Bridging Academia and Practice in Smart Contract Security

Francesco Salzano*
Lodovica Marchesi
Cosmo Kevin Antenucci
Simone Scalabrino
Roberto Tonelli
Rocco Oliveto
Remo Pareschi

7th Distributed Ledger Technologies Workshop

Finanziato dall'Unione europea
NextGenerationEU

Ministero dell'Università e della Ricerca

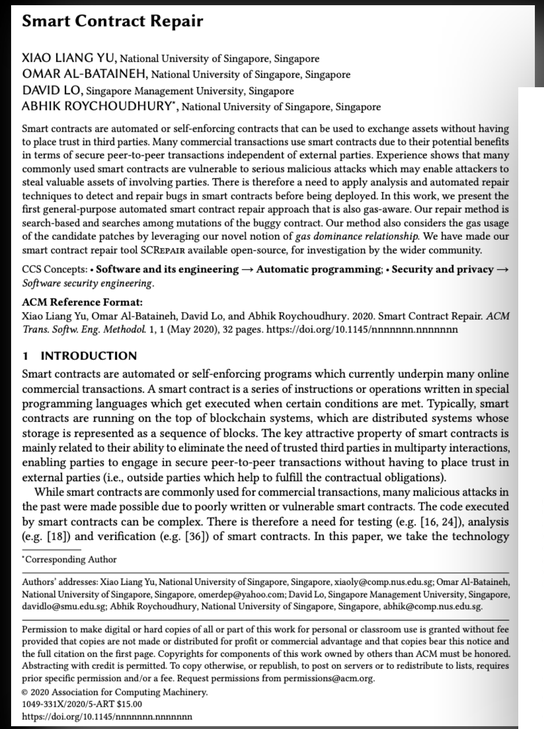
Italiadomani
PIANO NAZIONALE DI RIPRESA E RESILIENZA

Background: What is a Smart Contract?

Yu et al., 2020

Zou et al., 2019

Chen et al., 2020



Smart Contract Repair

XIAO LIANG YU, National University of Singapore, Singapore
OMAR AL-BATAINEH, National University of Singapore, Singapore
DAVID LO, Singapore Management University, Singapore
ABHIK ROYCHOUDHURY*, National University of Singapore, Singapore

Smart contracts are automated or self-enforcing contracts that can be used to exchange assets without having to place trust in third parties. Many commercial transactions use smart contracts due to their potential benefits in terms of secure peer-to-peer transactions independent of external parties. Experience shows that many commonly used smart contracts are vulnerable to serious malicious attacks which may enable attackers to steal valuable assets of involving parties. There is therefore a need to apply analysis and automated repair techniques to detect and repair bugs in smart contracts before being deployed. In this work, we present the first general-purpose automated smart contract repair approach that is also gas-aware. Our repair method is search-based and searches among mutations of the buggy contract. Our method also considers the gas usage of the candidate patches by leveraging our novel notion of *gas dominance relationship*. We have made our smart contract repair tool SCRepair available open-source, for investigation by the wider community.

CCS Concepts: • **Software and its engineering → Automatic programming**; • **Security and privacy →** *Software security engineering*.

ACM Reference Format:
Xiao Liang Yu, Omar Al-Bataineh, David Lo, and Abhik Roychoudhury. 2020. Smart Contract Repair. *ACM Trans. Softw. Eng. Methodol.* 1, 1 (May 2020), 32 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Smart contracts are automated or self-enforcing programs which currently underpin many online commercial transactions. A smart contract is a series of instructions or operations written in special programming languages which get executed when certain conditions are met. Typically, smart contracts are running on the top of blockchain systems, which are distributed systems whose storage is represented as a sequence of blocks. The key attractive property of smart contracts is mainly related to their ability to eliminate the need of trusted third parties in multiparty interactions, enabling parties to engage in secure peer-to-peer transactions without having to place trust in external parties (i.e., outside parties which help to fulfill the contractual obligations).

