The Role of Software Testing and Machine Learning in Automated Program Repair

Summary of the PhD Thesis Points

[1] Textual Similarity Techniques in Code Level Traceability

[2] Machine Learning in Automated Program Repair

[3] Automated Assessment of Automatically Generated Patches

Viktor Csuvik

Supervisor:

Dr. László Vidács

Doctoral School of Computer Science
Department of Software Engineering
Faculty of Science and Informatics
University of Szeged

Szeged
2024

Introduction

To discover the link between human thinking and mechanical computers was already a concern of philosophers and mathematicians in the 17th and 18th centuries. Questions related to **Artificial Intelligence (AI)** only emerged later, describing a hardware or software that allows a machine to mimic human intelligence. **Machine Learning (ML)** develops and applies algorithms that can learn from experience and improve over time without being explicitly programmed. Thus, it is a technique to imitate human intelligence. The PhD dissertation presents some ML applications in the hope of helping the software development process to produce better quality systems. In a narrower context, the field of **Automated Program Repair (APR)** is explored, with a particular focus on the role of tests.

Software testing constitutes a major aspect in the assurance of the quality of a software. Besides simply indicating faults in it, tests are also essential for other areas in software engineering, like APR. AP aims to automatically fix defects in computer programs. It has the potential to significantly improve software reliability and reduce the cost and time associated with manual debugging and repair [24]. The skeleton of most APR approaches are similar [11], and they mainly consist of the steps depicted on Figure 1. The thesis is divided into three main chapters, each aligning with one of the three main points.

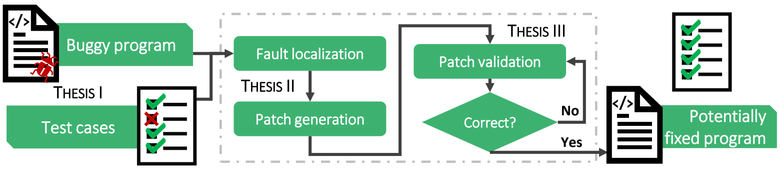


Figure 1: *General APR approach.*

The first theses group explores textual methods for identifying classes suitable for unit testing. It discusses the importance and recent advancements in test-to-code traceability techniques, emphasizing the use of textual methods.

The second part discusses using **Deep Learning (DL)** techniques to improve bug fixing in **Fault Localization (FL)** and APR. The chapter introduces methods for stable AI training, datasets for learning-based APR training, and tools for generating patches automatically.

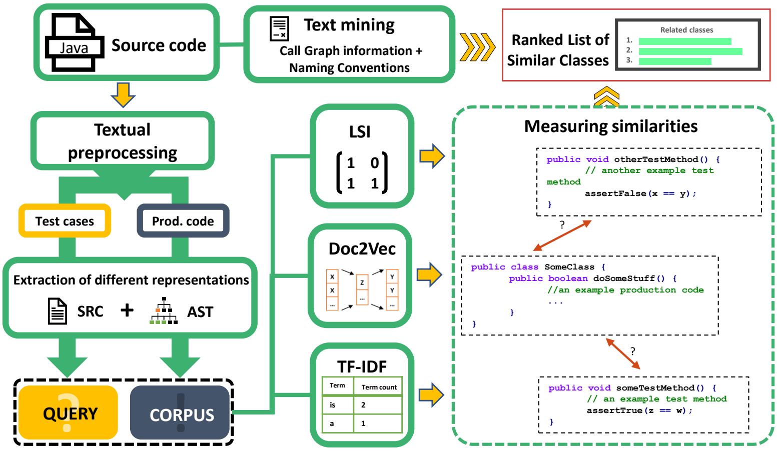
The last theses group explores solutions for the **Patch Correctness Check (PCC)** problem, which involves verifying the correctness of automatically generated fixes. It introduces a similarity-based approach and a classification method using cutting-edge techniques.

This summary organized as follows. In the next three chapters each theses is briefly presented. The contribution of the author is described at the end of each section. Next, the dissertation is concluded together with the mapping between the thesis points and publications. A Hungarian summary can be found in the last section.

