectelor
- Seturi de date Nasa
- Framework-uri Soft

# Cuprins

# Introducere

Această teză este rezultatul cercetării mele în domeniul dezvoltării de *modele de instruire automată pentru rezolvarea diferitor probleme legate de dezvoltare de soft*. Această cercetare a început în 2011, sub coordonarea lui Prof. Dr. Gabriela Czibula.

Direcţia principală de cercetare aspura căreia m-am concentrat a fost aplicarea diferitor modele şi algoritmi de instruire automată pentru probleme din ingineria soft. Problemele abordate în această teză sunt de o importanţă majoră pentru întreţinerea şi evoluţia sistemelor informatice. Soluţii la aceste probleme ar ajuta programatorii în diferite probleme de dezvoltare a sistemelor informatice.

Instruirea Automată este o ramură a Inteligenţei Artificiale care incearcă să găsească un răspuns la întrebarea "Cum putem construi sisteme informatice care se îmbunătăţesc automat prin experienţă, şi care sunt legile fundamentale care conduc toate procesele de învăţare?" [Mit06]. Prin urmare, domeniul de Instruire Automată este alcătuit dintr-o serie de abordări şi algoritmi care pot învăţa din date existente.

Pornind de la articolul publicat de Mark Harman şi Brian Jones în anul 2001 s-a născut un nou domeniu de cercetare, numit *Search Based Software Engineering - SBSE* (Inginerie Soft Bazată pe Căutare) [HJ01]. Acest articol descrie că sunt multe probleme de inginerie soft care pot fi reformulate ca şi probleme de căutare, iar apoi diferiţi algoritmi de căutare pot fi aplicaţi asupra lor. Dezvoltarea acestui nou domeniu a dus la noi abordări inteligente pentru probleme de inginerie soft: există probleme de inginerie soft care pot fi reformulate ca probleme de *clustering* asupra cărora pot fi aplicaţi algoritmi de *clustering*. De asemenea, există probleme de inginerie soft care pot fi reformulate ca probleme de clasificare sau predicţie asupra cărora pot fi aplicati diferiţi clasificatori.

Dar chiar dacă metode inteligente *pot* fi aplicate asupra problemelor de inginerie soft, *de ce* ar trebui să fie aplicate? De ce ar trebui să dezvoltăm abordări pentru rezolvarea problemelor care pot fi rezolvate de dezvoltatori soft? Răspunsul este că aceste probleme nu tot timpul pot fi rezolvate de dezvoltatori soft şi ei pot folosi ajutorul acestor abordări. Din cauza complexităţii şi dimensiunii sistemelor informatice, sunt multe situaţii când dezvoltatorii unui sistem nu au o privire de ansamblu asupra sistemului, nu înţeleg toate legăturiile între elementele sistemului şi nu văd care sunt părţile care ar trebui îmbunătăţite sau care are trebui testate în mod intensiv. De aceea metode bazate pe căutare, pe inteligenţă computaţională şi pe instruire automată sunt binevenite: ele pot ajuta dezvoltatorii soft în munca lor.

Dintre diferitele probleme de dezvoltare soft asupra cărora au fost aplicate metode inteligente, am ales două direcţii principale asupra cărora m-am concentrat în această teză. Prima direcţie a fost dezvoltarea abordărilor pentru restructurarea sistemelor informatice orientate obiect folosind algoritmi de *clustering*. Am lucrat atât la restructurare la nivel de pachete, cât şi la restructurare la nivel de clase. A doua direcţie principală abordată în această teză este problema detectării defectelor în sisteme informatice. Această direcţie poate fi împărţită în detectarea defectelor de proiectare (*design defect detection*) şi predicţia defectelor (*defect prediction*), şi pentru fiecare am propus o nouă metodă bazată pe reguli de asociere relaţionale. O contribuţie originală spre dezvoltarea sistemelor informatice este prezentată de asemenea, în forma unui *framework*, numit *FAOS* (*Framework for Analyzing Object-oriented Software systems*). Acest *framework* a fost dezvoltat pentru a sprijini evaluarea experimentală a ma-

joritătţii abordărilor prezentate în această teză.

Acestă teză este alcătuită din patru capitole, după cum urmează.

Primul capitol, **Instruire Automată în Inginerie Soft. Context General**, începe cu o prezentare scurtă a diferitor metode de inteligenţă computaţională în ingineria soft. Aceasta este urmată de prezentarea celor două direcţii de cercetare abordate în această teză: *Remodularizare Soft* şi *Detectarea Defectelor în Sisteme Informatice*. Pentru ambele direcţii problemele abordate sunt detaliate împreună cu o scurtă prezentare a metodelor existente în literatura de specialitate. În cele din urmă este o scurtă descriere a celor două metode de instruire automată care au fost folosite în abordăriile noastre: reguli de asociere relaţionale şi *clustering*.

Contribuţiile originale sunt prezentate în Capitolele 2, 3 şi 4, unde descriem modelele de instruire automată propuse pentru rezolvarea problemelor de inginerie soft ce fac studiul acestei teze.

Capitolul 2, **Abordări noi pentru Remodularizare Soft**, este original şi prezintă abordările noastre în cadrul primei direcţii de cercetare principală a acestei teze, şi anume restructurarea sistemelor informatice folosind algoritmi de *clustering*. Prezentarea începe cu prima problemă din acest domeniu: remodularizarea la nivel de pachete. Prezentăm doi algoritmi, unul care identifică o împărţire bună a claselor în pachete şi unul care, plecând de la o structură existentă de pachete, poate să identifice pachetul potrivit pentru o clasă care urmează să fie adăugată în sistem. Aceasta este urmată de evaluarea experimentală a acestor algoritmi, o analiză a abordării şi o comparaţie cu alte metode existente în literatura de specialitate pentru restructurare la nivel de pachete. A doua problemă abordată în Capitolul 2, restructurare soft la nivel de clasă, este prezentată în continuare. Prezentăm trei algoritmi care pot identifica o structură de clasă îmbunătăţită pentru un sistem informatic, împreună cu evaluarea lor experimentală. În primul rând se face o comparaţie a celor trei algoritmi urmând ca aceştia să fie comparaţi cu alţi algoritmi prezentaţi în literatura de specialitate. Ultima secţiune prezintă concluziile din capitol şi direcţii pentru cercetări ulterioare.

Capitolul 3, **Abordări noi pentru Detectarea Defectelor în Sisteme Informatice**, este original şi prezintă abordăriile mele pentru a doua direcţie principală de cercetare a acestei teze, detectarea defectelor în sisteme informatice. Ambele probleme abordate din această direcţie folosesc acelaşi model teoretic, care este prezentat în prima secţiune. Aceasta este urmată de introducerea abordării noastre pentru identificarea claselor cu defecte de proiectare folosind reguli de asociere relaţionale. După aceea este prezentată evaluarea experimentală a abordării şi o comparaţie cu metode existente. În continuare se prezintă un studiu despre efectele modificării valorilor parametrilor asupra abordării propuse în teză. A doua parte a Capitolului 3 prezintă abordarea noastră pentru a doua problemă din această direcţie, şi anume, predicţia defectelor în sisteme informatice. Descriem abordarea noastră originală, un clasificator care poate clasifica entităţile dintr-un sistem soft ca defective sau non-defective folosind reguli de asociere relaţionale. Urmează evaluarea experimentală şi o comparaţie cu abordări existente. La sfârşit prezentăm concluziile capitolului.

Capitolul 4, **Un framework pentru Analiza Sistemelor Informatice Orientate Obiect**, este original şi prezintă *framework*-ul *FAOS* (*Framework for Analyzing Object-oriented Software Systems*), un *framework* general dezvoltat în limbajul de programare Java pentru analizarea sistemelor orientate obiect. Capitolul prezintă cele trei module principale ale *framework*-ului, unul pentru analiza codului Java compilat şi extragerea unei liste de elemente (clase, metode, atribute) şi relaţiile între ele, unul pentru calcularea valorilor diferitor metrici soft, şi unul pentru implementarea abordării noastre pentru restructurarea unui sistem informatic la nivel de pachete. O comparaţie scurtă cu alte *framework*-uri şi unelte similare este prezentată după cele trei module. Concluziile si direcţiile de cercetare ulterioare se găsesc în ultima secţiune.

Contribuţiile originale introduse în această teză se găsesc în Capitolele 2, 3 şi 4 şi sunt

următoarele:

- O abordare nouă bazată pe *clustering* pentru remodularizare soft la nivel de pachete - abordarea *CASP* (*Clustering Approach for Software Package Restructuring*) (Subsecţiunea 2.1.1) împreună cu un algoritm bazat pe *clustering* ierarhic, *HASP*(*Hierarchical Clustering Algorithms for Software Packages Restructuring*) care poate fi folosit în abordarea *CASP* (Subsecţiunea 2.1.2) [MCC14].
  - Definiţia a şapte caracteristici, care pot fi agregate într-o singură valoare, numită *overallScore*, care poate fi folosită ca măsură de similaritate în algoritmul *HASP* (Subsecţiunea 2.1.2) şi definiţia unei măsuri noi, *CIP* (*Cohesion of Identified Packages*), care măsoară cât de aproape este o partiţie de o altă partiţie (Subsecţiunea 2.1.4). [MCC14].
  - Algoritmul *AssignClass*, care poate găsi pachetul potrivit pentru o clasă nou adăugată, folosind aceeaşi caracteristici şi scor ca şi algoritmul *HASP* (Subsecţiunea 2.1.3) [MCC14].
  - Evaluarea experimentală a algoritmilor *HASP* şi *AssignClass* (Subsecţiunea 2.1.4), analiza rezultatelor (Subsecţiunea 2.1.5) şi o comparaţie a măsurii *overallScore* cu alte măsuri existente (Subsecţiunea 2.1.6). [MCC14, Mar14b].
- Trei algoritmi noi pentru restructurarea unui sistem informatic la nivel de clasă: *ARI*, *HAC* (Subsecţiunea 2.2.1) şi o abordare bazată pe metrici agregate (Subsecţiunea 2.2.2) [MCC12, Mar12c, Mar12b, Mar12a].
  - Evaluarea experimentală a celor trei algoritmi şi o analiză comparativă a lor (Subsecţiunea 2.2.3).
- O metodă pentru detectarea defectelor de proiectare bazată pe reguli de asociere relaţionale - *SDDRAR* (Subsecţiunea 3.2.2) [CMC14a].
  - Un algoritm nou, similar cu Apriori [AS94], pentru a extrage regulile de asociere relaţionale de orice lungime dintr-un set de date - *DRAR* (Subsecţinea 3.2.1) [CMC14a].
  - O evaluare experimentală detaliată a abordării *SDDRAR* (Subsecţiunea 3.2.3) urmată de o analiză a abordării din mai multe puncte de vedere (Subsecţiunea 3.2.4) [CMC14a].
  - Un studiu despre efectele modificării valorilor parametrilor asupra rezultatelor abordării *SDDRAR* (Subsecţiunea 3.2.5) [Mar13b].
- O abordare de clasificare nouă, care, folosind reguli de asociere relaţionale, poate să clasifice entităţile dintr-un sistem informatic ca fiind defective sau non-defective - *DPRAR* (Subsecţiunea 3.3.1) [CMC14b].
  - O evaluare experimentală amănunţită a abordării *DPRAR*, folosind 10 seturi de date *NASA* (Subsecţiunea 3.3.2) şi o comparaţie a rezultatelor cu cele raportate în literatura de specialitate pentru aceleaşi seturi de date (Subsecţiunea 3.3.3) [CMC14b].
  - Studii aspura efectului eliminării unor caracteristici şi a modificării formulei de calcul a scorului (Subsecţiunea 3.3.4) [Mar13a].
- Un *framework* general, *FAOS*, pentru analiza sistemelor informatice orientate obiect (Secţiunea 4.1) [Mar14a]. Acest *framework* a fost implementat pentru a sprijini evaluarea experimentală a abordărilor dezvoltate.

# Capitolul 1

# Instruire Automată în Inginerie Soft

Aplicarea metodelor inteligente asupra problemelor din inginerie soft este un domeniu de cercetare relativ nou, apărut în anul 2001 cu un articol publicat de Mark Harman şi Bryan Jones, [HJ01]. De atunci acest domeniu a progresat mult. În [ZT05] autorii prezintă 44 probleme diferite din ingineria soft pentru care cel puţin un algoritm de instruire automată a fost aplicat. În mod similar, o analiză a tendinţelor curente în *SBSE*, [HMZ09], a adunat peste 500 de publicaţii din domeniu.

## 1.1 Inteligenţa Computaţională în Inginerie Soft

În anul 2001 Mark Harman şi Bryan Jones au publicat un articol, [HJ01], care este considerat manifestul unui domeniu de cercetare nou, numit *Search-Based Software Engineering - SBSE* (*Inginerie Soft Bazată pe Căutare*). Aceştia arată că tehnici de căutare (şi anume Algoritmi Genetici, Căutare Tabu şi Călirea Simulată) au fost aplicaţi în diferite domenii de inginerie şi că ei ar fi potriviţi şi pentru inginerie soft.

În următoarea perioadă un număr mare de abordări au fost prezentate în literatura de specialitate, abordări care aplică diferiţi algoritmi de căutare pentru probleme de inginerie soft. La 8 ani după apariţia domeniului, în anul 2009, Harman et al. au publicat o analiză a curentelor, tehnicilor şi aplicaţiilor existente în *SBSE*, [HMZ09], luând în calcul peste 500 de publicaţii din domeniu.

Algoritmii de căutare nu sunt singurele metode inteligente aplicate pe probleme de inginerie soft, aşa cum este prezentat în [Har12], unde autorul menţionează trei domenii largi din Inteligenţa Artificială folosite de ingineri soft: căutare şi optimizare computaţională, metode probabilistice şi *fuzzy*, învăţare şi predicţie.

Pe lânga [Har12], o altă analiză recentă despre rolul Inteligenţei Artificiale în ingineria soft [AAH12] prezintă că metode de inteligenţă computaţională au fost aplicate pentru proiectarea arhitecturii sistemelor informatice, testare soft, estimarea şi predicţia costului şi repararea automată a bug-urilor. În [DK05] Dick şi Kandel prezintă exemple de cum pot fi aplicate metode din inteligenţa computaţională şi instruirea automată pentru asigurarea calităţii softului.

Şi algoritmi de instruire automată au fost aplicaţi pentru diferite probleme din ingineria soft. În [ZT05] autorii prezintă 44 probleme diferite din ingineria soft pentru care cel puţin un algoritm de instruire automată a fost aplicat.

## 1.2 Remodularizare Soft

Un sistem informatic trebuie să se adapteze cerinţelor care se modifică pe parcursul duratei de viaţă, altfel nu va mai fi folosit. Modificările legate de această adaptare sunt efectuate de cele mai multe ori după lansarea softului, la întreţinere, şi dacă sunt efectuate doar modificări care adaugă noi funcţionalităti sau corectează erorile existente, sistemul va deveni din ce în ce mai greu de întreţinut. Pentru a preveni acest lucru, o altă activitate importantă este restructurarea codului sursă. Restructurarea este activitatea de a schimba (îmbunătăţi) structura internă a sistemului soft, fără a afecta comportamentul exterior al sistemului.