While smart contracts are commonly used for commercial transactions, many malicious attacks in the past were made possible due to poorly written or vulnerable smart contracts. The code executed by smart contracts can be complex. There is therefore a need for testing (e.g. [16, 24]), analysis (e.g. [18]) and verification (e.g. [36]) of smart contracts. In this paper, we take the technology

*Corresponding Author

Authors' addresses: Xiao Liang Yu, National University of Singapore, Singapore, xiaoly@comp.nus.edu.sg; Omar Al-Bataineh, National University of Singapore, Singapore, omerdep@yahoo.com; David Lo, Singapore Management University, Singapore, davidlo@smu.edu.sg; Abhik Roychoudhury, National University of Singapore, Singapore, abhik@comp.nus.edu.sg.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2020 Association for Computing Machinery.
1049-331X/2020/5-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Smart Contract Development: Challenges and Opportunities

Weiqin Zou, David Lo, Pavneet Singh Kochhar, Xuan-Bach Dinh Le, Xin Xia, Yang Feng, Zhenyu Chen, and Baowen Xu, *Member, IEEE*

Abstract—Smart contract, a term which was originally coined to refer to the automation of legal contracts in general, has recently seen much interest due to the advent of blockchain technology. Recently, the term is popularly used to refer to low-level code scripts running on a blockchain platform. Our study focuses exclusively on this subset of smart contracts. Such smart contracts have increasingly been gaining ground, finding numerous important applications (e.g., crowdfunding) in the real world. Despite the increasing popularity, smart contract development still remains somewhat a mystery to many developers largely due to its special design and applications. Are there any differences between smart contract development and traditional software development? What kind of challenges are faced by developers during smart contract development? Questions like these are important but have not been explored by researchers yet. In this paper, we performed an exploratory study to understand the current state and potential challenges developers are facing in developing smart contracts on blockchains, with a focus on Ethereum (the most popular public blockchain platform for smart contracts). Toward this end, we conducted this study in two phases. In the first phase, we conducted semi-structured interviews with 20 developers from GitHub and industry professionals who are working on smart contracts. In the second phase, we performed a survey on 232 practitioners to validate the findings from the interviews. Our interview and survey results revealed several major challenges developers are facing during smart contract development: (1) there is no effective way to guarantee the security of smart contract code; (2) existing tools for development are still very basic; (3) the programming languages and the virtual machines still have a number of limitations; (4) performance problems are hard to handle under resource constrained running environment; and (5) online resources (including advanced/updated documents and community support) are still limited. Our study suggests several directions that researchers and practitioners can work on to help improve developers' experience on developing high-quality smart contracts.

Index Terms—Smart contract, challenges, empirical study, blockchain

1 Introduction

Since the release of Bitcoin in 2009 [105], decentralized cryptocurrencies have gained considerable attention and adoption [2]. For instance, till February 2018, the numbers of coins and tokens hosted on the coinmarketcap[1] were 896 and 649, respectively. A cryptocurrency is administrated not by a central authority, but by automated consensus among networked users. The users in the cryptocurrency network run a consensus protocol to maintain and secure a public and append-only ledger of transactions, i.e., blockchain. In recent years, the potential of blockchain technology has been exploited beyond cryptocurrencies, among which a promising use of blockchain is *smart contract*.

The term "smart contract" was originally coined to refer to the automation of legal contracts in general [140]. The term was (and is still) used to refer to a legal contract which or at least parts of which is capable of being expressed and implemented in software [66]. The advent of blockchain technology has recently brought much interest on smart contracts. Today, the term is popularly used to refer to as code scripts that run synchronously on multiple nodes of a distributed ledger (e.g., a blockchain) [30]. In this paper, we mainly focus on the latter, more specific definition of smart contracts, i.e., low-level code scripts running on blockchains.

As a program running on a blockchain, a smart contract can be correctly executed by a network of mutually distrusting nodes without the need of an external trusted authority. The self-executing nature of smart contracts provides a tremendous opportunity for use in many fields that rely on data to drive transactions [139]. In the beginning of 2018, more than 10 percent of the jobs advertised on Guru[2] (one

1. http://coinmarketcap.com

- *W. Zou, Y. Feng, Z. Chen, and B. Xu are with the State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210008, China. E-mail: wqzou@smail.nju.edu.cn, charles.fy0708@gmail.com, {zychen, bxu}@nju.edu.cn.*
- *D. Lo is with the School of Information Systems, Singapore Management University, Singapore 188065. E-mail: davidlo@smu.edu.sg.*
- *P.S. Kochhar is with Microsoft, Mississauga, Canada. E-mail: kochharps.2012@phdis.smu.edu.sg.*
- *X.-B. Dinh Le is with the School of Computing and Information Systems, University of Melbourne, Parkville, VIC 3010, Australia. E-mail: bach.le@unimelb.edu.au.*
- *X. Xia is with the Faculty of Information Technology, Monash University, Clayton, VIC 3800, Australia. E-mail: xin.xia@monash.edu.*