1 Textual Similarity Techniques in Code Level Traceability

Test-to-code traceability means finding the links between test cases and production code. More precisely for a test case we want to find certain parts of the code which it was meant to test [26]. In this theses group textual similarity techniques are introduced to recover traceability links between code artifacts. Figure 2 provides an illustration of the proposed approach. A software system written in Java programming language is considered, which contains both test classes and production code and the aim is to recover the relationship between them. ML models are trained on the source code of the observed systems. Similarity is measured between tests and code classes, from which a ranked list is constructed. This list is then fine-tuned using previously extracted call graph information. Additionally to the three ML models and five representations, the first theses group presents experiments employing the $ensemble_N$ approach, where the optimal N value is sought using NC-based and manual traceability links. The basic idea is that test and code classes are *similar* in some sense and that additional structural information will filter out the errors of the previous steps.

Five code representations are introduced: IDENT, SRC, AST, LEAF and SIMPLE. These form the input of an ML algorithm that computes the similarity between distinct



```
public void otherTestMethod() {
        // another example test
method
        assertFalse(x == y);
}
```

```
public class SomeClass {
        public boolean doSomeStuff() {
                //an example production code
                ...
        }
}
```

```
public void someTestMethod() {
        // an example test method
        assertTrue(z == w);
}
```

Figure 2: *A high-level overview of Theses I.*

items. Abstract Syntax Tree were utilized to form a sequence of tokens from the structured source code. The five representations under evaluation were constructed according to the most widely used representations in other research experiments [25].

The evaluation encompassed six traceability link extraction methods assessed with five code representations. Among these, IDENT, leveraging ASTs, emerged as the most effective in the majority of cases during the naming convention-based evaluation. We refined Doc2Vec's ranked similarity list with the recommendation of the other approaches. With this experiment we have successfully improved the performance of it for every project, therefore introducing a successful mixed approach. However, the SRC representation exhibited greater precision when compared to a limited amount of manual data. The inclusion of call information retrieved via regular expressions significantly bolstered results when employed as a filtering technique for Doc2Vec. Although LSI and TF-IDF also demonstrated promise for the same purpose, the fusion of Doc2Vec and call information yielded superior results. While well-defined and consistently adhered to naming conventions have the potential to yield highly precise traceability links, their application remains constrained. Automatic recovery of naming conventions stands to benefit substantially from the integration of other text-based techniques, fostering a versatile semantic approach that can be effectively employed alongside other mainstream methods.

1.1 The Author's Contributions

In the first thesis group, the author implemented the Doc2vec and TF-IDF methods for recovering traceability links. Additionally, he implemented the text-based recovery technique that retrieved call graph information from static code. The definition of the used source code representations and metric visualizations was also part of the author's work. He also took part in the evaluation and explanation of various other results, as well as in the planning and writing of all the published papers. Detailed discussion can be found in Chapter 3 of the dissertation.

2 Machine Learning in Automated Program Repair

In recent years, there has been growing interest in using ML techniques for APR [21]. These techniques have shown promise in generating high-quality repair patches for a variety of programming languages and domains [22]. However, APR is a challenging task due to the complexity and variability of software systems with many open challenges. The current theses group applies ML on several subtasks in the APR domain, Figure 3 depicts a comprehensive overview of it.

The first section dives into more details about Fault Localization. The main focus in this chapter is on the relationship between traditional and DL algorithms. By conducting a large-scale training, the stability of DL-based FL methods are investigated. As can be seen on the figure, the FL part of the process takes the coverage information from test execution and outputs a list of most suspicious statements that needs to be repaired. The classical Generate&Validate patch generation approaches then try to modify these faulty statements to fix the whole program. On the other hand, DL approaches usually refrain from FL (assuming perfect knowledge) and only focus on the generation of repair candidates.

Next, a data collection approach is described, since modern data-intensive APR applications need a lot of training data. As JavaScript lacks such curated dataset, the evaluation of the proposed APR methods is difficult. The section introduces the *FixJS* dataset, describes its properties and structure. The proposed dataset is available on GitHub and have a DOI to make it easily citable. It contains roughly two million bug-fixing commits from GitHub. From these commits, the modified functions were extracted (the state before and after the bug fix happened). These functions are then tokenized and abstracted, resulting in three different source code representations. FixJS can be used to train and evaluate a deep learning model that predicts correct fixes without any further processing steps.