Remodularizarea soft poate fi efectuată pe diferite nivele ale unui sistem informatic. Pentru un sistem orientat obiect exista remodularizare la nivel de metodă, clasă, pachet şi arhitectură.

### 1.2.1 Remodularizare la nivel de pachete

**Definiţia şi relevanţa problemei.** În zilele noastre sistemele informatice devin din ce în ce mai complexe, sunt alcătuite din mii de clase care sunt grupate în pachete. Fără o structură de pachete adecvată sistemul devine greu de întreţinut. Astfel, remodularizarea soft la nivel de pachete este un proces important în întreţinerea şi evoluţia sistemului. Cu cât e mai complex sistemul, cu atât sunt mai ridicate costurile întreţinerii. Astfel, este destul de greu pentru dezvoltatorii soft să aleagă structura de pachete potrivită pentru sistem. Când numărul de clase este mare, găsirea pachetului potrivit pentru o clasă nouă nu este simplă, necesită o bună cunoaştere a sistemului.

Problema de restructurare a pachetelor se iveşte din nevoi practice, astfel abordări inteligente pot fi de folos, ajutând dezvoltatorii în munca lor.

**Abordări existente în literatură.** Există câteva metode prezentate în literatura de specialitate pentru identificarea modului în care clasele ar trebui grupate în pachete. O astfel de metodă este cea prezentată în [AAM11], unde *clustering* este folosit pentru a găsi gruparea ideală a claselor. O altă metodă, bazată pe detectarea comunităţilor constrânse, este prezentată în [PJL13]. Chiar dacă nu poate restructura un sistem întreg, metoda prezentată de Bavota et al. în [BLMO10] poate să dividă un pachet cu coeziune mică în mai multe pachete mai coezive. Metode bazate pe algoritmi de căutare sunt prezentate în [MHH03, MMCG99].

O altă direcţie de cercetare care trebuie menţionată este definirea diferitor metrici, care măsoară calitatea pachetelor într-un sistem. Astfel de metrici sunt prezentate în [SKR08] şi [DABH11].

### 1.2.2 Remodularizare la nivel de clasă

**Definiţia şi relevanţa problemei.** Cele mai multe metode legate de remodularizare soft din literatura de specialitate identifică sau efectuază restructurare la nivel de clasă. Aceste abordări încearcă să identifice acele clase din sistemul informatic, care nu sunt bine proiectate, şi să identifice acele restructurări care ar îmbunătăţi proiectarea lor.

O carte excelentă şi des menţionată în legătură cu restructurare este *Refactoring: Improving the Design of Existing Code* de Martin Fowler [Fow99]. În această carte Fowler prezintă de ce este importantă restructurarea şi cum se efectuază. De asemenea, el prezintă un catalog cu 72 de restructurări diferite. Chiar şi cu un catalog atât de complet, există două întrebări: când şi unde ar trebui efectuate restructurări? Pentru a răspunde la această întrebare, Fowler introduce noţiunea de *bad smell*, care este o structură în codul sursa care sugerează posibilitatea unei restructurări [Fow99]. În carte este şi o listă cu 22 *bad smell*-uri.

**Abordări existente în literatură.** Identificarea oportunităţilor de restructurare în

sisteme informatice este un domeniu bine-studiat. Diferite metode sunt prezentate în literatura de specialitate care folosesc Teoria Jocurilor [BOL+10, HT07], *Concept Analysis* [ST98], Reţele Bayesiene [KVGS09], *clustering* [RR11, FTCS09, AL99] sau diferite metode de căutare [MC11, OC08, GWI11, SSB06, HT07]. Cu toate că aproape toate metodele enumerate mai sus folosesc şi metrici soft, există abordări bazate doar pe metrici soft [ISIE12, SS07, HKI08, CLMM05, SSL01, TK03, DDN00].

## 1.3 Detectarea Defectelor în Sisteme Informatice

Detectarea defectelor în sisteme informatice este ramura ingineriei soft care încearcă să identifice sau să prezică defectele într-un sistem informatic. Este o activitate importantă, conform [Kan03], detectarea şi corectarea defectelor pot consuma în jur de 75% din toate costurile ciclului de viaţă. În [Kan03] Kandt prezintă două tehnici complementare de detectarea defectelor: inspecţia şi testarea. Ambele necesită timp şi efort considerabil, de cele mai multe ori greu de obţinut, mai ales în cazul sistemelor mari. Această lipsă a timpului, combinată cu dimensiunea mare a sistemelor informatice este motivul pentru care sunt dezvoltate metode inteligente care să ajute dezvoltatorii prin sugerarea părţilor din sistem care necesită o testare mai intenzivă.

### 1.3.1 Detectarea Defectelor de Proiectare

**Definiţia şi relevanţa problemei.** Structura internă a entităţilor într-un sistem informatic se poate modifica de mai multe ori pe parcursul ciclului de viaţă şi are un impact important asupra mentenabilităţii sistemului. Chiar dacă un sistem informatic are o proiectare bună la început, pe parcursul întreţinerii proiectarea se degradează, ceea ce face întreţinerea mai complicată şi mai costisitoare pe viitor. De aceea monitorizarea continuă a proiectării şi identificarea şi corectarea defectelor de proiectare este esenţială.

Detectarea defectelor de proiectare este înrudită cu o altă problemă de inginerie soft, şi anume, restructurarea. În general, soluţia pentru a elimina defectele de proiectare este restructurarea sistemului.

**Ce sunt defectele de proiectare?** Pentru a putea detecta defectele de proiectare, trebuie să definim ce este un defect de proiectare. Pentru a defini defectele de proiectare putem pleca de la un set de principii pentru o proiectare corectă, şi căuta încălcarea acestor principii. De exemplu, Larman în [Lar04] prezintă nouă principii (sau şabloane), cunoscute sub numele de şabloane *GRASP* (*General Responsibility Assignment Software Patterns*).

Defectele de proiectare sunt adesea împărţite în două categorii: defecte de proiectare de nivel înalt şi defecte de proiectare de nivel scăzut (numite şi *bad smell*).

**Abordări existente în literatură.** Identificarea manuală a defectelor de proiectare durează foarte mult pentru sisteme mari, de aceea în literatura de specialitate diferite abordări sunt prezentate pentru a le găsi în mod automat. Asemenea abordări sunt *detection strategies* introduse de Marinescu [Mar02], *rule cards* introduse de Moha [MGL06, Moh06], *plugin*-ul pentru *Eclipse JDeodorant* [FTSC11] şi abordarea bazată pe metrici soft a lui Munro [Mun05]. Aplicarea metodelor de căutare pentru detectarea defectelor de proiectare este prezentată în [KSBW11].

Fontana et al. prezintă în [FBZ12] o comparaţie între 4 instrumente care detectează *code smell*-uri, care sunt aplicate pe diferite sisteme informatice cu scopul de a detecta şase tipuri de *bad smell*. Una dintre cele mai importante concluzii ale articolului este că instrumente diferite, aplicate pe acelaşi sistem pentru a identifica acelaşi *bad smell*, produc rezultate diferite (singura excepţie a fost la *bad smell God Class*).

### 1.3.2 Predicţia Defectelor

**Definiţia şi relevanţa problemei.** Predicţia defectelor încearcă să determine acele părţi ale unui sistem informatic unde se găsesc erori, astfel, testerii pot să petreacă mai mult timp testând acele componente care probabil conţin erori, şi mai puţin timp testând componentele care probabil sunt fără erori. Multe abordări folosesc metrici soft, pentru a măsura calitatea softului pentru a prezice defectele. Astfel, predicţia defectelor constă în clasificarea modulelor ca fiind defective şi non-defective, folosind o clasificare bazată pe metrici [BMW02]. Cele mai multe abordări folosesc date de la alte proiecte sau versiuni anterioare, astfel reprezintă abordări de clasificare supervizată.

**Seturile de date *NASA*.** Multe abordări din literatura de specialitate folosesc pentru evaluarea experimentală seturile de date *NASA*. Aceste seturi de date pot fi găsite la [Nas], unde ele sunt accesibile tuturor, pentru a facilita crearea modelelor predictive.

**Abordări existente în literatură.** Una dintre primele metodele de predicţie a defectelor este metoda *CBA*, prezentată în [LHM98], o metoda care foloseşte o extensie a regulilor de asociere. [LMW01] prezintă o extensie a acestei metode, *CBA2*. Un model hibrid, care combină reguli de asociere şi regresia logistică este prezentată în [KMMiM08]. O altă metodă bazată pe reguli este *EDER-SD* care produce doar reguli care caracterizează modulele defective [RRRAR12].

Pe lângă metode bazate pe reguli au fost aplicaţi şi alţi algoritmi din instriurea automată pentru predicţia defectelor. O asemenea lucrare este [MGF07], în care autorii evaluează clasificatorii OneR, Naive Bayes şi J48. Challagulla et al. evaluează în [CBYP05] nişte modele de predicţie pe patru seturi de date *NASA*. Haghighi et al. prezintă în [HDF12] o analiză comparativă a 37 de clasificatori şi folosesc seturile *NASA* pentru experimente.

O altă direcţie este folosirea metodelor de învăţare semi-supervizate, bazate pe dezacord. Asemenea abordări sunt *ROCUS* [JLZ11], *ACoForest* [LZWZ12] şi folosirea metodei *Random Forests* prezentată în [GMCS04].

## 1.4 Prezentare metodelor relevante din Instruirea Automată

### 1.4.1 Reguli de Asociere Relaţionale

Extragerea regulilor de asociere din baze de date care conţin tranzacţii de articole a fost descrisă pentru prima dată de Agrawal et. al în anul 1993 [AIS93]. O *regulă de asociere* este o implicaţie de forma $X \Rightarrow Y$, unde $X$ şi $Y$ sunt mulţimi disjuncte de articole şi înseamnă că prezenţa lui $X$ implică prezenţa lui $Y$.

Una dintre dezavantajele regulilor de asociere este că acestea consideră doar prezenţa sau absenţa unui articol, şi ignoră orice relaţie care ar putea exista între elementele dintr-o mulţime de articole. Pentru a depăşi acest dezavantaj Marcus et al. propune în [MML01] o extensie a regulilor de asociere, *regulile de asociere ordinale*, unde există relaţii ordinale între elementele dintr-o mulţime de articole.

Totuşi, relaţiile ordinale nu sunt tot timpul suficiente, pot exista alte relaţii, care nu sunt ordinale, între articole. De aceea, *reguli de asociere relaţionale*, o extensie a regulilor de asociere ordinale, au fost introduse în [SCC06]. Regulile de asociere relaţionale permit orice tip de relaţie între articolele dintr-o mulţime.

Formal, aşa cum este prezentat în [SCC06], să considerăm că $R = \{r_1, r_2, \ldots, r_n\}$ este o mulţime de instanţe. Fiecare instanţă este caracterizată de o listă de $m$ atribute, $(a_1, \ldots, a_m)$. Valoarea atributului $a_i$ în instanţa $r_j$ este notată cu $\Phi(r_j, a_i)$. Pentru fiecare atribut $a_i$ există un domeniu $D_i$ de unde provin valorile. Acest domeniu conţine şi valoarea vidă, notată cu $\varepsilon$. Între doi domenii $D_i$ şi $D_j$ relaţii, cum ar fi *mai mic sau egal* ($\leq$), *egal* ($=$) şi *mai mare sau egal* ($\geq$) pot fi definite. Mulţimea tuturor relaţiilor posibile care pot fi definite între $D_i$

x $D_j$ este notată cu $M$.

**Definiţie 1** *[SCC06] O* regulă de asociere relaţională *este o expresie* $(a_{i_1}, a_{i_2}, a_{i_3}, \ldots, a_{i_\ell}) \Rightarrow (a_{i_1}\ \mu_1\, a_{i_2}\ \mu_2\ a_{i_3} \ldots \mu_{\ell-1}\ a_{i_\ell})$, *unde* $\{a_{i_1}, a_{i_2}, a_{i_3},\ \ldots, a_{i_\ell}\} \subseteq \mathcal{A} = \{a_1, \ldots, a_m\}$, $a_{i_j} \neq a_{i_k}$, $j, k = 1..\ell$, $j \neq k$ *şi* $\mu_i \in M$ *este o relaţie pe* $D_{i_j} \times D_{i_{j+1}}$, $D_{i_j}$ *este domeniul atributului* $a_{i_j}$ *dacă atributele* $a_{i_1}, a_{i_2}, a_{i_3}, \ldots, a_{i_\ell}$ *apar împreună (nu sunt vide) în s% dintre cele n instanţe (s fiind numit* suportul *regulii) şi notăm* $R' \subseteq R$ *mulţimea de instanţe unde* $a_{i_1}, a_{i_2}, \ldots, a_{i_l}$ *apar împreună şi* $\Phi(r_j, a_{i_k})\ \mu_k\ \Phi(r_j, a_{i_{k+1}})$ *este adevărat pentru fiecare* $1 \leq k \leq l-1$ *pentru fiecare instanţă* $r_j$ *din* $R'$ *(*$c = |R'|/|R|$ *fiind numit* confidenţa *regulii).*

În afară de suport şi confidenţă, o regulă poate fi caracterizată şi de lungimea ei, care este numărul de atribute care apar în ea. Într-un set de date un număr mare de reguli de asociere relaţionale pot fi găsite, dar în general suntem interesaţi doar de acele reguli ale căror confidenţă şi suport este mai mare decât un prag ales de utilizator $c_{min}$ şi $s_{min}$. Aceste reguli sunt numite interesante.

### 1.4.2 Clustering

*Clusteringul* este considerat cea mai importantă metodă de învăţare nesupervizată, a cărei scop principal este gruparea entităţilor dintr-un set de date în diferite grupe (numite *clusteri*) în aşa fel încât entităţile într-un *cluster* să fie mai similare între ele decât cu entităţile din afara *clusterului*.

Fie $\mathcal{O} = \{O_1, O_2, \ldots, O_n\}$ mulţimea obiectelor care trebuie să fie *clusterizaţi*. Pe parcursul procesului de clusterizare, un *cluster* $C$ este o mulţime nevidă de obiecte din $\mathcal{O}$, care aparţin împreună pe baza unei funcţii de distanţă $d$. O *partiţie* $P = (C_1, C_2, ..., C_k)$ este o mulţime de *clusteri*, astfel încât fiecare obiect din $\mathcal{O}$ apare într-un singur *cluster* din $P$.

Algoritmii de *clustering* pot fi împărţiţi în două grupe mari: algoritmi *ierarhici* şi *partiţionali*, iar algoritmii ierarhici pot fi împărţiţi în algoritmi *aglomerativi* şi *divizivi*. În cazul algoritmilor aglomerativi, procesul începe cu o partiţie în care fiecare entitate este plasată în propriul *cluster*, iar numărul *clusterilor* scade treptat, până se ajunge la o partiţie în care toate entităţile sunt în acelaşi *cluster*.