Manuscript received 3 Jan. 2019; revised 31 Aug. 2019; accepted 10 Sept. 2019. Date of publication 24 Sept. 2019; date of current version 15 Oct. 2021. (Corresponding author: Zhenyu Chen.)
Recommended for acceptance by R. Mirandola.
Digital Object Identifier no. 10.1109/TSE.2019.2942301

2. http://www.guru.com/

Defining Smart Contract Defects on Ethereum

Jiachi Chen, Xin Xia, David Lo, John Grundy, Xiapu Luo, and Ting Chen

Smart contracts are programs running on a blockchain. They are immutable to change, and hence can not be patched for ence deployed. Thus it is critical to ensure they are bug-free and well-designed before deployment. A *Contract defect* is an error, fault in a smart contract that causes it to produce an incorrect or unexpected result, or to behave in unintended ways. The on of contract defects is a method to avoid potential bugs and improve the design of existing code. Since smart contracts contain ous distinctive features, such as the *gas system*. *decentralized*, it is important to find smart contract specified defects. To fill this e collected smart-contract-related posts from Ethereum StackExchange, as well as real-world smart contracts. We manually d these posts and contracts; using them to define 20 kinds of *contract defects*. We categorized them into indicating potential , availability, performance, maintainability and reusability problems. To validate if practitioners consider these contract as , we created an online survey and received 138 responses from 32 different countries. Feedback showed these contract defects rmful and removing them would improve the quality and robustness of smart contracts. We manually identified our defined t defects in 587 real world smart contract and publicly released our dataset. Finally, we summarized 5 impacts caused by t defects. These help developers better understand the symptoms of the defects and removal priority.

Terms—Empirical study, smart contracts, ethereum, contract defects

DUCTION

iderable success of decentralized cryptocurren- attracted great attention from both industry and itcoin [1] and Ethereum [2], [3] are the two most ptocurrencies whose global market cap reached by April 2018 [4]. A *Blockchain* is the underlying of cryptocurrencies, which runs a consensus pro- ntain a shared ledger to secure the data on the Both Bitcoin and Ethereum allow users to encode pts for processing transactions. However, scripts re not Turing-complete, which restrict the scenar- age. Unlike Bitcoin, Ethereum provides a more chnology named *Smart Contracts*.
ntracts are Turing-complete programs that run on ain, in which consensus protocol ensures their cor- on [2]. With the assistance of smart contracts, an apply blockchain techniques to different fields and finance. When developers deploy smart con- ereum, the source code of contracts will be com- *tecode* and reside on the blockchain. Once a smart reated, it is identified by a 160-bit hexadecimal d anyone can invoke this smart contract by sending transactions to the corresponding contract address. Ethereum uses *Ethereum Virtual Machine (EVM)* to execute smart contracts and transaction are stored on its blockchain.

A blockchain ensures that all data on it is immutable, i.e., cannot be modified, which means that smart contracts cannot be patched when bugs are detected or feature additions are desired. The only way to remove a smart contract from blockchain is by adding a *selfdestruct* [5] function in their code. Even worse, smart contracts on Ethereum operate on a permission-less network. Arbitrary developers, including attackers, can call the methods to execute the contracts. For example, the famous*DAO attack* [6] made the Decentralized Autonomous Organization (DAO) lose 3.6 million Ethers ($150/Ether on Feb 2019), which then caused a controversial hard fork [7], [8] of Ethereum.

It is thus critical to ensure that smart contracts are bug-free and well-designed before deploying them to the blockchain. In software engineering, a software defect is an error, flaw or fault in a computer program or system that causes it to produce an incorrect or unexpected result, or to behave in unintended ways [9], [10]. Contract defects are related to not only security issues but also design flaws which might slow down development or increase the risk of bugs or failures in the future. Detecting and removing contract defects helps increase software robustness and enhance development efficiency [11], [12]. Since the revolutionary changes of smart contracts compared to traditional softwares, e.g., the gas system, decentralized features, smart contracts contain many specific defects.