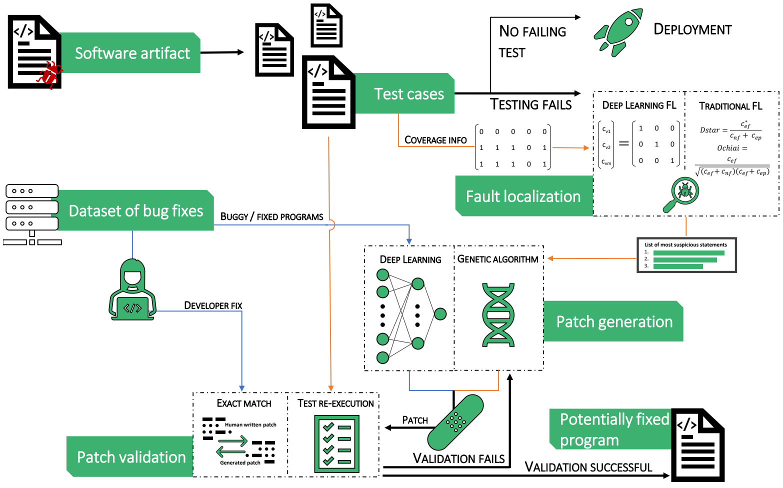


Figure 3: *A comprehensive overview of Theses II.*

The last chapter focuses on the generation of patches both with traditional and data-driven approaches. An adaptation of the seminal APR tool GenProg is introduced. Also, to bypass the cumbersome process of designing, training, and evaluating a new model, an approach is proposed that fixes buggy programs automatically using ChatGPT, simply relying on this Large Language Model. On Figure 3 we can see that there is a fundamental difference of the two: while learning-based APR approaches are evaluated against the developer fix, traditional tools evaluate the generated patch by re-executing the test cases. The former one is obviously more strict, but has three main flaws: (1) it needs a sufficient database to train and evaluate on, (2) does not consider semantically identical patches and (3) overlook the fact that a developer patch can be flawed. On the other hand, classifying a patch by test execution is both more time-consuming and more susceptible to overfitting - where only test cases are successfully executed, but the real bug is not fixed. This phenomena is detailed in the next theses group.

The findings of the theses group are manyfold. First, in Fault Localization, it was found that the output of the same model varies greatly due to the effect of random factors during training phase. In the dataset creation process the characteristics of FixJS has been introduced and its relevance in the field of APR have been described. Patch generation is described in detail in the thesis group. First, GenProgJS is described, which is based on a genetic algorithm and targets buggy programs written in JavaScript. According to the first experiments GenProgJS found plausible repairs for 31 bugs in 6 Node.js projects, which is a comparable result to the related work done on other languages. The capabilities of ChatGPT were also investigated on how it performs when tasked with fixing buggy code. 200 buggy codes were sampled from APR datasets, consisting of 100 Java and 100 JavaScript samples. The best prompt for Java generated correct answers in 19% of cases, while for JavaScript, the same prompt yielded a performance of only 4%. In total, 44 distinct bugs were repaired in Java and 24 in JavaScript out of the overall 200 samples and 1000 repair trials.

2.1 The Author's Contributions

In the second thesis group, the author coordinated the experimentations on diverse network architectures on DL-based FL and implemented the bucketing approach. He also adapted churn metric and took part in the design and writing of the published paper. The FixJS benchmark creation and ChatGPT experiments were entirely the work of the author. In the GenProgJS tool, the author implemented the base genetic algorithm, and the interface for test case evaluation and operator calls. He also executed the experiments, coordinated the analysis and took a big part in the explanation of results. Detailed discussion can be found in Chapter 4 of the dissertation.

3 Automated Assessment of Automatically Generated Patches

A notable obstacle encountered in test-suite-based repair is the potential to create a patch that enables the entire test suite to pass, yet remains incorrect. This phenomenon is commonly referred to as the overfitting patch problem [23] and the goal of Patch Correctness Check is to determine the actual correctness of a patch, without additional manual effort. The generation of overfitting patches leads to the generation of program repair patches with limited utility, thereby substantially affecting the practical applicability of program repair. It also makes developers less confident in APR tools, thus reducing their widespread use.