Un pas important pe parcursul *clusterizării* ierarhice este calculul distanţei între doi *clusteri*, notată cu $dist(C_i, C_j)$, pentru ca această distanţă decide care *clusteri* vor fi uniţi. Cele mai cunoscute metrici de distanţă în literatura de specialitate sunt *single-link* - unde distanţa minimă dintre două entităti din $C_i$ şi $C_j$ este considerată -, *complete-link* - unde distanţa maximă este considerată - şi *average-link* - unde media distanţelor se foloseşte.

Algoritmi de *clustering* partiţionali [Han05] crează o singură partiţie care conţine $K$ *clusteri*. Algoritmul porneşte cu $K$ centre iniţiale pentru *clusteri* şi fiecare entitate este adăugată în *clusterul* cu centrul cel mai apropiat. După aceea, centrele sunt recalculate şi entităţile sunt adăugate din nou. Aceste paşi se repetă până se ajunge la *clusteri* stabili.

# Capitolul 2

# Abordări noi pentru Remodularizare Soft

În acest capitol, care este original, abordăm problema de remodularizare soft. Remodularizare soft înseamnă modificarea structurii interne a unui sistem informatic, fără a afecta comportamentul extern, şi este o activitate importantă, pentru că ajută la menţinerea unei proiectări bune a sistemului. În acest capitol sunt prezentate abordările noastre pentru remodularizare la nivel de pachet si la nivel de clasă folosind *clustering*.

Abordările prezentate în acest capitol reprezintă lucrările noastre originale, publicate în [Mar14b], [MCC12], [Mar12c], [Mar12b], [Mar12a], sau în curs de recenzie: [MCC14].

**Motivaţie.** În zilele noastre sistemele informatice trebuie să se adapteze cerinţelor utilizatorilor sau a mediului în care sunt folosite. Majoritatea modificărilor legate de această adaptare sunt făcute după lansarea softului, şi în multe cazuri influenţează negativ calitatea sistemului. Acest lucru face întreţinerea mai complicată şi costisitoare. Pentru a evita aceste costuri, întreţinerea calităţii sistemului este un scop important, dar, din cauza dimensiunii sistemelor, e aproape imposibil pentru un dezvoltator să aibă o viziune de ansamblu asupra sistemului şi să decidă modificările necesare. De aceea, diferite metode şi unelte care găsesc ce modificări ar trebui făcute pentru îmbunătăţirea calităţii sistemului sunt binevenite.

Am plecat de la ideea că un sistem informatic poate fi considerat un set de date, care conţine înregistrări care pot fi elementele sistemului informatic. În acest context aplicarea algoritmilor de *clustering* pentru a grupa entităţile care aparţin împreună pare naturală. Pentru a ghida acest proces, am decis să folosim metrici soft, pentru că s-au dovedit a fi utile în alte abordări.

## 2.1 Remodularizare la nivel de pachete

În această secţiune se prezintă abordarea originală pentru restructurarea unui sistem informatic la nivel de pachete. Această abordare primeşte un sistem informatic şi îl remodularizează la nivel de pachet folosind *clustering*-ul ierarhic pentru a obţine pachete structurate mai bine. Considerând o structură de pachete existentă, metoda propusă în această secţiune poate fi folosită şi pentru a sugera dezvoltatorului pachetul potrivit pentru o clasă nou adăgată.

### 2.1.1 Abordarea *CASP*

În această subsecţiune se introduce o abordare bazată pe *clustering* (*CASP - Clustering Approach for Software Packages Restructuring*) pentru remodularizare la nivel de pachete, alcătuită din doi paşi:

- **Colectarea datelor** - Sistemul informatic este analizat pentru a extrage din el informaţia relevantă despre clasele, metodele, atributele şi relaţiile dintre ele.
- **Gruparea** - Clasele din sistemul informatic sunt grupate în pachete folosind informaţia extrasă la pasul anterior şi un algoritm de *clustering* (*HASP* în abordarea noastră). Scopul acestui pas este obţinerea unei împărţiri a sistemului în pachete.

### 2.1.2 Gruparea în pachete

În continuare se introduce un algoritm nou de *clustering* ierarhic aglomerativ (*HASP - Hierarchical Clustering Algorithm for Software Packages Restructuring*) care urmăreşte identificarea unei partiţii a unui *framework* $S$, care corespunde unei structuri de pachete bune. În această partiţie fiecare *cluster* reprezintă câte un pachet al sistemului informatic.

Au fost identificate şapte caracteristici relevante pentru problema împărţirii claselor în pachete. Aceste caracteristici măsoară coeziunea, cuplarea, potenţialul de refolosire şi cât de similare sunt elementele dintr-un pachet. Ele au fost agregate într-o singură valoare, numită *score*, în felul următor:

$$score(K_i \bigcup K_j, \mathcal{K}^*) = \frac{\sum_{i=1}^{2} w_i \cdot F_i - w_3 \cdot F_3}{|K_i \bigcup K_j|^2 - 1} + \sum_{i=4}^{7} w_i \cdot F_i \tag{2.1}$$

unde $K_i$ şi $K_j$ reprezintă doi *clusteri* care voi fi uniţi, $\mathcal{K}^*$ reprezintă partiţia fără $K_i$ şi $K_j$, $F_i,\ 1 \leq i \leq 7$, sunt valorile celor şapte caracteristici şi $w_i$ $(0 \leq w_i \leq 1)$ sunt valorile ponderilor asociate acestor caracteristici. Aceste ponderi au fost determinate folosind o căutare de tip *grid* şi un sistem informatic pentru care se cunoaşte o structură corectă a pachetelor. Pentru a ghida procesul de căutare de tip *grid* am introdus măsura *CIP* (*Cohesion of Identified Package*) care determină distanţa dintre două partiţii.

Algoritmul *HASP* este bazat pe ideea de *clustering* ierarhic aglomerativ. La fiecare pas, perechea de *clusteri* pentru care *score*-ul este cel mai mare este unită. Procesul de *clustering* se efectuază până când se ajunge la o partiţie cu un singur *cluster*, şi creează o serie de partiţii cu un număr de *clusteri* care tot scade. Pentru a identifica "cea mai bună" partiţie dintre toate cele generate, am introdus o nouă măsură, *overallScore*, care e calculată considerând valoarea pentru o variantă uşor modificată a măsurii *score*.

### 2.1.3 Atribuirea claselor în pachete

O altă problemă pe care dezvoltatorii soft o întâlnesc des este găsirea pachetului potrivit pentru o nouă clasă. Pentru a rezolva această problemă am introdus algoritmul *AssignClass*. Să considerăm că $\mathcal{K} = \{K_1, K_2, ..., K_v\}$ este partiţia care reprezintă structura actuală de pachete a sistemului. O clasă nouă, $C$, este adăugată în sistem. Pentru a găsi cel mai potrivit pachet pentru $C$, se calculează $score(K_i \bigcup C,\ \mathcal{K}^*)$ pentru fiecare pachet $K_i \in \mathcal{K}$. Pachetul pentru care *score* este maxim este cel în care $C$ ar trebui adăugată.

### 2.1.4 Evaluarea experimentală

Pentru evaluarea experimentală a abordării noastre au fost folosite două *framework*-uri disponibile public, *Commons DbUtils* [DbU] şi un *Reinformcement Learning* (*Învăţare cu întărire*) *framework* [rlf]. Pentru ambele *framework*-uri au fost efectuate două experimente: am executat algoritmul *HASP* pentru a identifica o împărţire bună a claselor în pachete şi am identificat pachetul potrivit pentru anumite clase folosind algoritmul *AssignClass*.

#### 2.1.4.1 Commons DbUtils. Rezultate

Am aplicat algoritmul *HASP* pe *framework*-ul *DbUtils*; acesta a returnat o partiţie cu patru *clusteri* (pachete) ca structura optimă, în locul structurii originale alcătuite din trei pachete. Analizând rezultatele algoritmului am concluzionat ca partiţia sugerată de algoritmul *HASP* este mai bună decăt cea originală.

La al doilea experiment am executat de mai multe ori algoritmul *AssignClass*, de fiecare dată eliminând câte o clasă din *DbUtils* şi încercând să găsim pachetul potrivit pentru ele. Din 19 rulări ale algoritmului în 16 cazuri el a identificat pachetul corect, având o acurateţe de 84.2 %.

#### 2.1.4.2 Reinforcement Learning. Rezultate

Am aplicat algoritmul *HASP* pentru *framework*-ul Reinforcement Learning; acesta a returnat o partiţie cu 11 pachete, în locul partiţiei originale (despre care ştim că este cea corectă) cu 10 pachete. Totuşi, cele două partiţii sunt foarte similare. Am calculat valoare metricii *CIP* pentru a vedea cât de aproape este partiţia generată de algoritmul *HASP* de cea originală, şi a dat valoarea 0.95, care este destul de mare.

Pentru al doilea experiment am creat trei clase noi şi am folosit algoritmul *AssignClass* pentru a sugera pachetul potrivit pentru ele. Toate cele trei clase au fost plasate în pachetul corect de algoritmul *AssignClass*.

### 2.1.5 Analiză şi comparaţie cu metode existente

Am efectuat anumite experimente adiţionale pe lângă cele prezentate mai sus pentru a investiga eficacitatea algoritmului *HASP*. Pentru aceste experimente am folosit pe lângă *framework*-ul *DbUtils* alte două *framework*-uri disponibile public *Email* [Ema] şi *EL* [EL].

În această analiză am calculat valoarea diferitor metrici şi măsuri din literatura de specialitate pentru structura originală a sistemelor şi pentru structura sugerată de algoritmul *HASP*. Am calculat în total 29 de valori, şi în 25 de cazuri partiţia generată de algoritmul *HASP* avea o valoarea mai bună sau egală cu partiţia originală.

Dintre metodele raportate în literatura de specialitate, cea mai apropiată de metoda noastră este abordarea din [AAM11], care foloseşte *clustering* ierarhic precum o face şi metoda noastră. Acest articol prezintă două abordări; ambele au fost aplicate pe *framework*-ul *DbUtils*. Prima abordare este bazată pe numărul de apeluri la constructorul claselor, dar aceasta nu este o direcţie bună în cazul *framework*-urilor, care în general au multe interfeţe şi clase abstracte care nu sunt iniţializate niciodată. Pentru *framework*-ul *DbUtils* această abordare nu sugerează nicio modificare.

A doua metodă prezentată în [AAM11] foloseşte o reprezentare multi-dimensională bazată pe dependenţe create de apeluri de metode. Toate cele trei măsuri de distanţă la *clustering* (*single-link*, *complete-link* şi *average-link*) au fost încercate; de asemenea a fost introdus şi un algoritm nou cu o complexitate computaţională mai mică. Am încercat cele trei metode de *clustering* pe sistemul *DbUtils*, folosind ca şi criteriu de oprire momentul când distanţa între toate perechile de clusteri a fost 1 (cea mai mare distanţă posibilă). După aceea am calculat valoarea măsurii *CIP* între rezultatele primite şi structurile originale şi sugerate de *HASP*. Valorile între 0.3 şi 0.43 sugerează că rezultatele sunt departe de ambele.

### 2.1.6 O comparaţie a măsurii *overallScore* cu alte măsuri

În continuare este prezentat un studiu efectuat pentru a evalua cât de bine sunt structurate pachetele într-un sistem informatic, folosind cele două măsuri introduse mai sus: *overallScore*

şi *CIP*. Prin experimentele efectuate, urmărim să accentuăm că cele două măsuri sunt bine-corelate. Astfel, în locul măsurii *CIP*, care necesită cunoaşterea unei structuri de pachete corecte, *overallScore* poate fi folosită pentru a evalua pachetele dintr-un sistem informatic.

Pentru experimente au fost folosite cele trei sisteme disponibile public utilizate şi mai sus: *DbUtils*, *Email* şi *EL*. Pentru fiecare sistem am considerat patru structuri de pachete diferite: originala, cea sugerată de algoritmul *HASP* şi două "partiţii greşite" create manual prin modificarea structurii originale a pachetelor.

Pentru experimente am calculat valoarea măsurilor *overallScore* şi *CIP* pentru fiecare partiţie. După aceea am calculat valoarea a două măsuri de corelaţie bazate pe ranguri: corelaţia Spearman [Spe04] şi *Footrule*-ul lui Spearman [DG77]. Ambele măsuri de corelaţie sugerează că măsurile *overallScore* şi *CIP* sunt bine corelate.

Pentru a compara aceste măsuri cu alte măsuri din literatura de specialitate, am calculat valoarea a şapte metrici introduse în [DABH11] pentru cele patru partiţii pentru cele trei sisteme. După aceea am calculat *Footrule*-ul lui Spearman pentru aceste metrici şi măsura *overallScore*. Comparaţia valorilor este prezentată în Figura 2.1.

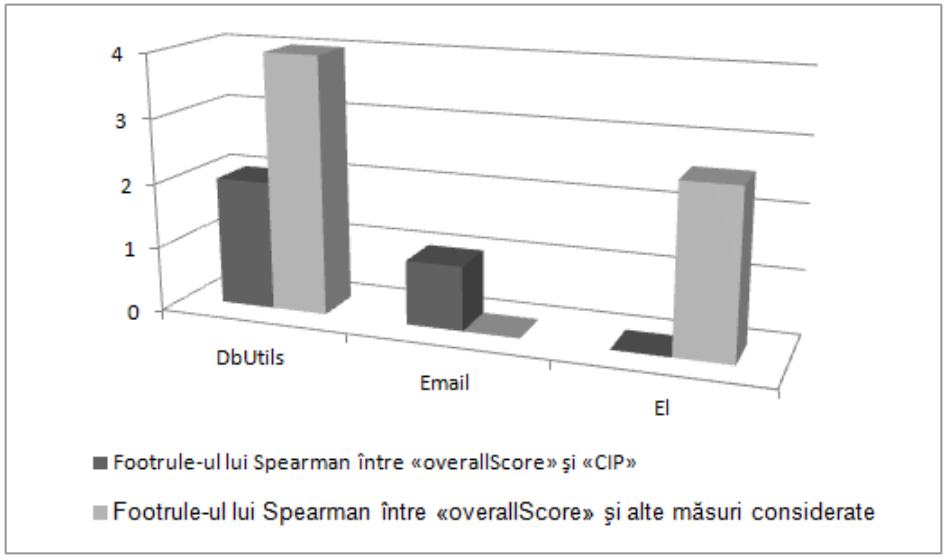


Figura 2.1: Comparaţia valorilor *Footrule*.

Considerând comparaţia valorilor *Footrule* prezentate în Figura 2.1 putem concluziona că măsura *overallScore* este mai potrivită pentru evaluarea unei partiţii a sistemului informatic decât restul metricilor considerate. În plus, *overallScore* poate fi folosit în locul măsurii *CIP*, deoarece cele două sunt bine corelate.