In this paper, we conduct an empirical study on defining smart contracts defects on Ethereum platform, the most popular decentralized platform that runs smart contracts. Please note that some previous works [13], [14], [15] focus on improving the quality of smart contracts from the security aspect. However, this is the first paper that aims to provide a

Xin Xia, and John Grundy are with the Faculty of Informa- logy, Monash University, Melbourne, VIC 3800, Australia. ch.Chen, Xin.Xia, John.Grundy}@monash.edu.
with the School of Information Systems, Singapore Manage- rsity, Singapore 188065. E-mail: davidlo@smu.edu.sg.
with the Department of Computing, Hong Kong Polytechnic 999077, Hong Kong. E-mail: csxluo@comp.polyu.edu.hk.
with the School of Computer Science and Engineering, Univer- nic Science and Technology of China, Chengdu 610054, China. ndragon@uestc.edu.cn.
ived 5 June 2019; revised 13 Apr. 2020; accepted 15 Apr. 2020. ion 20 Apr. 2020; date of current version 10 Jan. 2022. author: Xin Xia.)
or acceptance by L. Tan.
dentifier no. 10.1109/TSE.2020.2989002

Smart contracts are automated programs that can perform business logic without having to place trust in third parties.

Smart Contract Security Fixes: How Far Are We?

Yu et al., 2020

Defining Smart Contract Defects on Ethereum

Jiachi Chen, Xin Xia, David Lo, John Grundy, Xiapu Luo, and Ting Chen

Demir et al., 2019

Security Smells in Smart Contracts

Mehmet Demir[1], Manar Alalfi[1], Ozgur Turetken[2], Alexander Ferworn[1]

Security Code Recommendations for Smart Contract

Xiaocong Zhou[1], Yingye Chen[1], Hanyang Guo[2], Xiangping Chen[3], Yuan Huang[2,*]

SGUARD: Towards Fixing Vulnerable Smart Contracts Automatically

Tai D. Nguyen, Long H. Pham, Jun Sun

Zhou et al., 2023

Nguyen et al., 2021

Security smells

Security Defects

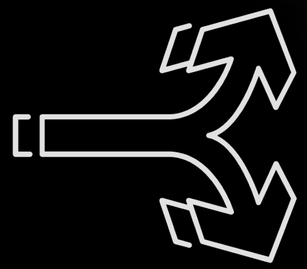
What about fixing strategies?

Which is the Gap to Bridge?

Literature Guidelines

to Fix Smart Contracts

Vulnerabilities



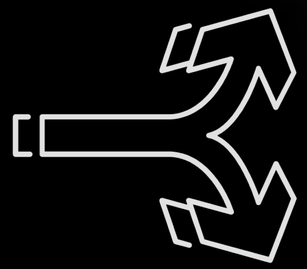
Do developers follow such guidelines?

Which is the Gap to Bridge?

Literature Guidelines

to Fix Smart Contracts

Vulnerabilities



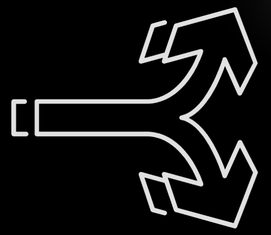
Do developers follow such guidelines?

To what extent?

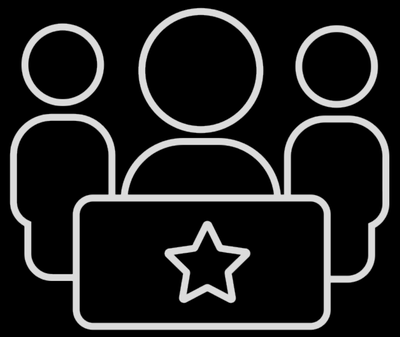
Which is the Gap to Bridge?

Literature Guidelines to Fix Smart Contracts Vulnerabilities

Do developers follow such guidelines?



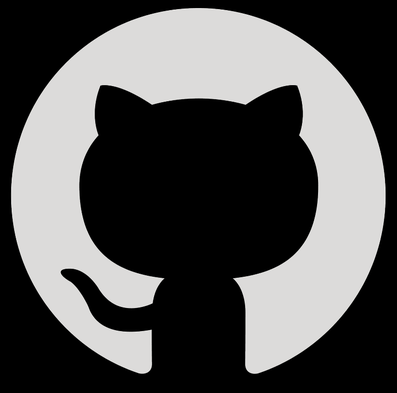
To what extent?



Literature-untracked fixes performed by Developers

Are these fixes valid?