In the third theses group two approaches are executed and evaluated, this can be observed on Figure 4. First, it is investigated whether similarity is an appropriate approach to tackle with the PCC problem. To do so, similarity between generated plausible patches and the original code is measured. The intuition behind similarity-based approach is that more similar patches deem to be more simple and thus not overfitting ”alien codes” (left side of Figure 4). The source code similarities are measured using document/sentence embeddings, specifically with two state of the art techniques borrowed from the NLP domain: Doc2vec and Bert.

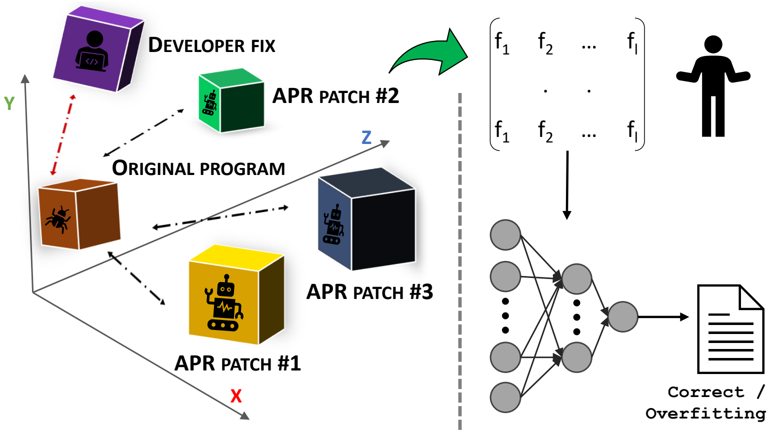


Figure 4: *The general outline of Theses III.*

Next, PCC is considered as a classification problem. Code features are borrowed from previous works achieving state-of-the-art results: hand-crafted and static features together with embedding vectors and similarity metrics. On this set, a feature selection process is performed, which resulted in a 43-dimensional feature vector (right side of Figure 4). The selected features served as the basis for model selection, in which ML models are trained, evaluated and optimized.

Based on the results, similarity-based approach may be useful when a high number of plausible patches are present, but we found that plain source code embeddings fail to capture nuanced code semantics, thus a more sophisticated technique is needed to correctly validate patches. It is expected that a more complex language understanding model may be advantageous in deciding whether a patch is correct or not. On the other hand, with classifiers one can filter out overfitting patches with a high degree of confidence. Initially, the feature set has been reduced, indicating the limited informational contribution of most original features. Subsequently, we conducted training and evaluation of nine ML models to discern the optimal performer. Our findings suggest better performance of the models on average when utilizing the selected feature set in comparison to the entire feature set or other subsets. The best performing models, achieved average F1 scores of 0.8-0.77. Employing a more complex neural architecture that integrates learned embeddings with other features enabled us to mitigate the variability in training, reducing the absolute fluctuation in the F1 score from 30% to 16%.

3.1 The Author's Contributions

In the third thesis group, the author laid the groundwork for the similarity-based PCC technique and implemented the base algorithm. He took part in the manual annotation of the generated patches. The author coordinated the implementation of the ML-based classifiers and conducted benchmark creation / gathering of all required metrics. He also planned the experiment guidelines and took a big role in the evaluation and explanation of the results and their implications. Detailed discussion can be found in Chapter 5 of the dissertation.

Summary

The first part of the dissertation, provides insight into a traceability problem: to connect test cases with code classes solely base on textual methods. The thesis examines several prevalent methods, such as naming conventions and LSI, and introduces the use of Doc2Vec. It experimented with different source code representations and found that IDENT, a simple representation, yielded better results for traceability. Doc2Vec-based similarity outperformed other methods. Combining Doc2Vec with recommendations from other approaches further improved performance, establishing a successful mixed approach for matching tests with production code. It is evident that a combination of methods yields optimal results in this field, and textual analysis is expected to remain important in future work.