## 2.2 Remodularizare la nivel de clasă

În această secţiune este prezentată o abordarea originală pentru remodularizarea la nivel de clasă a unui sistem informatic [MCC12, Mar12c, Mar12b, Mar12a]. Vor fi prezentate metode care sunt capabile să identifice automat o structură îmbunătăţită a sistemului informatic folosind valoarea diferitor metrici soft. Această structura internă îmbunătăţită poate fi realizată aplicând diferite restructurări pe sistemul original.

### 2.2.1 Restructurare bazată pe metrici soft

În următoarele subsecţiuni următoarele notaţii vor fi folosite: vom considera că un sistem informatic este o mulţime $S = \{s_1, s_2, ..., s_n\}$ unde $s_i$, $1 \leq i \leq n$, se numeşte o entitate şi

poate fi o clasă sau o metodă dintr-o clasă. Am dorit să folosim valoarea diferitor metrici pentru a reprezenta aceste entităţi, deci am avut nevoie de metrici care pot fi calculate atât pentru clase cât şi pentru metode. Am ales următoarele 5 metrici: *Relevant Properties* (*RP*), *Depth in Inheritance Tree* (*DIT*) [CK91], *Number of Children* (*NOC*) [CK91], *Fan-In* (*FI*) [HK81, Mai09], *Fan-Out* (*FO*) [HK81, Mai09]. Dintre acestea, valoarea lui *RP* este o mulţime, iar restul metricilor au ca valoare numere naturale.

#### 2.2.1.1 Modelul vectorial şi funcţia de distanţă

Ideea de bază a abordării noastre este să caracterizăm entităţile dintr-un sistem informatic printr-o listă de valori a metricilor prezentate mai sus. Astfel, fiecare entitate $s_i$ $(1 \leq i \leq n)$ din sistemul informatic va fi reprezentată ca un vector cu 5 componente: $(rp(s_i), dit(s_i), noc(s_i), fi(s_i), fo(s_i))$. Valorile metricilor au fost scalate în intervalul [0,1] acolo unde a fost necesar. Pentru a aplica un algoritm de *clustering* pe această reprezentare am definit o funcţie de distanţă, care este prezentată pe Ecuaţia 2.2.

$$d(s_i, s_j) = \begin{cases} 0 & dacă \quad i = j, \\ \sqrt{\frac{1}{5} \cdot \left(1 - \frac{|s_{i1} \cap s_{j1}|}{|s_{i1} \cup s_{j1}|} + \sum_{k=2}^{5} (s_{ik} - s_{jk})^2\right)} & dacă \quad s_{i1} \cap s_{j1} \neq \emptyset \ , \\ \infty & altfel \end{cases} \tag{2.2}$$

#### 2.2.1.2 Algoritmul ARI

Folosind metricile soft prezentate mai sus, modelul vectorial şi funcţia de distanţă am definit algoritmul *ARI* (*Automatic Refactoring Identification*). Ideea principală a algoritmului este identificarea unei partiţii $\mathcal{K} = \{K_1, K_2, ..., K_m\}$ a sistemului informatic $S$ care corespunde unei structuri îmbunătăţite a lui $S$. Fiecare *cluster* $K_i$ corespunde unei clase din această structură îmbunătăţită.

Algoritmul *ARI* porneşte cu o partiţie vidă. După aceea pentru fiecare metodă din $S$ clasa cea mai apropiată (considerând funcţia de distanţă din Ecuaţia 2.2) este căutată. Dacă se găseşte o clasă cu o distanţă mai mică decât $\infty$ metoda ori este plasată în *clusterul* clasei (daca clasa este deja întru-un *cluster*) ori un *cluster* nou este creat unde cele două entităţi sunt plasate. Dacă distanţa metodei faţă de toate clasele este $\infty$, căutăm în *clusterii* fără clase metoda cea mai apropiată de această metodă. Dacă toate distanţele sunt $\infty$ vom crea un *cluster* nou şi vom adăga metoda în acel *cluster*. Altfel o adăugăm în *clusterul* găsit. Când toate metodele au fost adăugate, calculăm distanţa dintre toate perechile de clase. Dacă este vreo pereche cu distanţă sub $\infty$ vom uni *clusterii* respectivi.

#### 2.2.1.3 Algoritmul *HAC*

Folosind aceeaşi metrici, reprezentare şi funcţie de distanţă, am introdus o extensie a algoritmului *ARI*, numit algoritmul *HAC* (*Hierarchical Agglomerative Clustering*). Este bazată pe *clustering* ierarhic aglomerativ şi foloseşte o funcţie euristică care exprimă că doi clusteri sunt uniţi doar dacă distanţa dintre ei este mai mică ca 1. Motivul pentru folosirea acestei funcţii este că distanţe mai mari ca 1 apar numai pentru entităti necoezive.

#### 2.2.1.4 Restructurări identificate

Atât algoritmul *ARI* cât şi algoritmul *HAC* creează o partiţie a sistemului informatic $S$, care corespunde unei structuri interne bune. Comparând această partiţie cu partiţia originală, o listă de restructurări poate fi identificată, care ar îmbunătăţi structura sistemului

| Metrică | Ex 1 | Ex 2 | Ex 3 | Ex 4 | Metrică | Ex 1 | Ex 2 | Ex 3 | Ex4 |
|---|---|---|---|---|---|---|---|---|---|
| NOA | A | A | A | A | ICH | C | A | A | A |
| NOM | C | A | A | C | CA | A | A | A | A |
| CBO | A | A | A | A | INS | A | A | A | A |
| DAC | A | A | A | A | LCOM1 | A | S | S | S |
| TCC | A | C | C | C | LCOM2 | A | S | S | S |
| LCC | A | C | C | C | LCOM4 | A | A | S | S |
| MPC | S | A | S | S | LCOM5 | A | C | A | C |
| RFC | C | A | C | A | LD | A | A | C | A |

Tabela 2.1: Direcţia în care valoarea metricilor s-a modificat pentru cele două variante a exemplelor.

informatic. Pentru ambii algoritmi aceste restructurări sunt: *Move Method*, *Inline Class* şi *Extract Class*.

### 2.2.2 Restructurare bazată pe metrici soft agregate

Această subsecţiune prezintă un al treilea algoritm pentru restructurarea sistemelor informatice, care, similar cu algoritmi prezentaţi mai sus, foloseşte metrici soft, dar agreghează valorile lor într-o singură valoare.

Pentru această abordare am avut nevoie de metrici soft care pot fi calculate pentru o clasă. De asemenea, din moment ce aceste valori vor fi folosite să caracterizeze calitatea unei clase (şi calitatea sistemului informatic prin clasele lui) intenţia noastră a fost să alegem metrici care măsoară dimensiunea, coeziunea şi cuplarea sistemului informatic. Studiind lista metricilor implementate de diferite instrumente am selectat următoarele 16 metrici: *Afferent Coupling* (*CA*) [BJWD99], *Coupling Between Objects* (*CBO*) [CK94], *Data Abstraction Coupling* (*DAC*) [LH93], *Information Flow Based Cohesion* (*ICH*) [LLWW95], *Instability* (*INS*), *Tight Class Coupling* (*TCC*) [BK95], *Loose Class Coupling* (*LCC*) [BK95], *Lack of Cohesion in Methods 1* (*LCOM1*) [CK94], *Lack of Cohesion in Methods 2* (*LCOM2*) [HS96], *Lack of Cohesion in Methods 4* (*LCOM4*) [HM95], *Lack of Cohesion in Methods 5* (*LCOM5*), *Locality of Data* (*LD*) [HM95], *Message Passing Coupling* (*MPC*) [LH93], *Number of Attributes* (*NOA*) [LH93], *Number of Methods* (*NOM*) [LH93], *Response for a Class* (*RFC*) [CK94].

#### 2.2.2.1 Studiu de caz

Deoarece valori ideale pentru aceste metrici soft nu sunt cunoscute, am efectuat un studiu, pentru a analiza cum se modifică valoarea acestor metrici dacă îmbunătăţim sistemul informatic. Am ales patru cazuri de studii, fiecare fiind un exemplu mic cu 2-3 clase, şi în fiecare exemplu a fost cel puţin o metodă care era plasată într-o clasă diferită faţă de clasa unde ar fi trebuit să fie.

Am calculat valoarea metricilor prezentate în subsecţiunea anterioară pentru aceste exemple. După aceea, am "corectat" exemplele, mutând metodele în clasele unde ar trebui să fie, şi am calculat valoarea metricilor din nou. Am fost interesate să vedem cum se modifică valoarea metricilor pentru cele două versiuni. Rezultatul studiului este prezentat pe Tabelul 2.1, unde eticheta *A* înseamnă că valoarea metricii a rămas aceeaşi, *C* înseamnă că a crescut, iar *S* înseamnă că a scăzut pentru varianta corectă.

Cel mai important lucru de observat pe Tabelul 2.1 este că pentru fiecare metrică direcţia schimbării este consistentă: pentru fiecare exemplu valoarea unei metrici ori nu s-a modificat deloc, sau, dacă s-a modificat, tot timpul s-a modificat în aceeaşi direcţie. Acest lucru este importantă, pentru că ne ajută la comparaţia a două versiuni a aceluiaşi sistem.

#### 2.2.2.2 Algoritm de restructurare bazat pe metrici soft agregate

Ideea principală a algoritmului nostru este să caracterizăm fiecare clasă din sistemul informatic printr-o singură valoare agregată a acestor metrici soft. Pentru asta, prima dată am normalizat valorile metricilor. Am efectuat normalizarea în aşa fel încât cu cât e mai mică valoarea unei metrici, cu atât e mai bună proiectarea sistemului.

După normalizarea valorilor am introdus o măsură, $M(C)$, definită pentru o clasă $C$ ca suma valorilor normalizate pentru cele 16 metrici soft. Cu cât e mai mică valoarea acestei metrici, cu atât e mai bună structura internă a clasei $C$. Deoarece un sistem informatic $S$ este alcătuit din mai multe clase, putem defini o măsură agregată, $AM(S)$, ca media valorilor $M(C)$ pentru toate clasele din sistem.

Folosind măsura $AM(S)$ şi *clustering* ierarhic aglomerativ, abordarea noastră prima dată plasează fiecare metodă în propriul *cluster* (care reprezintă o clasă cu o singură metodă), şi va uni *clusteri* până când ajunge la o partiţie unde toate metodele sunt în acelaşi *cluster*. Din seria partiţiilor generate se returnează ca soluţie partiţia cu cea mai mică valoare pentru măsura $AM(S)$.

### 2.2.3 Evaluare experimentală

**Algoritmul *ARI*.** Pentru a evalua performanţa algoritmului *ARI* am folosit două studii de caz. Primul este un exemplu simplu, cu două clase, unde o clasă conţine o metodă care ar trebui să fie în cealaltă clasă. Pentru acest exemplu algoritmul *ARI* sugerează mutarea metodei în cealaltă clasă.

Al doilea studiu de caz folosit pentru evaluarea algoritmului *ARI* este sistemul informatic *JHotDraw* versiunea 5.1 disponibil public [Gam]. Am ales sistemul *JHotDraw* pentru că este un exemplu bine cunoscut pentu folosirea şabloanelor de proiectare şi pentru că este un proiect complex, alcătuit din 173 clase, 1375 metode şi 475 atribute.

Algoritmul *ARI* a identificat 20 de restructurări *Move Method* în sistemul *JHotDraw*. După o analiză a rezultatelor am ajuns la concluzia că 11 sunt justificate şi 9 sunt greşite.

**Algoritmul *HAC*.** Am folosit sistemul informatic *JHotDraw* ca studiu de caz şi la algoritmul *HAC*. Rezultatul algoritmului a conţinut doar trei *clusteri* (reprezentând clase) care erau diferiţi de structura existentă a sistemului. Aceşti trei *clusteri* reprezintă restructurări posibile de tip *Extract Class*, adică clase noi ar trebui create şi metodele selecţionate ar trebui mutate în aceste clase noi. După o analiză a rezultatelor am ajuns la concluzia că o restructurare este justificată, iar celelalte două sunt parţial justificate, adică o parte din metodele sugerate ar trebui mutate în clase noi.

**Abordare bazată pe metrici soft agregate.** Evaluarea abordării bazate pe metrici soft agregate a fost efectuată până acum doar pe exemple mici (cele patru studii de caz prezentate în Subsecţiunea 2.2.2), pentru că anumiţi paşi din algoritm trebuie să fie executaţi manual. Motivul este că pentru a avea rezultate valide, trebuie să ne asigurăm la fiecare pas că nu am schimbat comportamentul sistemului.

Am aplicat algoritmul pentru trei exemple din Subsecţiunea 2.2.2 şi în toate cele 3 cazuri a identificat partiţia corectă pentru studiul de caz.

#### 2.2.3.1 Analiză comparativă

**Analiză comparativă a celor trei algoritmi.** Algoritmii *ARI* şi *HAC* sunt destul de similari, folosesc acelaşi model vectorial pentru a reprezenta entităţile din sistemul informatic şi aceleaşi metrici soft. Cu toate acestea, algoritmul *HAC* foloseşte *clustering* ierarhic aglomerativ, o metodă de învăţare nesupervizată.

Ambii algoritmi au identificat majoritatea sistemului *JHotDraw* ca fiind corect proiectat, aşa cum ne-am aşteptat, deoarece *JHotDraw* este considerat un exemplu de proiectare corectă. La restructurările sugerate, algoritmul *HAC* a avut doar 5 greşeli, în timp ce algoritmul *ARI* a avut 9. De asemenea, algoritmul *HAC* a identificat 3 restructurări *Extract Class*, care nu au fost identificare de algoritmul *ARI*, astfel demonstrând că folosirea metodelor de învăţare nesupervizate este folositoare, pentru că pot identifica şabloane ascunse în date.

Algoritmul bazat pe metrici soft agregate are un studiu de caz comun cu algoritmul *ARI*, acel exemplu artificial simplu care a fost primul caz de studiu pentru *ARI*. Pentru acest exemplu ambii algoritmi au identificat restructurarea corectă.

**Analiză comparativă cu metode existente.** Seng et al. aplică în [SSB06] o căutare multiobiectivă ponderată unde metrici soft sunt combinate într-o singură funcţie obiectivă. Această abordare produce parţial aceleaşi rezultate ca şi algoritmul *ARI*, dar *ARI* are mai puţine sugestii greşite, este deterministic şi are un timp de rulare mai mic.

În [CS06] este prezentată o abordare care foloseşte *clustering* pentru a îmbunătăţi structura de clase înt-un sistem informatic. Această abordare foloseşte un algoritm de *clustering* partiţional, *kRED* (*k-means for REfactorings Determination*). Spre deosebire de *kRED*, algoritmul *ARI* poate să identifice şi restructurări de tip *Extract Class*. Comparat cu [RR11] şi [FTCS09] algoritmii *HAC* şi *ARI* pot identifica trei tipuri de restructurări (nu numai *Extract Class*).

Abordarea cea mai apropiată de abordarea noastră este algoritmul *HARS*, [CS07], care foloseşte doar metrica *Relevant Properties* pentru a calcula distanţa dintre două entităţi, şi care consideră şi atributele ca fiind entităţi. În [CS07] sistemul informatic *JHotDraw* este folosit ca şi caz de studiu, şi găseşte doar una dintre cele trei restructurări *Extract Class* pe care algoritmul *HAC* le găseşte (dar găseşte două restructurări *Move Attribute* pe care nici *ARI* nici *HAC* nu le pot găsi).