Study Context

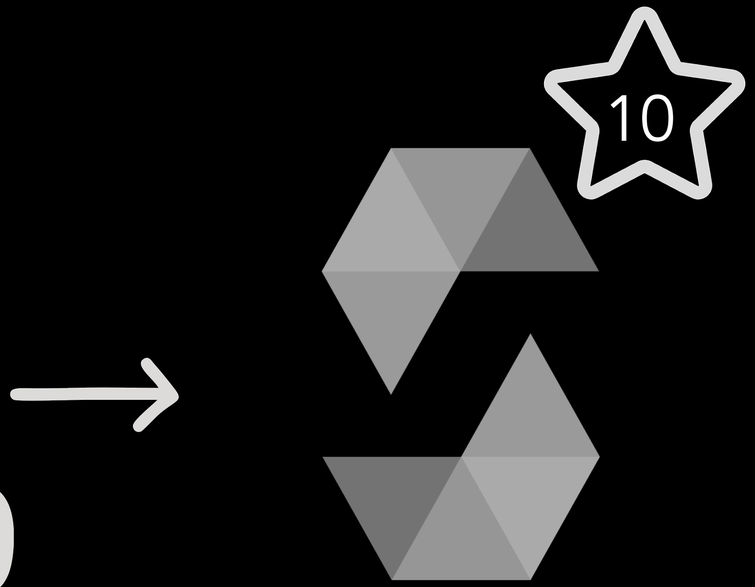


GitHub

Cloud-based service for software development and version control

Study Context

Cloud-based service for software development and version control



Solidity repositories with at least 10 stars

Study Context

Sampling Projects in GitHub for MSR Studies

Ozren Dabic, Emad Aghajani, Gabriele Bavota
SEART @ Software Institute, USI Università della Svizzera italiana, Lugano, Switzerland

***Abstract*—Almost every Mining Software Repositories (MSR) study requires, as first step, the selection of the subject software repositories. These repositories are usually collected from hosting services like GitHub using specific selection criteria dictated by the study goal. For example, a study related to licensing might be interested in selecting projects explicitly declaring a license. Once the selection criteria have been defined, utilities such as the GitHub APIs can be used to "query" the hosting service. However, researchers have to deal with usage limitations imposed by these APIs and a lack of required information. For example, the GitHub search APIs allow 30 requests per minute and, when searching repositories, only provide limited information (*e.g.*, the number of commits in a repository is not included). To support researchers in sampling projects from GitHub, we present GHS (GitHub Search), a dataset containing 25 characteristics (*e.g.*, number of commits, license, etc.) of 735,669 repositories written in 10 programming languages. The set of characteristics has been derived by looking for frequently used project selection criteria in MSR studies and the dataset is continuously updated to (i) always provide fresh data about the existing projects, and (ii) increase the number of indexed projects. The GHS dataset can be queried through a web application we built that allows to set many combinations of selection criteria needed for a study and download the information of matching repositories: `https://seart-ghs.si.usi.ch`.**

***Index Terms*—GitHub, search, sampling repositories**

I. Introduction

The amount of data available in software repositories is growing faster than ever. At the time of writing, GitHub [1] hosts over 80 Million public repositories[1] accounting for over 1 billion commit activities. Such an unprecedented amount of software data represents the main ingredient of many Mining Software Repositories (MSR) studies.

One of the fist steps in MSR studies consists in selecting the subject projects, *i.e.,* the software repositories to analyze in order to answer the research questions (RQs) of interest. Such a step is crucial to achieve generalizability of the findings and ensure that the selected projects result in useful data points for the goal of the study. For example, a study investigating the types of issues reported in GitHub [2] requires the selection of repositories regularly using the GitHub integrated issue tracker. Instead, a study interested in the pull request (PR) process of OSS projects [3] must ensure that the subject systems actually adopt the PR mechanism (*e.g.,* by verifying that at least n PRs have been submitted in a given repository). In addition to RQ-specific selection criteria, several studies adopt specific filters to exclude toy and personal projects. For example, previous works excluded repositories having a low number of stars [4], commits [5], or issues [2].

[1] `https://api.github.com/search/repositories?q=is:public+fork:true`

Once the selection criteria have been defined, software repositories satisfying them must be identified. Frequently, the search space is represented by all projects hosted on GitHub that, as previously said, are tens of millions. To query such a collection of repositories, developers can use the official GitHub APIs [6] that, however, come with a number of limitations both in terms of number of requests that can be triggered and information that can be retrieved. For example, the GitHub search API allows for a maximum of 30 requests per minute and each request can return at most 100 results. Only searching for some basic information about the public Java repositories hosted on GitHub would require, at the time of writing, ∼160k requests (∼88 hours). If additional information is required for each repository (*e.g.,* its number of commits), additional requests must be triggered, making the process even more time expensive. Moreover, setting an appropriate value for the selection criteria (*e.g.,* a project must have at least 100 commits) without having an overall view of the available data can be tricky. For instance, researchers cannot easily select the top 10% repositories in terms of number of commits without firstly collecting this information for the entire population. Finally, given a selection criteria, the GitHub search API provides at most the first 1,000 results (through 10 requests). This means there is no easy way to retrieve all matching results for a selection criteria if it exceed this upper bound.