The second theses group started with a discussion on Fault Localization using Deep Learning techniques. The findings of the study can be generalized to the whole Software Engineering and Artificial Intelligence domain: scientific work using Machine Learning should concentrate more on reproducibility and stability aside from publishing great results. Next, a data mining approach has been presented, and the FixJS dataset. It can be used for APR research: just as in the subsequent parts where patches are generated both by DL and traditional approaches. Results show that in practical application Generate&Validate APR approaches still play a promi-

nent role, with DL-based tools outperforming them in some cases. An important and difficult task for future research will be to combine the strengths of the two areas and avoid the weaknesses. The dissertation presents a thorough examination of the effectiveness of genetic operators and showcased instances of potential patches discovered by both of the algorithms.

The last part tackled with the PCC problem - that is, given an automatically generated patch, one should decide whether it is a real fix to the bug, or an overfit to the test oracle. The chapter elaborates on how our work contributed to the field, by defining similarity-based patch filtering and evaluating classification on state-of-the-art features sets. In the realm of APR, patch validation remains relatively uncharted yet promising. Filtering out erroneous patches is crucial for enhancing confidence in automatic tools. This chapter explores experiments employing both similarity-based patch filtering and feature-based classification methods. Results show that while the current solutions still have some flaws, by selecting proper features and classifiers, one can filter out overfitting patches with a high degree of confidence. The research findings emphasize two key points. Firstly, constructing effective PCC classifiers demands meticulous consideration of feature selection and model construction. While handcrafted features remain essential, embeddings can also offer valuable insights. Secondly, ML approaches should prioritize model stability, as it significantly impacts the reliability and importance of the results obtained - a similar observation made in previous section.

The methodologies, experiments, and findings presented in the thesis have been extensively discussed in several of the author's prior works, nine of which are referenced here. Their relevance to the specific thesis points of the thesis is presented in Table 1.

Table 1: *Correspondence between the thesis points and publications.*

	Publications								
No.	[16]	[17]	[20]	[19]	[12]	[18]	[15]	[13]	[14]
I.	●	●	●						
II.				●	●	●			
III.							●	●	●

Acknowledgement

The research presented was supported in part by the ÚNKP-SZTE New National Excellence Program and the national project TKP2021-NVA-09, financed under the TKP2021-NVA funding scheme of the Ministry for Culture and Innovation from the source of the National Research, Development and Innovation Fund, and by the European Union project RRF-2.3.1-21-2022-00004 within the framework of the Artificial Intelligence National Laboratory.

4 Összefoglalás

A doktori disszertáció három fő témát tárgyal, amelyek mindegyike valamelyest kapcsolódik egymáshoz. Egy szoftvertermék minősége nagyban függ a fejlesztői szokásoktól és gyakorlatoktól. Az értekezésből látható a tesztesetek fontossága - kezdve a megnevezésüktől a hibalokalizációban és az automatikus programjavításban betöltött szerepükben.

A disszertáció első része betekintést nyújt egy nyomonkövethetőségi problémába: a tesztesetek összekapcsolásába kódosztályokkal, kizárólag szöveges módszerek felhasználásával. A dolgozat számos jól ismert módszert vizsgál, mint például a névkonvenciókat, az LSI-t, és bemutat egy új alternatívát is: a Doc2Vec-et. Különböző forráskód reprezentációk kerültek bemutatásra, és azt láthattuk, hogy az egyszerű IDENT reprezentáció jobb eredményeket adott a nyomonkövethetőség tekintetében a többi szöveges reprezentációtól. A Doc2Vec-alapú hasonlóság felülmúlta a többi módszert. A Doc2Vec más megközelítésekből származó hasonlósági listákkal való kombinálása tovább javította a teljesítményt. Láthattuk, hogy a szöveges technikák rugalmas megközelítést biztosítanak, valamint ezek kombinációja javítja a teljesítményt, így a szövegelemzés várhatóan továbbra is fontos marad a jövőbeni munkákban.

A következő tézispont a mélytanulás alapú hibalokalizáció tárgyalásával kezdődik. A fejezet megállapításai általánosíthatók az egész szoftverfejlesztés és mesterseges intelligencia területre: a gépi tanulást használó tudományos munkáknak a nagyszerű eredmények publikálása mellett jobban kellene koncentrálnia a reprodukálhatóságra és a stabilitásra. Ezután egy adat kinyerési megközelítés került bemutatásra, és a FixJS adathalmaz. A következő fejezetben patchek mind mélytanulással, mind hagyományos megközelítésekkel generálódnak. Ezek az eredmények azt mutatják, hogy a gyakorlati alkalmazásban a hagyományos genetikus megközelítések még mindig kiemelkedő szerepet játszanak, a mélytanuló eszközök csak egyes esetekben múlják felül azokat. A jövőbeli kutatások fontos és nehéz feladata lesz e két terület erősségeinek ötvözése a gyengeségek elkerülésével. A disszertáció alaposan megvizsgálja a genetikus operátorok hatékonyságát, és bemutatja a mindkét algoritmus által generált potenciálisan javító patch-eket.