## 2.3 Concluzii şi cercetări ulterioare

Acest capitol a prezentat diferite metode pentru restructurarea automată a unui sistem informatic pentru a îmbunătăţi calitatea lui. Am început cu prezentarea abordării *CASP* şi a algoritmului *HASP*, care este o metodă de restructurare la nivel de pachete. Am evaluat experimental abordarea noastră folosind două studii de caz, şi rezultatele arată că algoritmul *HASP* poate găsi o structură bună a pachetelor.

Am prezentat de asemenea trei algoritmi care folosesc valoarea diferitor metrici soft şi *clustering* ierarhic pentru restructurarea la nivel de clase a unui sistem informatic. Evaluarea experimentală a acestor metode arată că ele pot identifica o partiţie a sistemului care are o proiectare mai bună.

În viitor dorim să extindem evaluarea experimentală pentru toate abordările şi pe alte sisteme informatice disponibile public. De asemenea dorim să îmbunătăţim abordările noastre eliminând deficienţele posibile.

# Capitolul 3

# Abordări noi pentru Detectarea Defectelor în Sisteme Informatice

În acest capitol, care este original, abordăm problema de detectare a defectelor în sisteme informatice, o problemă importantă în inginerie soft, deoarece, din cauza dimensiunii şi complexităţii mari a sistemelor informatice, găsirea manuală a acestor defecte este din ce în ce mai complicată. Prin urmare, abordările inteligente, care îi ajută dezvoltatorii soft în identificarea părţilor defective care necesită atenţie, sunt binevenite. În literatura de specialtate se pot găsi abordări bazate pe instruire automată pentru ambele direcţii, *detectarea defectelor de proiectare* şi *predicţia defectelor*. În acest capitol prezentăm o abordare nouă, şi anume, aplicarea regulilor de asociere relaţionale pentru ambele direcţii.

Cele două abordări prezentate în acest capitol sunt lucrări originale, publicate în [CMC14a], [Mar13b], [CMC14b] şi [Mar13a].

**Motivaţie.** Toate sistemele informatice trec prin multe schimbări pe parcursul ciclului de viaţă, altfel devin învechite şi nu mai sunt folosite. Aceste modificări, executate în general pe parcursul întreţinerii, pot adăuga funcţionalităţi noi, corecta defectele existente, sau pot fi făcute pentru adaptarea sistemului la mediul în care este folosit. Ele pot degrada structura sistemului, ceea ce, la rândul ei, va face modificările următoare şi mai grele. Pentru a evita acest ciclu vicios, identificarea entităţilor cu defecte este foarte importantă.

Un sistem informatic poate fi reprezentat ca un set de date unde elementele sunt clasele, folosind o reprezentare multi-dimensională bazată pe metrici soft. Din acest set de date informaţia importantă poate fi extrasă. Diferite tipuri de relaţii pot fi definite între valorile metricilor soft şi reguli de asociere relaţionale pot fi extrase. Aceste reguli pot furniza şabloane interesante, care pot indica clasele cu defecte de proiectare, sau ele pot descrie entităţile defective şi non-defective din sistemul informatic.

## 3.1 Model teoretic

În această secţiune prezentăm modelul teoretic care este folosit atât la abordarea de detectare a defectelor de proiectare, cât şi la abordarea de predicţia defectelor.

Ideea principală a modelului este reprezentarea entităţilor dintr-un sistem informatic ca un vector multidimensional, elementele căruia sunt valorile diferitor metrici soft calculate pentru entitatea respectivă. Considerăm că un sistem informatic este o mulţime de componente (entităţi) $\mathcal{S} = \{s_1, s_2, ..., s_n\}$. Deoarece dorim să identificăm entităţile rău-structurate, vom considera o mulţime de metrici soft care caracterizează structura internă a entităţilor. Prin urmare, avem un set de caracteristici alcătuit din metrici soft $\mathcal{SM} = \{sm_1, sm_2, ..., sm_k\}$ şi astfel, fiecare entitate $s_i \in \mathcal{S}$ din sistemul informatic poate fi reprezentată ca un vector $k$-dimensional, compus din valoarea metricilor soft din $\mathcal{SM}$.

## 3.2 Detectarea Defectelor de Proiectare

Această secţiune prezintă abordarea noastră, numită *SDDRAR* (*Software Design Defect detection using Relational Association Rules*), pentru detectarea defectelor de proiectare într-un sistem informatic [CMC14a].

### 3.2.1 Algoritmul *DRAR*

Regulile de asociere relaţionale sunt o extensie a regulilor de asociere ordinale (aşa cum a fost prezentat şi în Secţiunea 1.4.1), care permit definirea relaţiilor mai generale între atribute, nu doar relaţii ordinale. Un algoritm similar cu algoritmul Apriori, intitulat *DOAR* (*Discovery of Ordinal Association Rules*) a fost introdus în [CSTM06]. Acest algoritm găseşte într-un mod eficient toate regulile de asociere ordinale care există într-un set de date.

Algoritmul *DOAR* a fost extins în [CMC14a] spre algoritmul *DRAR* (*Discovery of Relational Association Rules*) pentru a găsi reguli de asociere relaţionale interesante de orice lungime într-un set de date.

### 3.2.2 Abordarea *SDDRAR*

Considerăm o mulţime de sisteme informatice bine-proiectate, notată cu $\mathcal{S}_{good}$. Folosind modelul teoretic din Secţiunea 3.1, considerăm că $\mathcal{DS} = \{ds_1, ds_2, \ldots, ds_n\}$ este setul de date compus din reprezentare $k$-dimensională a entităţilor din sistemele informatice din $\mathcal{S}_{good}$. Setul de caracteristici folosit pentru acest set de date este o mulţime a metricilor soft $\mathcal{SM} = \{sm_1, sm_2, ..., sm_k\}$ relevante pentru procesul de detectare a defectelor de proiectare.

Pentru a detecta clasele rău-structurate folosind reguli de asociere relaţionale, vom executa următorii paşi

1. Colectarea şi preprocesarea datelor
2. Construirea modelului *SDDRAR*
3. Testare

#### 3.2.2.1 Colectarea şi preprocesarea datelor

În cursul acestui pas, o analiză statistică este efectuată pe setul de date $\mathcal{DS}$ pentru a găsi o submulţime a caracteristicilor, care sunt relevante pentru această problema. Pentru a determina dependenţele dintre caracteristici folosim coeficientul de corelaţie Pearson [Tuf11]. Pentru fiecare metrică soft, $sm$, calculăm *corelaţia ei medie absolută* (notată cu $avg(sm)$) ca media corelaţiilor metricii cu restul metricilor. Calculăm media, $m$, şi deviaţia standard, $stdev$, a acestor valori, şi eliminăm acele metrici soft pentru care $|avg(sm) - m| > stdev$.

#### 3.2.2.2 Construirea modelului *SDDRAR*

Primul pas pentru a construi modelul *SDDRAR* este definirea relaţiilor posibile dintre metricile soft. Aceste relaţii sunt necesare pentru procesul de extragere a regulilor de asociere relaţionale. După ce relaţiile au fost definite, extragem toate regulile de asociere relaţionale, de orice lungime din setul de date $\mathcal{DS}$, având suport minimum $s_{min}$ şi confidenţă minimă $c_{min}$. Mulţimea regulilor este notată cu $\mathcal{RAR}$. Aceste reguli vor fi folosite pentru a identifica entităţile rău-proiectate din sistemul informatic.

#### 3.2.2.3 Testare

În cursul testării un nou sistem informatic, $\mathcal{S}_{new}$, trebuie analizat pentru a determina entităţile care au o proiectare greşită. Prima dată $\mathcal{S}_{new}$ este analizată şi reprezentarea $k$-dimensională a entităţilor din $\mathcal{S}_{new}$ este calculată. După aceea, următorii trei paşi sunt executaţi:

1. **Pasul de calcul al erorilor.** Pentru fiecare entitate $e_i$ din sistemul analizat, *numărul de erori*, $err(e_i)$, este calculat ca numărul de reguli de asociere relaţionale din $\mathcal{RAR}$ care nu sunt verificate de reprezentarea $k$-dimensională. După aceea, *procentul de erori*, $pe_{e_i}$, este calculată ca numărul de erori împărţite la numărul de reguli.

2. **Pasul de detectare.** După determinarea *numărului de erori* pentru fiecare entitate, raportăm ca entităţi cu posibile erori de proiectare acele entităţi care încalcă un număr destul de mare de reguli.

3. **Analiza defectelor.** Pentru entităţile raportate la pasul anterior, calculăm pentru fiecare metrică soft *numărul de erori*, ca numărul de reguli de asociere relaţionale binare din $\mathcal{RAR}$ care nu sunt respectate de entitatea respectivă. Aceste numere sunt analizate mai departe pentru a determina cauza defectului.

### 3.2.3 Evaluare experimentală

Pentru evaluarea experimentală a abordării *SDDRAR*, am ales ca entităţi clasele dintr-un sistem informatic. După cum a fost precizat în Secţiunea 3.1, pentru a folosi modelul vectorial avem nevoie de o mulţime de metrici soft relevante pentru a caracteriza proiectarea unei clase. Am ales să folosim aceeaşi 16 metrici care au fost folosite şi pentru abordarea noastră bazată pe metrici agregate (Secţiunea 2.2.2).

#### 3.2.3.1 Date de antrenament

Pentru pasul de antrenament este nevoie de o mulţime de sisteme informatice bine-proiectate, $\mathcal{S}_{good}$, de unde reprezentarea $k$-dimensională a claselor este calculată. Pentru implementarea curentă a abordării noastre am ales un singur sistem informatic bine-proiectat, sistemul disponibil public *JHotDraw*, versiunea 5.1 [Gam].

Pentru primul pas al abordării *SDDRAR* prima dată am analizat sistemul informatic *JHotDraw* şi am extras din el setul de date $\mathcal{DS}$, alcătuit din reprezentarea 16-dimensională a claselor. Am scalat valoarea metricilor să fie în intervalul [0,1], şi am efectuat analiza statistică care a dus la eliminarea a patru metrici soft: *CA*, *NOM*, *RFC* şi *TCC*.

Al doilea pas al abordării *SDDRAR* constă în extragerea din setul de date $\mathcal{DS}$ a regulilor de asociere relaţionale interesante, de orice lungime, având suport minim $s_{min} = 0.9$ şi confidenţă minimă $c_{min} = 0.85$.

#### 3.2.3.2 Studii de caz

Al treilea pas al abordării *SDDRAR* este *Testarea*, unde modelul *SDDRAR*, construit la pasul anterior, este folosit pentru a detecta entităţile rău-proiectate într-un sistem informatic. Pentru acest pas am considerat şase studii de caz, dintre care două au fost exemple simple, iar celelalte patru au fost sisteme informatice disponibile public pe pagina *SourceForge*. Rezultatele pentru cele şase studii de caz sunt prezentate în continuare:

- Primul studiu de caz simplu preluat din [SSL01] - abordarea *SDDRAR* a identificat clasa cu defect de proiectare.

- Al doilea studiu de caz simplu preluat din [Fow99] pagini 22-26 - abordarea *SDDRAR* a identificat clasa cu defect de proiectare.
- Sistemul informatic disponibil public *FTP4J* [FTP13] versiuni 1.5, 1.5.1, 1.6 şi 1.6.1 - abordarea *SDDRAR* a identificat clasa *FTPClient* pentru toate versiunile. Analiza noastră a arătat că această clasă are un defect de proiectare, este *God Class*, deci abordarea *SDDRAR* a identificat-o corect.
- Sistemul informatic disponibil public *ISO8583* [ISO13] versiuni 1.5.2, 1.5.3 şi 1.5.4 - abordarea *SDDRAR* a identificat clasa *MessageFactory* pentru toate versiunile. Analiza noastră a arătat că această clasă are un defect de proiectare, este *God Class*, deci abordarea *SDDRAR* a identificat-o corect.
- Sistemul informatic disponibil public *Profiler4J-Agent* [Pro13] versiuni 1.0-alpha5, 1.0-alpha6, 1.0-alpha7 şi 1.0-beta1 - pentru versiunea 1.0-alpha5 clasa *Server* este raportată; pentru versiunea 1.0-alpha6 clasa *MemoryInfo* este raportată; pentru ultimele două versiuni clasa *Config* este raportată. Analiza noastră a arătat că *Server* este o clasa foarte cuplată, iar clasele *MemoryInfo* şi *Config* sunt *Data Class*, deci au fost identificare corect de abordarea *SDDRAR*.
- Sistemul informatic disponibil public *WinRun4J* [Win13] versiuni 0.4.0, 0.4.1, 0.4.2, 0.4.3 şi 0.4.4. - abordarea *SDDRAR* a identificat clasa *NativeBinder* pentru toate versiunile. Analiza noastră a arătat că această clasa are anumite probleme de proiectare, dar nu are un singur defect serios, deci am considerat că nu se poate decide clar dacă identificarea ei este corectă sau nu.

### 3.2.4 Analiză şi comparaţie cu abordări existente

În această subsecţiune prezentăm o analiză din mai multe puncte de vedere a abordării *SDDRAR* şi o comparaţie cu alte abordări similare din literatura de specialitate.

**Precizia de detectare.** Am considerat că pentru primele 5 studii de caz precizia este 1 (clasa identificată are defecte de proiectare), iar pentru ultimul am considerat o precizie de 0.5. Astfel, *precizia de detectare* medie este 0.917 pentru cele şase studii de caz.

**Măsura de media erorilor.** În afara determinării entităţilor cu posibile defecte de proiectare, abordarea *SDDRAR* a fost folosită pentru un studiu în modul următor. Pentru fiecare versiune a sistemelor informatice disponibile public am calculat media numărului de erori a claselor. Această valoare scade (sau rămâne la fel) pentru fiecare versiune nouă a sistemului. Acest lucru confirmă că valori mai mici pentru erori corespund unor proiectări mai bune, pentru că ne aşteptăm ca versiunile mai mari să aibă o proiectare mai bună.

**Robusteţe.** Procesul de detectare a entităţilor rău-structurate depinde de entităţile din setul de date $\mathcal{DS}$. Aceste entităţi sunt considerate bine-proiectate, dar trebuie să studiem ce se întâmplă dacă introducem zgomot în aceast set de date. Am adăugat nişte entităţi rău-proiectate în $\mathcal{DS}$ şi am aplicat metoda *SDDRAR* la studii de caz prezentate mai sus. Aşa cum ne-am aşteptat, metoda *SDDRAR* este robustă, prezenţa unei cantităţi mici de zgomot nu influenţează rezultatele ei.

**Mulţimea relaţiilor.** Ultima perspectivă constă în analizarea mulţimii de relaţii $\mathcal{R}$ considerate pentru extragerea regulilor de asociere relaţionale. Am efectuat nişte experimente cu combinaţii diferite a relaţiilor posibile şi am ajuns la concluzia că şi aceste combinaţii pot identifica defecte de proiectare în seturi de date.