To support developers in mining GitHub, several solutions have been proposed. Popular ones are GHTorrent [7] and GHArchive [8]. Both projects continuously monitor public events on GitHub and archive them. While the value of these tools is undisputed, as the benefits they brought to the research community, they do not provide a handy solution to support the sampling of projects on GitHub accordingly to the desired selection criteria. For example, computing the number of commits, issues, etc. for a repository in GHTorrent would require MySQL queries aimed at joining multiple tables.

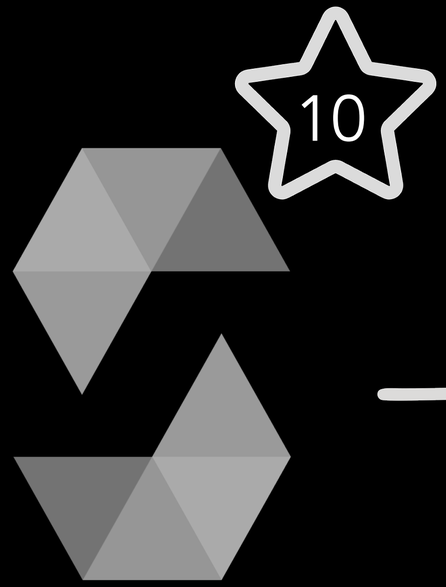
We present GHS (GitHub Search) [9], a dataset and a tool to simplify the sampling of projects to use in MSR studies. GHS continuously mines a set of 25 characteristics of GitHub repositories that have been often used as selection criteria in MSR studies and that, accordingly to our experience in the field, can be useful for sampling projects (*e.g.,* adopted license, number of commits, contributors, issues, and pull requests).

The tool behind GHS can be configured to mine projects written in specific programming languages. As of today, it mined information about over 700k repositories written in 10 different languages (*i.e.,* Python, Java, C++, C, C#, Objective-C, Javascript, Typescript, Swift, and Kotlin).

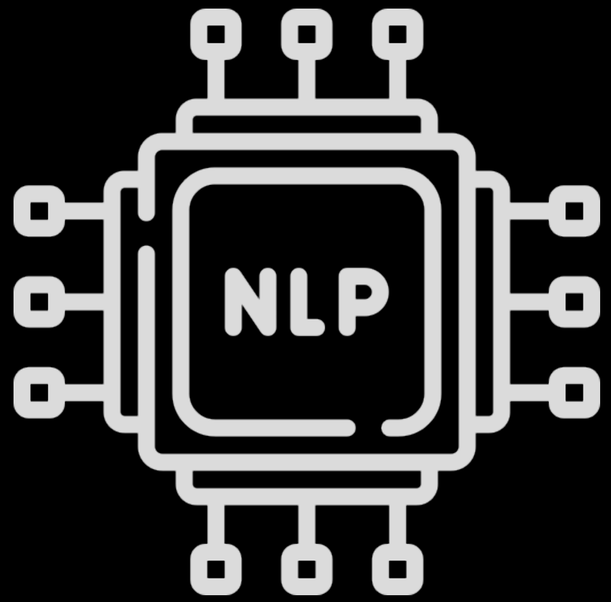
"A threshold of 10 stars offers a reasonable compromise between data quality and the time required to mine and continuously update all projects."