A harmadik rész az automatikus patch értékelés problémával foglalkozik - vagyis egy automatikusan generált javítás esetén el kell dönteni, hogy az a hiba valódi javítását jelenti-e, vagy csupán túlillesztés történt a tesztekre. A tézispontban bemutatásra kerül munkánk hogyan járul hozzá a területhez - a hasonlóság-alapú patch szűrés definiálásával és az osztályozók kiértékelésével a legkorszerűbb jellemzőkészleteken. A hibás javítások kiszűrése kulcsfontosságú az ilyen automatikus eszközökbe vetett bizalom növeléséhez. Az eredmények alapján, a jelenlegi megközelítések még korán sem tökéletesek, a megfelelő jellemzők és osztályozók kiválasztásával nagy megbízhatósággal ki lehet szűrni a túlillesztett javításokat.

References

[11] Fatmah Yousef Assiri and James M. Bieman. Fault localization for automated program repair: effectiveness, performance, repair correctness. *Software Quality Journal*, 25(1):171–199, mar 2017.

[12] Viktor Csuvik., Tibor Gyimóthy., and László Vidács. Can chatgpt fix my code? In *Proceedings of the 18th International Conference on Software Technologies - ICSOFT*, pages 478–485. INSTICC, SciTePress, 2023.

[13] Viktor Csuvik, Dániel Horváth, Márk Lajkó, and László Vidács. Exploring plausible patches using source code embeddings in javascript. *2021 IEEE/ACM International Workshop on Automated Program Repair (APR)*, pages 11–18, 2021.

[14] Viktor Csuvik, Daniel Horvath, and Laszlo Vidacs. Feature extraction, learning and selection in support of patch correctness assessment. In *Proceedings of the 19th International Conference on Software Technologies - ICSOFT*, 2024.

[15] Viktor Csuvik, Deniel Horvath, Ferenc Horvath, and Laszlo Vidacs. Utilizing Source Code Embeddings to Identify Correct Patches. In *2020 IEEE 2nd International Workshop on Intelligent Bug Fixing (IBF)*, pages 18–25. IEEE, 2020.

[16] Viktor Csuvik, András Kicsi, and László Vidács. Evaluation of Textual Similarity Techniques in Code Level Traceability. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 11622 LNCS, pages 529–543. Springer Verlag, 2019.

[17] Viktor Csuvik, András Kicsi, and László Vidács. Source code level word embeddings in aiding semantic test-to-code traceability. In *10th International Workshop at the 41st International Conference on Software Engineering (ICSE) – SST 2019*. IEEE, 2019.

[18] Viktor Csuvik, Aszmann Roland, Beszédes Árpád, Horváth Ferenc, and Gyimóthy Tibor. On the stability and applicability of deep learning in fault localization. In *2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2024.

[19] Viktor Csuvik and László Vidács. Fixjs: A dataset of bug-fixing javascript commits. In *2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR)*, pages 712–716, 2022.

[20] András Kicsi, Viktor Csuvik, and László Vidács. Large scale evaluation of natural language processing based test-to-code traceability approaches. *IEEE Access*, 9:79089–79104, 2021.

[21] Fan Long and Martin Rinard. Automatic patch generation by learning correct code. *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages - POPL 2016*, pages 298–312, 2016.

[22] Martin Monperrus. The Living Review on Automated Program Repair. Technical report, dec 2020.

[23] Shangwen Wang, Ming Wen, Bo Lin, Hongjun Wu, Yihao Qin, Deqing Zou, Xiaoguang Mao, and Hai Jin. Automated patch correctness assessment: how far are we? In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, ASE '20, page 968–980, New York, NY, USA, 2021. Association for Computing Machinery.