#### 3.2.4.1 Comparaţie cu abordări existente

Pentru a compara abordarea noastră cu alte abordări din literatura de specialitate, am ales două instrumente disponibile public: *JDeodorant* un *plugin Eclipse* [JDe13] şi *iPlasma*

| | Ftp4J | ISO8583 | WinRun4J |
|---|---|---|---|
| *SDDRAR* | FtpClient | MessageFactory | NativeBinder |
| JDeodorant | FtpClient (36)<br>NVTASCIIWriter (0)<br>NVTASCIIReader (0)<br>FTPProxyConnector (2)<br>FTPCommunication -<br>- Channel (0) | ISOMessage (28)<br>MessageFactory (38) | Launcher (28)<br>FileAssociation (23)<br>RegistryKey (0) |
| iPlasma | FTPClient -<br>Brain Class (36)<br>FTPFile -<br>Data Class (19) | MessageFactory -<br>God Class (38) | Closure -<br>God Class (27)<br>FileVerb -<br>Data Class (26)<br>NativeBinder -<br>God Class (31) |

Tabela 3.1: Clasele raportate de JDeodorant, iPlasma şi metoda *SDDRAR*.

un instrument prezentat în [Mar02] şi disponibil la [IPl13]. Clasele cu defecte de proiectare raportate de cele două instrumente şi abordarea noastră pentru prima versiune a trei studii de caz sunt prezentate în Tabelul 3.1. Pentru *JDeodorant* am considerat doar defectele *God Class*. Pentru *iPlasma* am specificat după fiecare clasă numele defectului. După fiecare clasă, între paranteze am adăugat numărul de erori raportate de abordarea *SDDRAR* pentru clasa respectivă.

În Tabelul 3.1 se poate observa că nici măcar cele două instrumente nu sunt de acord în legătură cu care clase au defecte de proiectare şi care nu au. Ca şi majoritatea abordărilor, am decis să analizăm manual clasele raportate, pentru a decide care intrument să fie folosit pentru comparaţie. În cazul instrumentului *iPlasma* suntem de acord cu clasele raportate. Instrumentul *JDeodorant* pare că încearcă să găsească restructurări *Extract Class*, nu *God Class*.

Comparând rezultatele raportate de instrumentul *iPlasma* cu rezultatele metodei *SDDRAR*, putem vedea că clasele raportate de *SDDRAR* sunt raportate şi de *iPlasma*, iar pentru clasele raportate doar de *iPlasma* metoda noastra raportează un număr mare de erori (chiar dacă nu suficient de mare pentru a raporta clasa ca fiind defectivă). Folosind rezultatele raportate de *iPlasma* am calculat *recall*-ul pentru abordarea *SDDRAR*, care este 0.85.

### 3.2.5 Studiu despre variaţia parametrilor

Abordarea *SDDRAR* depinde de o serie de parametrii, valoarea cărora influenţează rezultatele raportate, astfel am efectuat un studiu pentru a investiga următoarele aspecte:

- Folosirea valorilor originale sau valorilor normalizate pentru metrici soft.
- Folosirea regulilor de asociere relaţionale binare sau de orice lungime.
- Folosirea doar regulilor de asociere relaţionale maximale, sau a tuturor regulilor extrase.
- Efectul modificării parametrului $\tau$ şi a confidenţei minime pe acurateţea detectării.

Ca rezultat al studiului am ajuns la concluzia că este mai bine să folosim valori normalizate pentru metrici soft, să extragem reguli de asociere relationale de orice lungime, şi să folosim doar reguli maximale. Pentru ultimul aspect, dacă valoarea lui $\tau$ este scăzută, mai multe clase sunt raportate ca fiind defective, şi majoritatea lor într-adevăr a avut mai multe sau

mai puţine defecte. Scăzând valoarea minimă pentru confidenţă mai multe clase au fost raportate, dintre care multe erau *Data Class*.

## 3.3 Predicţia Defectelor

Această secţiune prezintă abordarea noastră, numită *DPRAR - Defect Prediction using Relational Association Rules*, de învăţare supervizată pentru predicţia defectelor într-un sistem informatic [CMC14b]. Această abordare foloseşte modelul teoretic prezentat în Secţiunea 3.1 şi algoritmul *DRAR* pentru extragerea regulilor de asociere relaţionale.

În abordarea noastră, problema de *predicţia defectelor* este considerată o problemă de clasificare binară supervizată, deoarece fiecare entitate trebuie clasificată *defectivă*, notată cu "+" sau *non-defectivă* notată cu "-". Similar altor abordări, modelul nostru este construit folosind date de antrenament, adică un set de date care conţine entităţi despre care se ştie dacă sunt defective sau nu.

### 3.3.1 Classificatorul *DPRAR*

Deoarece predicţia defectelor este o problemă de clasificare supervizată, vom avea un pas de antrenament şi un pas de testare. Pentru antrenament considerăm două seturi de date: $DS_+$ care conţine entităţi $k$-dimensionale defective şi $DS_-$ care conţine entităţi $k$-dimensionale non-defective. Pentru a clasifica o entitate ca fiind defectivă sau non-defectivă, următorii paşi sunt efectuaţi:

1. Preprocesarea datelor
2. Antrenament/Construirea clasificatorului *DPRAR*
3. Testare/Clasificare

#### 3.3.1.1 Preprocesarea datelor

În cadrul acestui pas, datele de antrenament sunt scalate ca să aibă valori în intervalul [0,1] şi o analiză statistică este efectuată pe seturile de antrenament $DS_+$ şi $DS_-$ pentru a determina o submulţime a caracteristicilor care sunt corelate cu variabila de prezis. Pentru a determina dependenţele dintre caracteristici şi variabila ţintă, se foloseşte coeficientul de corelaţie Spearman [Spe04].

Pentru a decide care caracteristici ar trebui eliminate, pentru fiecare caracteristică (metrică soft) $sm_i \in \mathcal{SM}$ calculăm corelaţia Spearman, $(cor(sm_i, target))$, între valorile caracteristicii şi a variabilei ţintă. Vom nota cu $m$ media şi cu $stdev$ deviaţia standard acestor valori de corelaţie. O caracteristică $sm_i$ este eliminată dacă $abs(cor(sm_i, target)) < m - stdev$.

#### 3.3.1.2 Antrenament/Construirea clasificatorului *DPRAR*

Prima dată definim mulţimea relaţiilor dintre caracteristici care va fi folosită în procesul de extragere a regulilor de asociere relaţionale. După aceea efectuăm extragerea de reguli de asociere relaţionale separat pe cele două seturi de date. Astfel vom avea două seturi de reguli de asociere relaţionale: $RAR_+$ şi $RAR_-$. La fiecare regulă $r$ asociem o valoare, numită $ratio(r)$, calculată prin împărţirea confidenţei regulii la suportul ei.

#### 3.3.1.3 Testare/Clasificare

La pasul de clasificare, după ce antrenamentul s-a terminat şi clasificatorul *DPRAR* a fost construit, o entitate nouă $e$ trebuie clasificată. Pentru clasificare am calculat două scoruri

| Nume | Entităţi defective | Entităţi non-defective |
|---|---|---|
| CM1 | 42 (12.84 %) | 285 (87.16%) |
| KC1 | 314 (26.54%) | 869 (73.46%) |
| KC3 | 36 (18.56 %) | 158 (81.44 %) |
| PC1 | 61 (8.65 %) | 644 (91.35%) |
| JM1 | 1672 (21.49 %) | 6110 (78.51 %) |
| MC2 | 44 (35.2 %) | 81 (64.8%) |
| MW1 | 27 (10.67%) | 226 (89.33 %) |
| PC2 | 16 (2%) | 729 (98%) |
| PC3 | 134 (12.4%) | 943 (87.6 %) |
| PC4 | 177 (13.8%) | 1110 (86.2 %) |

Tabela 3.2: Seturile de date *NASA* folosite pentru evaluarea experimentală.

| Set de date | $c^+_{min}$ | $c^-_{min}$ | Lungime | Acc | Pd | Spec | Prec | AUC |
|---|---|---|---|---|---|---|---|---|
| CM1 | 0.927 | 0.94 | orice | 0.8716 | 0.929 | 0.8632 | 0.5 | 0.896 |
| KC1 | 0.8 | 0.822 | 2 | 0.823 | 0.818 | 0.825 | 0.628 | 0.822 |
| KC3 | 0.885 | 0.96 | 2 | 0.83 | 0.889 | 0.8165 | 0.5246 | 0.85225 |
| PC1 | 0.95 | 0.995 | 2 | 0.956 | 0.885 | 0.963 | 0.692 | 0.924 |
| JM1 | 0.95 | 0.995 | orice | 0.96 | 0.842 | 0.992 | 0.967 | 0.917 |
| MC2 | 0.96 | 0.99 | orice | 0.896 | 0.773 | 0.9632 | 0.919 | 0.868 |
| MW1 | 0.97 | 0.975 | orice | 0.941 | 0.889 | 0.947 | 0.667 | 0.918 |
| PC2 | 0.95 | 0.995 | orice | 0.984 | 0.938 | 0.985 | 0.577 | 0.962 |
| PC3 | 0.95 | 0.995 | 2 | 0.967 | 0.85 | 0.983 | 0.877 | 0.917 |
| PC4 | 0.95 | 0.995 | 2 | 0.961 | 0.814 | 0.985 | 0.894 | 0.899 |

Tabela 3.3: Rezultatele obţinute pentru clasificatorul *DPRAR* pentru seturile de date considerate.

$score_+(e)$ (similaritatea lui $e$ cu clasa defectivă) şi $score_-(e)$ (similaritatea lui $e$ cu clasa non-defectivă). Pentru a calcula $score_+(e)$ considerăm *ratio*-ul mediu al regulilor din $RAR_+$ care sunt verificate de $e$, dar şi *ratio*-ul mediu al regulilor din $RAR_-$ care nu sunt verificate de $e$. $Score_-(e)$ este calculat în mod similar.

Dacă $score_+ > score_-$ atunci $e$ va fi clasificată defectivă, altfel va fi clasificată non-defectivă.

### 3.3.2 Evaluare Experimentală

Pentru evaluarea experimentală a abordării noastre am folosit zece seturi de date *NASA* [Nas]. Numele si caracteristicile acestor seturi de date sunt prezentate în Tabelul 3.2.

Pentru evaluarea performanţei clasificatorului *DPRAR*, o validare încrucişată de tip *leave-one-out* a fost folosită, şi următoarele măsuri de performanţă au fost calculate: acurateţea (Acc), probabilitatea de detecţie (Pd), *specificity* (Spec), precizie (Prec), şi $AUC$. Tabelul 3.3 prezintă cele mai bune rezultate obţinute de clasificatorul *DPRAR* pentru seturile de date considerate.

### 3.3.3 Analiză şi comparaţie cu abordări existente

Am comparat rezultatele clasificatorului *DPRAR* cu rezultate raportate în literatura de specialitate pentru alte abordări pentru care evaluarea experimentală a fost făcută folosind aceleaşi seturi de date. Am ales metoda $CBA2$ [BDVB11], clasificatorul $1R$ [CBYP05], $Bagging$ [HDF12] şi $EDER-SD$ [RRRAR12].

Nu toate abordările au fost testate pe toate cele 10 seturi de date pe care le-am folosit, şi nu toate abordările raportează toate măsurile de performanţă pe care le-am folosit. Considerând toate studiile de caz şi toate măsurile de performanţă, *DPRAR* a avut performanţe mai bune în 45 de cazuri, similară o dată şi performanţe mai slabe în 23 de cazuri dintr-un total de 69 de comparaţii. În plus, măsura $AUC$ (considerată în literatură una dintre cele mai bune măsuri de evaluare pentru clasificatori) pentru clasificatorul *DPRAR* depăşeşte media valorilor $AUC$ raportate de celelalte metode la toate studiile de caz. Acest lucru indică eficienţa bună a clasificatorului *DPRAR*.

### 3.3.4 Un studiu despre clasificatorul *DPRAR*

Am efectuat un studiu asupra clasificatorului *DPRAR* pentru a analiza cum influenţează eliminarea diferitor caracteristici rezultatele raportate, şi care sunt efectele folosirii unei alte formule pentru a calcula acele scoruri pe baza cărora o entitate nouă este clasificată.

Primul pas al clasificatorului *DPRAR* este preprocesarea datelor, când diferite metrici soft pot fi eliminate. Pentru metoda *DPRAR* am ales să eliminăm acele caracteristici pentru care corelaţia Spearman cu valoarea ţintă este mai mică decât diferenţa dintre media corelaţiilor, $m$, şi deviaţia standard a corelaţiilor, $stdev$. În studiul nostru am comparat această metodă de eliminare a caracteristicilor cu trei alte metode (nicio caracteristică nu este eliminată, sunt eliminate caracteristicile ale căror corelaţie este mai mare decât $m + stdev$, sunt eliminate şi caracteristicile ale căror corelaţie este mai mică decât $m - stdev$ şi caracteristicile ale căror corelaţie este mai mare decât $m + stdev$). Pentru evaluarea experimentală am folosit setul de date PC3.

Măsurile de evaluare folosite au sugerat că cea mai bună opţiune este eliminarea caracteristicilor care au corelaţia sub $m - stdev$ sau peste $m + stdev$, iar cea mai slabă variantă este să nu fie folosită nicio metodă de eliminare.

Am efectuat încă un studiu, în legătură cu calculul scorului, folosind ca studiu de caz setul de date PC3, şi folosind ca metodă de eliminare a caracteristicilor varianta cu cele mai bune rezultate din primul studiu. De data asta am folosit numărul regulilor de asociere relaţionale (şi nu *ratio*-ul lor) verificate şi neverificate pentru a calcula scorul pe baza căruia se face clasificarea. Măsurile de performanţă au sugerat că acestă variantă pentru calculul scorului este mai bună.

Am comparat rezultatele acestor două variante de a calcula scorul şi cu metode din literatura de specialitate: *CBA2*, *ROCUS*, *Random Forests* şi *Dynamic AdaBoost.NC*. Clasificatorul *DPRAR* cu calculul scorului bazat pe numărul de reguli a avut cele mai bune rezultate, urmat de *DPRAR* cu calculul scorului bazat pe *ratio*-ul regulilor.

## 3.4 Concluzii şi cercetări ulterioare

Acest capitol a prezentat abordările noastre bazate pe extragerea regulilor de asociere relaţionale pentru detectarea defectelor în sisteme informatice. Acest domeniu are două direcţii principale, *detectarea defectelor de proiectare* şi *predicţia defectelor*, ambele fiind acoperite în cadrul cercetării prezentate în acest capitol.

În prima parte a capitolului am prezentat abordarea *SDDRAR*, care poate să identifice entităţile cu defecte de proiectare într-un sistem informatic. Abordarea a fost evaluată folosind şase studii de caz şi analiza manuală a rezultatelor a arătat că *SDDRAR* e capabil să

identifice entităţile cu defecte de proiectare. Am comparat abordarea noastră cu alte două abordări din literatura de specialitate, şi rezultatele au fost similare cu rezultatele raportate de una dintre instrumentele folosite. Aceste rezultate arată potenţialul abordării propuse.