Study Context

Cloud-based service for software development and version control

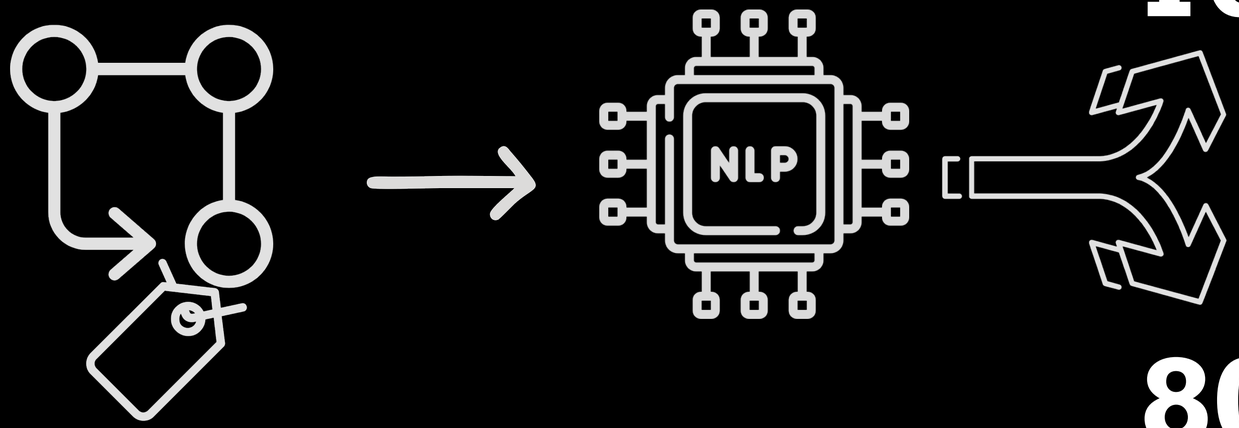


Solidity repositories with at least 10 stars



Filtering commits applying Spacy NLP-based filter on commit messages

NLP-based filter evaluation



Separate sample of 400 random commits

NLP-based filter

100% Specificity

80% Recall

Research Questions

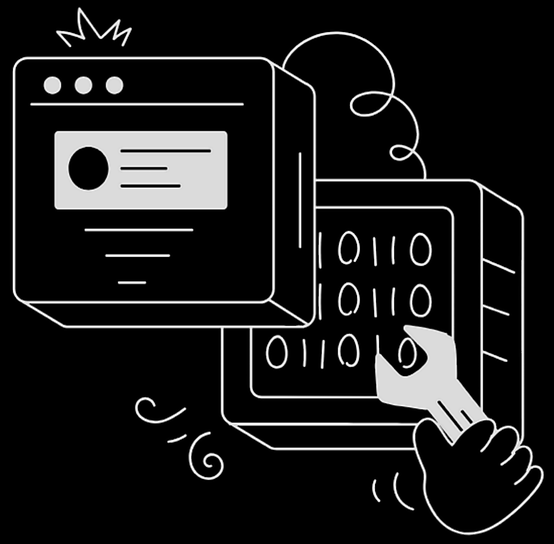
- **RQ1**: *To what extent do developers adhere to the fixing guidelines provided in the literature?*

Research Questions

- **RQ1**: *To what extent do developers adhere to the fixing guidelines provided in the literature?*
- **RQ2**: *What are the valid fixing approaches beyond those documented in the literature?*

Experimental Procedure

RQ1: To what extent do developers adhere to the fixing guidelines in the literature?



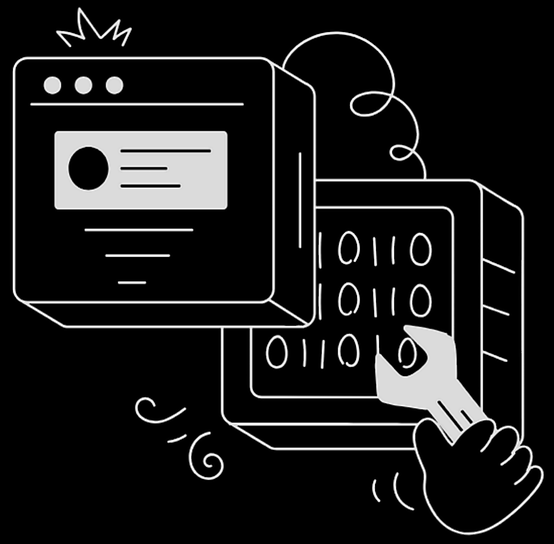
Dataset of **candidate** fixing commits

Ensuring commit relevance with double checked manual analysis

Is each commit relevant for our study?

Experimental Procedure

RQ1: To what extent do developers adhere to the fixing guidelines in the literature?