[24] Westley Weimer, ThanhVu Nguyen, Claire Le Goues, and Stephanie Forrest. Automatically finding patches using genetic programming. In *Proceedings of the 31st International Conference on Software Engineering*, ICSE '09, page 364–374, USA, 2009. IEEE Computer Society.

[25] Martin White, Michele Tufano, Christopher Vendome, and Denys Poshyvanyk. Deep learning code fragments for code clone detection. *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering - ASE 2016*, pages 87–98, 2016.

[26] Greg Wilson, D. A. Aruliah, C. Titus Brown, Neil P. Chue Hong, Matt Davis, Richard T. Guy, Steven H.D. Haddock, Kathryn D. Huff, Ian M. Mitchell, Mark D. Plumbley, Ben Waugh, Ethan P. White, and Paul Wilson. Best Practices for Scientific Computing. *PLoS Biology*, 12(1):e1001745, jan 2014.

Nyilatkozat

Csuvik Viktor *"The Role of Software Testing and Machine Learning in Automated Program Repair"* című PhD disszertációjában a következő eredményekben fejezetenként Csuvik Viktor hozzájárulása volt a meghatározó:

1. Szöveges Hasonlósági Technikák a Kódszintű Nyomonkövethetőségben

(Textual Similarity Techniques in Code Level Traceability)

[1] Doc2Vec és TF-IDF szemantikai hasonlósági mérések implementációja a tesztek és kódosztályok, valamint kód-metódusok között. SRC, IDENT és AST kód reprezentációkat előállító Java csomag tervezése, implementációja és futtatása, melyeken a modellek futottak. Modell eredmények kiértékelésének futtatása, kinyert metrikák vizualizációja és táblázatba foglalása.

- Fejezetek: **1.3.1**, **1.3.2**, **1.4** és **1.6**
- **Viktor Csuvik**, András Kicsi, and László Vidács. Evaluation of Textual Similarity Techniques in Code Level Traceability. In *Lecture Notes in Computer Science (including subseries Lecture*

Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) - LNCS, Springer, 529–543, 2019.

- **Viktor Csuvik**, András Kicsi, and László Vidács. Source code level word embeddings in aiding semantic test-to-code traceability. In *10th International Workshop at the 41st International Conference on Software Engineering (ICSE) – SST*, IEEE, 29-33, 2019.

[2] LEAF és SIMPLE forráskód reprezentációkat előállító Java csomag implementálása és futtatása. A Doc2Vec, TF-IDF és *Ensemble* módszerek ajánlórendszerként való felhasználása, kiértékelése és magyarázata. Futtatások kivitelezése, táblázatok adattal veló feltöltése, összefüggések magyarázata. Hívási gráf (CG) információt előállító forráskód implementációja, ezen információ kombinálása a szemantikai módszerekkel.

- Fejezetek: **1.3.4**, **1.3.5**, **1.3.6**, **1.6** és **1.7.1**
- András Kicsi, **Viktor Csuvik** and László Vidács. Large scale evaluation of natural language processing based test-to-code traceability approaches. *IEEE Access*, Volume(9), 79089-79104, 2021.

[3] *Ensemble* megközelítés megtervezése, implementá-

lása és kiértékelése. Hasonlósági listák nagyságának értékelése és kísérletek végrehajtása.

- Fejezetek: **1.3.4** és **1.6**
- **Viktor Csuvik**, András Kicsi, and László Vidács. Evaluation of Textual Similarity Techniques in Code Level Traceability. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) - LNCS*, Springer, 529–543, 2019.
- András Kicsi, **Viktor Csuvik** and László Vidács. Large scale evaluation of natural language processing based test-to-code traceability approaches. *IEEE Access*, Volume(9), 79089-79104, 2021.

2. Gépi Tanulás az Automatikus Programjavítás Területén

(Machine Learning in Automated Program Repair)

[1] Mélytanuló modell működésének magyarázata a hibalokalizációs eljárás során, neuronháló tanítását

javító kísérletek vezetése. Churn metrika adaptálása az FL eljárásra, eredmények értelmezése és magyarázata.

- Fejezetek: **2.3** és **2.4.1**
- **Viktor Csuvik**, Roland Aszmann, Árpád Beszédes, Ferenc Horváth, and Tibor Gyimóthy On the stability and applicability of deep learning in fault localization. In *2024 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, IEEE, 2024.