În partea a doua a capitolului am prezentat abordarea *DPRAR*, un model de clasificare binară, bazată pe reguli de asociere relaţionale pentru predicţia defectelor. Pentru evaluarea experimentală am folosit zece seturi de date *NASA*. Comparaţia rezultatelor cu alte rezultate raportate în literatura de specialitate arată că abordarea *DPRAR* este comparabilă sau mai bună decăt metodele existente.

Pentru cercetări ulterioare vrem să extindem pe alte seturi de date sau sisteme informatice evaluarea experimentală a acestor metode. Deasemenea, vrem să extindem abordările noastre spre reguli de asociere relaţionale *fuzzy*.

# Capitolul 4

# Un Framework pentru Analiza Sistemelor Informatice Orientate Obiect

În capitolele anterioare am prezentat tehnici bazate pe inteligenţă computaţională pentru a rezolva diferite probleme de o importanţă ridicată pe parcursul întreţinerii şi evoluţiei sistemelor informatice. Pentru dezvoltarea acestor tehnici am folosit *framework*-ul *FAOS* (*Framework for Analyzing Object-oriented Software systems*) care a fost proiectat ca să fie suficient de general pentru a oferi un suport pentru analiza unui sistem informatic orientat obiect şi extragerea informaţiilor relevante din el, precum şi pentru a oferi implementări pentru mai multe metrici soft care pot fi folosite pentru a măsura calitatea sistemului informatic. În acest capitol prezentăm acest *framework* mai detaliat.

## 4.1 *Framework*-ul FAOS

*Framework*-ul *FAOS* este alcătuit din trei module, *Analyzer*, *Metrics* şi *PackageRestructuring*.

**Modulul *Analyzer*.** Primul modul din *FAOS* este *Analyzer*, implementarea căruia poate să analizeze un sistem informatic realizat în limbajul de programare Java, şi necesită ori fişierele compilate *.class* ori o arhivă *jar* pentru această analiză. Am definit un model de date care este alcătuit din clase care reprezintă atribute, metode, clase din sistemul analizat. Scopul principal al modulului *Analyzer* este să efectueze analiza sistemului informatic şi să extragă o listă de clase împreună cu metodele şi atributele lor, şi relaţiile între ele. Pentru analiza codului compilat am folosit *framework*-ul de manipulare a *bytecode*-ului *ASM* [ASM13].

**Modulul *Metrics*.** Al doilea modul din *FAOS* este *Metrics*, un modul care conţine implementarea diferitor metrici soft. Modulul este împărţit în două pachete principale, pachetul *classLevel* conţine implementarea a 17 metrici soft, calculate pentru o clasă, iar pachetul *packageLevel* conţine implementarea a 20 de metrici şi măsuri soft calculate pentru un pachet.

**Modulul *PackageRestructuring*.** Al treilea modul din *FAOS* se numeşte *packageRestructuring* şi abordarea de restructurare la nivel de pachete prezentată în Secţiunea 2.1 este implementată aici, de aceea este un modul mai puţin abstract ca primele două.

## 4.2 Comparaţie cu framework-uri similare

În inginerie soft bazată pe inteligenţă computaţională cercetătorii adeseori implementează abordarea lor sub forma unor instrumente şi *framework*-uri, care pot fi folosite atât de alţi

cercetători pentru comparaţia rezultatelor, cât şi de dezvoltatori soft. Cu toate că nu există un alt *framework* cu exact aceeaşi funcţinalităţi ca *FAOS*, există anumite instrumente care sunt similare cu diferite părţi din el, de exemplu *JDeodorant* - un *plugin Eclipse* pentru a identifica *bad smell*-uri în sisteme informatice -, *iPlasma* - un mediu proiectat pentru analiza calităţii sistemelor orientate obiect - şi instrumentul *Bunch* care încearcă să găsească o partiţie bună a unui sistem informatic folosind algoritmi de căutare. Primii doi au propria reprezentare internă (similară cu modelul nostru de date), iar *Bunch* foloseşte *Module Dependency Graph*-ul sistemului de restructurat. Astfel *Bunch* este independent de orice limbaj de programare, *JDeodorant* analizează sisteme realizate în limbajul Java şi *iPlasma* poate să analizeze sisteme realizate în Java sau C++. Avantajul lor faţă de *framework*-ul *FAOS* este că sunt instrumente complete, gata de a fi folosite. Pe de altă parte *FAOS* este mai flexibil, poate fi extins uşor pentru a efectua alte tipuri de analize, nu numai cele implementate momentan.

În concluzie, *JDeodroant* şi *iPlasma* sunt potrivite pentru situaţii când cineva vrea să analizeze codul sursă, să caute metrici soft cu valori anormale sau *bad smell*-uri. *Framework*-ul *FAOS* este mai potrivit pentru situaţii când cineva vrea să analizeze un sistem şi să facă ceva cu lista elementelor extrase. Astfel, avantajul principal al *framework*-ului *FAOS* este că poate fi extins uşor: noi metrici soft pot fi adăgate, lista entităţilor returnate de modulul *Analyzer* poate fi folosită pentru analize ulterioare. Aşa s-a întâmplat şi la modulul *PackageRestructure* unde această listă este folosită pentru a găsi o împărţire bună a claselor în pachete.

## 4.3 Concluzii şi cercetări ulterioare

În acest capitol am prezentat *framework*-ul *FAOS*, un *framework* pentru analiza sistemelor informatice orientate obiect. *FAOS* a fost dezvoltat pentru a sprijini evaluarea experimentală a majorităţii abordărilor bazate pe instruire automată prezentate în capitolele anterioare. Astfel, modulul *Analyzer* a fost folosit pentru toate abordările cu excepţia predicţiei de defecte, unde seturile de date *NASA* sunt folosite. Modulul *Metrics* a fost folosit pentru restructurare la nivel de pachete şi clase dar şi la abordarea de detectare a defectelor de proiectare. În cele din urmă, modulul *PackageRestrucuring* conţine implementarea abordării noastre pentru restructurarea la nivel de pachete a unui sistem informatic.

Ca cercetare ulterioată dorim să facem acest *framework* disponibil public, ca alte persoane să poată să-l folosească şi să-l extindă în funcţie de context. Pentru a facilita folosirea lui, ne gândim să dezvoltăm un *plugin Eclipse* bazat pe *FAOS*. Deasemenea dorim să extindem partea de analiză la alte limbaje de programare, deoarece dacă reprezentarea internă a entităţilor este construită, restul *framework*-ului poate fi folosite fără probleme.

# Concluzii

În această teză am prezentat lucrările noastre originale pentru aplicarea metodelor şi algoritmilor de instruire automată pentru rezolvarea diferitor probleme din inginerie soft. Asemenea abordări inteligente sunt necesare pentru că în zilele noastre sistemele informatice sunt destul de complexe, alcătuite din multe componente şi relaţii între aceste componente şi abordările care sugerează care sunt acele părţi care necesită atenţie pot fi foarte utile dezvoltatorilor soft. Prin urmare, putem spune că asemenea abordări pot fi considerate instrumente importante şi sunt benefice dezvoltatorilor soft.

Am ales două probleme din ingineria soft care au reprezentat principalele direcţii de cercetare în această teză. În cadrul fiecărei direcţii am lucrat la două probleme separate. Prima direcţie principală este folosirea algoritmilor de *clustering* pentru remodularizare soft atât la nivel de pachete, cât şi la nivel de clasă. A doua direcţie principală a fost folosirea regulilor de asociere relaţionale pentru detectarea defectelor în sisteme informatice. În cadrul acestei direcţii am lucrat la problema de detectare a defectelor de proiectare şi predicţia defectelor în sisteme informatice. În cele din urmă am prezentat o contribuţie spre dezvoltarea sistemelor informatice, *framework*-ul *FAOS*, dezvoltat pentu analiza sistemelor informatice orientate obiect.

Evaluarea experimentală a abordărilor dezvoltate pentru prima direcţie de cercetare au arătat că tehnici de *clustering*, combinate cu o măsură bună a similarităţii/disimilarităţii dintre entităţi poate să identifice o împărţire bună a claselor în pachete, sau o împărţire bună a metodelor şi atributelor în clase.

În a doua direcţie principală de cercetare am folosit reguli de asociere relaţionale pentru detectarea defectelor în sisteme informatice. Prima problemă în cadrul acestei direcţii a fost identificarea claselor cu defecte de proiectare. Experimentele efectuate arată că acele clase care au fost identificate de abordarea noastră au avut defecte de proiectare. În ceea ce priveşte predicţia defectelor, o problemă de clasificare, abordarea noastră a fost testată pe 10 seturi de date *NASA* [Nas]. Am efectuat o comparaţie cu alte abordări folosind multe seturi de date şi măsuri de performanţă, care a arătat că performanţa clasificatorului nostru este comparabilă sau mai bună decât metodele existente, demonstrând potenţialul abordării noastre.

*Framework*-ul *FAOS* prezentat în această teză a fost dezvoltat pentru a sprijini evaluarea experimentală a abordărilor propuse în această tezû. Cu toate acestea, a fost proiectat să fie general şi extensibil ca să poată fi folosit şi de alţii. *Framework*-ul oferă suport pentru analiza unui sistem informatic, pe parcursul căreia o reprezentare internă a sistemului analizat este construită. Aceasta poate fi folosită ulterior pentru diferite probleme, problemele implementate momentan permit calculul valorilor diferitor metrici soft şi restructurarea la nivel de pachete a unui sistem informatic.

În ceea ce priveşte cercetările ulterioare, intenţionăm să lucrăm la neajunsurile abordărilor noastre pentru a le îmbunătăţi. De asemenea dorim să extindem evaluarea experimentală, să folosim alte sisteme informatice sau seturi de date. Vom considera aplicarea variantelor *fuzzy* ale abordărilor propuse, acolo unde este posibil. În cele din urmă vrem să dezvoltăm abordări noi şi pentru alte probleme din ingineria soft.

# Bibliografie


[AAH12] H. H. Ammar, W. Abdelmoez, and M. S. Hamdi. Software engineering using artificial intelligence techniques: Current state and open problems. In *Proceedings of First Taibah University International Conference on Computing and Information Technology*, 2012.

[AAM11] A. Alkhalid, M. Alshayeb, and S.A. Mahmoud. Software refactoring at the package level using clustering techniques. *IET Software*, 5(3):274–286, 2011.

[AIS93] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216, 1993.

[AL99] Nicolas Anquetil and Timothy Lethbridge. Experiments with clustering as a software remodularization method. In *Proceedings of 6th Working Conference on Reverse Engineering*, pages 235–255, Atlanta, USA, October 1999.

[AS94] R. Agrawal and R. Srikant. Fast algorithm for mining association rules. In *Proceeding of the 20th VLDB Conference*, pages 487–499, 1994.

[ASM13] ObjectWeb: Open Source Middleware, 2013. http://asm.objectweb.org/.

[BDVB11] Ma Baojun, Karel Dejaeger, Jan Vanthienen, and Bart Baesens. Software defect prediction based on association rule classification. Technical report, Katholieke Universiteit Leuven, February 2011.

[BJWD99] L.C. Briand and and al. J. W. Daly. A unified framework for coupling measurement in object-oriented systems. 25(1):91–121, 1999.

[BK95] J. M. Bieman and B. K. Kang. Cohesion and reuse in an object-oriented system. *ACM SIGSOFT Software Engineering Notes*, 20(SI):259–262, 1995.

[BLMO10] Gabriele Bavota, Andrea De Lucia, Andrian Marcus, and Rocco Oliveto. Software remodularization based on structural and semantic metrics. In *17th Working Conference on Reverse Engineering*, pages 195–204, 2010.

[BMW02] Lionel C. Briand, Walcelio L. Melo, and J. Wust. Assessing the applicability of fault-proneness models across object-oriented software projects. *IEEE Trans. Softw. Eng.*, 28(7):706–720, July 2002.

[BOL+10] Gabriele Bavota, Rocco Oliveta, Andrea De Lucia, Giuliano Antoniol, and Yann-Gaël Guéhéneuc. Playing with refactoring: Identifying extract class opportunities through game theory. In *Software Maintenance, 2010 IEEE International Conference*, pages 1–5, 2010.

[CBYP05] Venkata U. B. Challagulla, Farokh B. Bastani, I-Ling Yen, and Raymond A. Paul. Empirical assessment of machine learning based software defect prediction techniques. In *Proceedings of the 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems*, WORDS '05, pages 263–270, Washington, DC, USA, 2005. IEEE Computer Society.

[CK91] Shyam R. Chidamber and Chris F. Kemerer. Towards a metrics suite for object-oriented design. In *Conference Proceedings on Object Oriented Programming Systems, Languages, and Applications*, pages 197–211, 1991.

[CK94] Shyam R. Chidamber and Chris F. Kemerer. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493, June 1994.

[CLMM05] Yania Crespo, Carlos López, Esperanza Manso, and Raúl Marticorena. Language independent metric support towards refactoring inference. In *Proceedings of the 9th Workshop on QAOOSE*, pages 18–29, 2005.

[CMC14a] Gabriela Czibula, Zsuzsanna Marian, and Istvan Gergely Czibula. Detecting software design defects using relational association rule mining. *Knowledge and Information Systems*, 2014. DOI: 10.1007/s10115-013-0721-z (available online at http://link.springer.com/article/10.1007/s10115-013-0721-z.

[CMC14b] Gabriela Czibula, Zsuzsanna Marian, and Istvan Gergely Czibula. Software defect prediction using relational association rule mining. *Information Sciences*, 264:260–278, 2014.

[CS06] I.G. Czibula and G. Serban. Improving systems design using a clustering approach. *International Journal of Computer Science and Network Security*, 6(12):40–49, 2006.

[CS07] Istvan Gergely Czibula and Gabriela Serban. Hierarchical clustering for software systems restructuring. *INFOCOMP Journal of Computer Science*, 6(4):43–51, 2007.

[CSTM06] Alina Campan, Gabriela Serban, Traian Marius Truta, and Andrian Marcus. An algorithm for the discovery of arbitrary length ordinal association rules. In *Proceedings of the 2006 International Conference on Data Mining (DMIN)*, pages 107–113, 2006.

[DABH11] Stéphane Ducasse, Nicolas Anquetil, Usman Bhatti, and Andre Cavalcante Hora. Software metrics for package remodularization. Technical report, Institut National de Recherche en Informatique et en Automatique, 2011.

[DbU] Commons dbutils. http://commons.apache.org/proper/commons-dbutils/index.html.

[DDN00] Serge Demeyer, Stephance Ducasse, and Oscar Nierstrasz. Finding refactorings via change metrics. In *Proceedings of the ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages and Applications*, pages 166–177, 2000.

[DG77] Persi Diaconis and R. L. Graham. Spearman's footrule as a measure of disarray. *Journal of the Royal Statistical Society*, 39(2):262–268, 1977.

[DK05] Scott Dick and Abraham Kandel. *Computational Intelligence in Software Quality Assurance*. Series in Machine Perception and Artificial Intelligence. World Scientific Publishing, 2005.

[EL] El. http://commons.apache.org/proper/commons-el/.

[Ema] Email. http://commons.apache.org/proper/commons-email/.

[FBZ12] Francesca Arcelli Fontana, Pietro Braione, and Marco Zanoni. Automatic detection of bad smells in code: An experimental assessment. *Journal of Object Technology*, 11:5:1–38, 2012.

[Fow] Martin Fowler. Refactoring malapropism. http://martinfowler.com/bliki/RefactoringMalapropism.html.

[Fow99] Martin Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, Boston, 1999.

[FTCS09] Marios Fokaefs, Nikolaos Tsantalis, Alexander Chatzigeorgiu, and Jorg Sander. Decomposing object-orietend class modules using an agglomerative clustering technique. In *Proceedings of International Conference on Software Maintenance*, pages 93–101, Edmonton, Canada, 2009.

[FTP13] Ftp4j, 2013. http://sourceforge.net/projects/ftp4j/.

[FTSC11] Marios Fokaefs, Nikolaos Tsantalis, Eleni Stroulia, and Alexander Chatzigeorgiou. Jdeodorant: Identification and application of extract class refactorings. In *Proceedings of the 33rd International Conference on Software Engineering, ICSE*, pages 1037–1049, 2011.

[Gam] E. Gamma. JHotDraw Project. http://sourceforge.net/projects/jhotdraw.

[GMCS04] Lan Guo, Yan Ma, Bojan Cukic, and Harshinder Singh. Robust prediction of fault-proneness by random forests. In *ISSRE*, pages 417–428, 2004.

[GWI11] Isaac Griffith, Scott Wahl, and Clemente Izurieta. Truerefactor: An automated refactoring tool to improve legacy system and application comprehensibility. In *Proceedings of the ISCA 24th International Conference on Computer Applications in Industry and Engineering*, pages 316–321, 2011.

[Han05] Jiawei Han. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[Har12] Mark Harman. The role of artificial intelligence in software engineering. In *1st International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering*, 2012.

[HDF12] A.A. Shahrjooi Haghighi, M. Abbasi Dezfuli, and S.M. Fakhrahmad. Applying mining schemes to software fault prediction: A proposed approach aimed at test cost reduction. In *Proceedings of the World Congress on Engineering 2012 Vol I*, WCE 2012,, pages 1–5, Washington, DC, USA, 2012. IEEE Computer Society.

[HJ01] Mark Harman and Bryan F. Jones. Search-based software engineering. *Information and Software Technology*, 43(14):833–839, 2001.

[HK81] S. Henry and D. Kafura. Software structure metrics based on information flow. *IEEE Transactions on Software Engineering*, 7(5):510–518, September 1981.

[HKI08] Yoshiki Higo, Shinji Kusumoto, and Katsuro Inoue. A metric-based approach to identifying refactoring opportunities for merging code clones in a java software system. *Journal of Software Maintenance and Evolution: Research and Practice*, 20:435 – 461, 2008.

[HM95] M. Hitz and B. Montazeri. Measuring coupling and cohesion in object-oriented systems. In *Proceedings of International Symposium on Applied Corporate Computing*, Monterrey, Mexico, October 1995.

[HMZ09] Mark Harman, S. Afshin Mansouri, and Yuanyuan Zhang. Search based software enginerring: A comprehensive analysis and review of trend techniques and applications. Technical report, Department of Computer Science, King's College, London, 2009.

[HS96] B. Henderson-Sellers. *Object-Oriented Metrics Measures of Complexity*. Prentice-Hall, 1996.

[HT07] Mark Harman and Laurence Tratt. Pareto optimal search based refactoring at the design level. In *Conference on Genetic and Evolutionary Computation*, pages 1106–1113, 2007.

[IPl13] iplasma, 2013. http://loose.upt.ro/reengineering/research/iplasma.

[ISIE12] Safwat M. Ibrahim, Sameh A. Salem, Manal A. Ismail, and Mohamed Eladawy. Identification of nominated classes for software refactoring using object-oriented cohesion metrics. *International Journal of Computer Science Issues*, 9(2):68–76, 2012.

[ISO13] Iso8583, 2013. http://sourceforge.net/projects/j8583/.

[JDe13] Jdeodorant, 2013. http://www.jdeodorant.com/.

[JLZ11] Yuan Jiang, Ming Li, and Zhi-Hua Zhou. Software defect detection with rocus. *Journal of Computer Science and Technology*, 26(2):328–342, 2011.

[Kan03] Ronald Kirk Kandt. A software defect detection methodology, 2003.

[KMMiM08] Yasutaka Kamei, Akito Monden, Shuji Morisaki, and Ken ichi Matsumoto. A hybrid faulty module prediction using association rule mining and logistic regression analysis. In *Proceedings of the International Symposium on Empirical Software Engineering and Measurements(ESEM)*, pages 279–281, 2008.

[KSBW11] Marouane Kessentini, Houari Sahraoui, Mounir Boukadoum, and Manuel Wimmer. Search-based design defects detection by example. In *Proceedings of the 14th International Conference on Fundamental Approaches to Software Engineering*, pages 401–415, 2011.

[KVGS09] Foutse Khomh, Stéphane Vaucher, Yann-Gaël Guéhéneuc, and Houari Sahraoui. A bayesian approach for the detection of code and design smells. In *Proceedings of the 9th International Conference on Quality Software*, pages 305–314, 2009.

[Lar04] Craig Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*. Addison Wesley Professional, third edition, 2004.

[LH93] Wei Li and Sallie Henry. Object oriented metrics which predict maintainability. *Journal of Systems and Software*, 23(2):111–122, 1993.

[LHM98] Bing Liu, Wynne Hsu, and Yiming Ma. Integrating classification and association rule mining. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 80–86, 1998.

[LLWW95] Y. S. Lee, B. S. Liang, S. F. Wu, and F. J. Wang. Measuring the coupling and cohesion of an object-oriented program based on information flow. In *Proceedings of International Conference on Software Quality*, Maribor, Slovenia, 1995.

[LMW01] Bing Liu, Yiming Ma, and Ching-Kian Wong. *Data Mining for Scientific and Engineering Applications*, chapter Classification Using Association Rules: Weaknesses and Enhancements. Kluwer Academic, 2001.

[LZWZ12] Ming Li, Hongyu Zhang, Rongxin Wu, and Zhi-Hua Zhou. Sample-based software defect prediction with active and semi-supervised learning. *Automated Software Engineering*, 19(2):201–230, 2012.

[Mai09] Sayyed Garba Maisikeli. *Aspect Mining Using Self-Organizing Maps With Method Level Dynamic Software Metrics as Input Vectors*. PhD thesis, Graduate School of Computer and Information Sciences Nova Southeastern University, 2009.

[Mar02] Radu Marinescu. *Measurement and Quality in Object-Oriented Design*. PhD thesis, Politechnica University Timisoara, Faculty of Automatics and Computer Science, 2002.

[Mar10] Zsuzsanna Marian. Solving the subset sum problem with dna computation. In *Proceedings of the National Symposium ZAC*, pages 25–29, 2010.

[Mar12a] Zsuzsanna Marian. Aggregated metrics guided software refactoring. In *Proceedings of the 8th IEEE International Conference on Intelligent Computer Communication and Processing*, pages 259–266, 2012.

[Mar12b] Zsuzsanna Marian. Software metrics based refactoring: a case study. In *Proceedings of the National Symposium ZAC*, pages 59–64, 2012.

[Mar12c] Zsuzsanna Marian. A study on hierarchical clustering based software restructuring. *Studia Universitatis "Babes-Bolyai" Informatica*, LVII(2):20–31, 2012.

[Mar13a] Zsuzsanna Marian. On the software metrics influence in relational association rule-based software defect prediction. *Studia Universitatis "Babes-Bolyai" Informatica*, LVIII(4):35–48, 2013.

[Mar13b] Zsuzsanna Marian. A study on association rule mining based software defect detection. *Studia Universitatis "Babes-Bolyai" Informatica*, LVIII(1):42–57, 2013.

[Mar14a] Zsuzsanna Marian. Faos - a framework for analyzing object-oriented software systems. *Studia Universitatis "Babes-Bolyai" Informatica*, 2014. Under review.

[Mar14b] Zsuzsanna Marian. On evaluating the structure of software packages. *Studia Universitatis "Babes-Bolyai" Informatica*, LIX(1):46–58, 2014.

[MC11] Iman Hemati Moghadam and Mel Ó. Cinnéide. Code-imp: A tool for automated search-based refactoring. In *Proceeding of the 4th Workshop on Refactoring Tools*, pages 41–44, Honolulu, USA, 2011.

[MCB11] Zsuzsanna Marian, Cosmin Coman, and Attila Bartha. Learning to play the guessing game. *Studia Universitatis "Babes-Bolyai" Informatica*, LVI(2):119–124, 2011.

[MCC12] Zsuzsanna Marian, Gabriela Czibula, and Istvan-Gergely Czibula. Using software metrics for automatic software design improvement. *SIC Journal, Studies in Informatics and Control*, 21(3):249–258, 2012.

[MCC14] Zsuzsanna Marian, Gabriela Czibula, and Istvan Gergely Czibula. Software packages refactoring using a hierarchical clustering-based approach. *Fundamenta Informaticae*, 2014. Under review.

[MGF07] Tim Menzies, Jeremy Greenwald, and Art Frank. Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering*, 33(1):2–13, 2007.

[MGL06] Naouel Moha, Yann-Gaël Guéhéneuc, and Pierre Leduc. Automatic generation of detection algorithms for design defects. In *Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering*, pages 297–300, 2006.

[MHH03] Kiarash Mahdavi, Mark Harman, and Robert Mark Hierons. A multiple hill climbing approach to software module clustering. In *Proceedings of the International Conference on Software Maintenance*, pages 315–324, 2003.

[Mit06] Tom M. Mitchell. The discipline of machine learning. Working paper, 2006.

[MMCG99] S. Mancoridis, B. S. Mitchell, Y. Chen, and E. R. Gansner. Bunch: A clustering tool for the recovery and maintenance of software system structures. In *In Proceedings of the IEEE International Conference on Software Maintenance*, pages 50–59, 1999.

[MML01] Andrian Marcus, Jonathan I. Maletic, and K. I. Lin. Ordinal association rules for error identification in data sets. In *Proceedings of the 10th International Conference on Information and Knowledge Management*, pages 589–591, 2001.

[Moh06] Naouel Moha. Detection and correction of design defects in object-oriented architectures. In *Doctoral Symposium, 20th edition of the European Conference on Object-Oriented Programming*, 2006.

[MS10] Zsuzsanna Marian and Christian Săcărea. Using contextual topology to discover similarities in modern music. In *Proceedings of the IEEE International Conference on Automation Quality and Testing, Robotics*, volume 3, pages 1–6, 2010.

[Mun05] Matthew James Munro. Product metrics for automatic identification of "bad smell" design problems in java source code. In *Proceedings of the 11th IEEE International Software Metrics Symposium*, 2005.

[Nas] Nasa software defect datasets.

[OC08] Mark O'Keeffe and Mel Ó. Cinnéide. Search-based refactoring for software maintenance. *The Journal of Systems and Software*, 81:502–516, 2008.

[PJL13] Wei-Feng Pan, Bo Jiang, and Bing Li. Refactoring software packages via community detection in complex software networks. *International Journal of Automation and Computing*, 10(2):157–166, 2013.

[Pro13] Profiler4j, 2013. http://sourceforge.net/projects/profiler4j/.

[rlf] Reinforcement learning framework. http://www.cs.ubbcluj.ro/∼gabis/rl.

[RR11] Akepogu Ananda Rao and Kalam Narendar Reddy. Identifying clusters of concepts in a low cohesive class for extract class refactoring using metrics supplemented agglomerative clustering technique. *International Journal of Computer Science Issues*, 8(2):185–194, 2011.

[RRRAR12] D. Rodríguez, R. Ruiz, J. C. Riquelme, and J. S. Aguilar-Ruiz. Searching for rules to detect defective modules: A subgroup discovery approach. *Information Sciences*, 191:14–30, May 2012.

[SCC06] Gabriela Serban, Alina Câmpan, and Istvan Gergely Czibula. A programming interface for finding relational association rules. *International Journal of Computers, Communications & Control*, I(S.):439–444, June 2006.

[SKR08] Santonu Sarkar, Avinash C. Kak, and Girish Maskeri Rama. Metrics for measuring the quality of modularization of large-scale object-oriented software. *IEEE Transactions on Software Engineering*, 34(5):700–720, 2008.

[SMV09] Adrian Sterca, Zsuzsanna Marian, and Alexandru Vancea. Distortion-based media-friendly congestion control. In *Proceedings of the International Conference on Knowledge Engineering, Principles and Techniques*, pages 265–267, 2009.

[Spe04] C. Spearman. The proof and measurement of association between two things. *Amer. J. Psychol.**15***, pages 72–101, 1904.

[SS07] Konstantinos Stroggylos and Diomidis Spinellis. Refactoring - does it improve software quality? In *Proceedings of the 5th International Workshop on Software Quality*, 2007.

[SSB06] Olaf Seng, Johannes Stammel, and David Burkhart. Search-based determination of refactorings for improving the class structure of object-oriented systems. In *Conference on Genetic and Evolutionary Computation*, pages 1909 – 1916, 2006.

[SSL01] Frank Simon, Frank Steinbrückner, and Claus Lewerentz. Metrics based refactoring. In *Proceedings of the 5th European Conference on Software Maintenance and Reengineering*, pages 30–38, 2001.

[ST98] Gregor Snelting and Frank Tip. Reengineering class hierarchies using concept analysis. In *Proceedings of the 6th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 99–110, 1998.

[ŢIM09] Radu Ţurcaş, Oana Iova, and Zsuzsanna Marian. The autonomous robotic tank (art): an innovative lego mindstorm nxt battle vehicle. In *Proceedings of the International Conference on Knowledge Engineering, Principles and Techniques*, pages 95–98, 2009.

[TK03] Ladan Tahvildari and Kostas Kontogiannis. A metric-based approach to enhance the design quality through meta-pattern transformations. In *Proceedings of the European Conference on Software Maintenance and Reengineering*, pages 183–192, 2003.

[TLM11] Doina Tatar, Mihaiela Lupea, and Zsuzsanna Marian. Text summarization by formal concept analysis approach. *Studia Universitatis "Babes-Bolyai" Informatica*, LVI(2):7–12, 2011.

[Tuf11] Stphane Tuffry. *Data Mining and Statistics for Decision Making*. John Wiley and Sons, 2011.

[Win13] Winrun4j, 2013. http://sourceforge.net/projects/winrun4j/.

[ZT05] Du Zhang and Jeffrey J. P. Tsai. *Machine Learning Applications in Software Engineering*. Series on Software Engineering and Knowledge Engineering. World Scientific Publishing, 2005.