[2] FixJS adatbázis kinyerését végrehajtó szkript tervezése, implementációja és futtatása. Kinyert adatok elemzése, kézi ellenőrzése és csoportosítása.

- Fejezetek: **2.5**
- **Viktor Csuvik**, and László Vidács. Fixjs: A dataset of bug-fixing javascript commits. In *2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR)*, ACM, 712–716, 2022.

[3] ChatGPT felhasználása az automatikus programjavítás területén, promptok tervezése, adathalmaz gyűjtése és eredmények manuális kinyerése. Új

javítások kiértékelése, ezek magyarázata, értékelése és konklúziók levonása.

- Fejezetek: **2.6**
- **Viktor Csuvik**, Tibor Gyimóthy, and László Vidács. Can chatgpt fix my code?. In *Proceedings of the 18th International Conference on Software Technologies - ICSOFT*, SciTePress, 478-485 2023.

[4] Genetikus algoritmus és rendszer architektúra tervezése, implementációja és leírása. Operátorok és tesztek futtatásának integrációja a genetikus algoritmusba. A rendszer futtatása, eredmények gyűjtése és adminisztrációja. A kísérletek koordinációja, az eredmények és összefüggések magyarázata.

- Fejezetek: **2.7.1**, **2.7.2** és **2.7.3** (”Repaired bugs”, ”Repairing on script language” és ”Multiline repairs” részek)

3. Automatikusan Generált Javítások Spontán Értékelése

(Automated Assessment of Automatically Generated Patches)

[1] Hasonlóság-alapú kiértékelések megtervezése és a Doc2Vec modell implementációja. Adatgyűjtés, feldolgozás és rendszerezés. Kinyert hasonlóságok értelmezése, küszöb kísérletek. Az eredmények értelmezése, összefüggések magyarázata.

- Fejezetek: **3.3.1** és **3.5.1**
- **Viktor Csuvik**, Dániel Horváth, Ferenc Horváth, and László Vidács. Utilizing Source Code Embeddings to Identify Correct Patches. In *IEEE 2nd International Workshop on Intelligent Bug Fixing (IBF)*, IEEE, 18–25, 2020.

[2] GenprogJS által generált javítások gyűjtése és rendszerezése. Fejlesztői kiértékelések vitás eseteinek eldöntése. Metrikák mérése, ezek értelmezése és megjelenítése.

- Fejezetek: **3.3.1**, **3.4.1** és **3.6.1**
- **Viktor Csuvik**, Dániel Horváth, Márk Lajkó, and László Vidács. Exploring plausible patches using source code embeddings in javascript. In *IEEE/ACM International Workshop on Automated Program Repair (APR)*, IEEE, 11–18, 2021.

[3] Adagyűjtés és rendszerezés, melyen az osztályozók tanulni tudnak. Új kapcsolódó munkák kutatása.

Futtatások végrehajtása, melynek eredményei a beágyazások és metrikák. Kísérletek koordinálása és technikák ismertetése. A különböző technikák eredményeinek összehasonlítása, ezen eredmények értelmezése és magyarázata.

- Fejezetek: **3.3.2**, **3.3.3**, **3.5.2** és **3.6.2**
- **Viktor Csuvik**, Dániel Horváth, and László Vidács. Feature extraction, learning and selection in support of patch correctness assessment. In *Proceedings of the 19th International Conference on Software Technologies - ICSOFT*, SciTePress, 2024.

Ezek az eredmények Csuvik Viktor PhD disszertációján kívül más tudományos fokozat megszerzésére nem használhatók fel.

Szeged, 2024.06.07.

Csuvik Viktor

[illegible]

Csuvik Viktor
jelölt

Dr. Vidács László
témavezető

Az Informatika Doktori Iskola vezetője kijelenti, hogy jelen nyilatkozatot minden társszerzőhöz eljuttatta, és azzal szemben egyetlen társszerző sem emelt kifogást.

Szeged, 2024/06/19

[illegible]

SZEGEDI TUDOMÁNYEGYETEM INFORMATIKA DOKTORI ISKOLA SZEGED

Dr. Jelasity Márk
Informatikai Doktori
Iskola vezetője