Dataset of **candidate** fixing commits

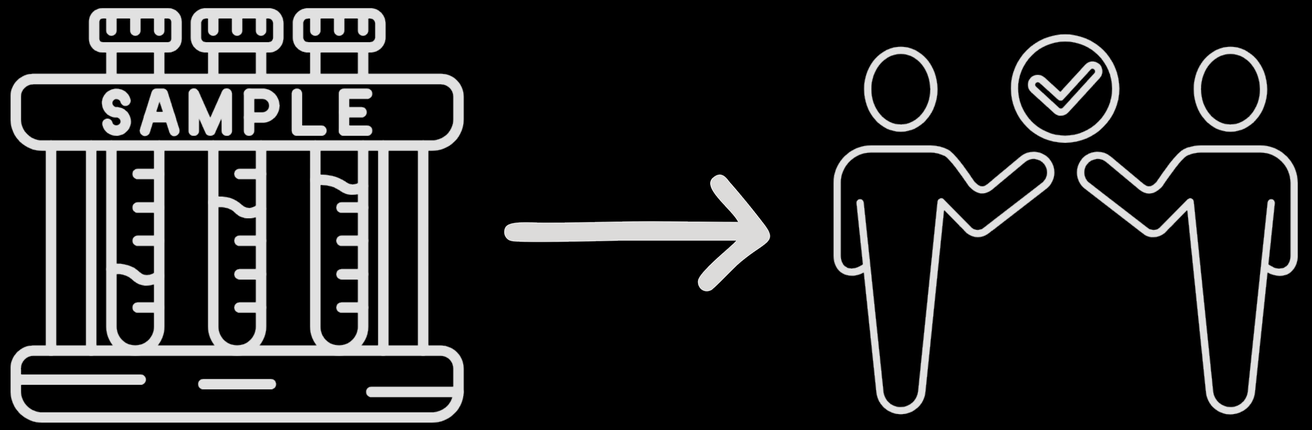
Ensuring commit relevance with double checked manual analysis

Focus on DASP

TOP 10 Taxonomy

Experimental Procedure

RQ1: To what extent do developers adhere to the fixing guidelines in the literature?



Sample of Interest of

364 relevant commits

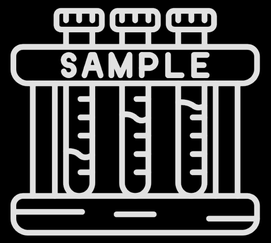
High Level of Agreement

With Cohen’s Kappa value

of 0.89

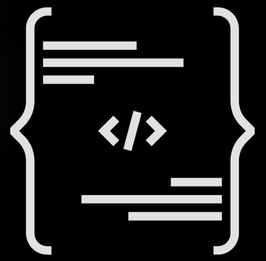
Experimental Procedure

RQ1: To what extent do developers adhere to the fixing guidelines in the literature?



Starting from the

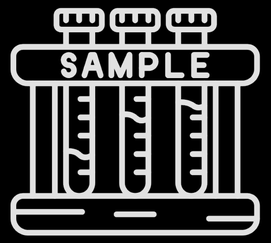
Sample of Interest



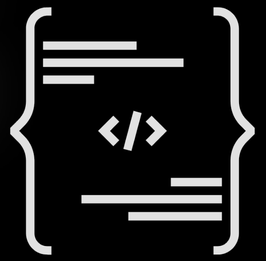
Each fix is analyzed by two independent evaluators.

Experimental Procedure

RQ1: To what extent do developers adhere to the fixing guidelines in the literature?



Starting from the Sample of Interest

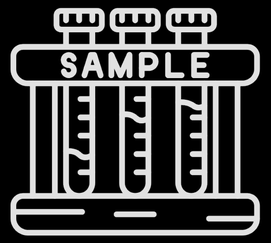


Is it known in literature?

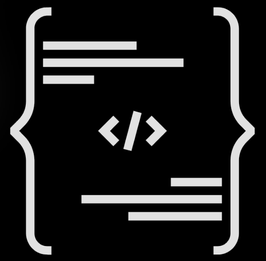
Each fix is analyzed by two independent evaluators.

Experimental Procedure

RQ1: To what extent do developers adhere to the fixing guidelines in the literature?

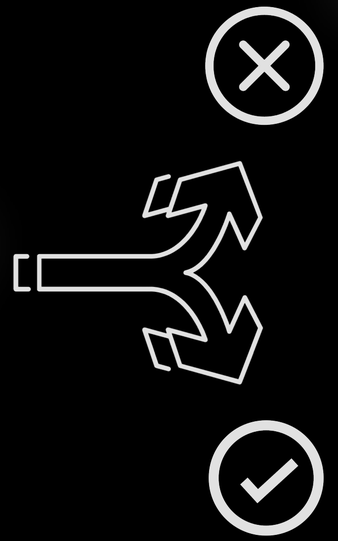


Starting from the Sample of Interest



Each fix is analyzed by two independent evaluators.

Is it known in literature?



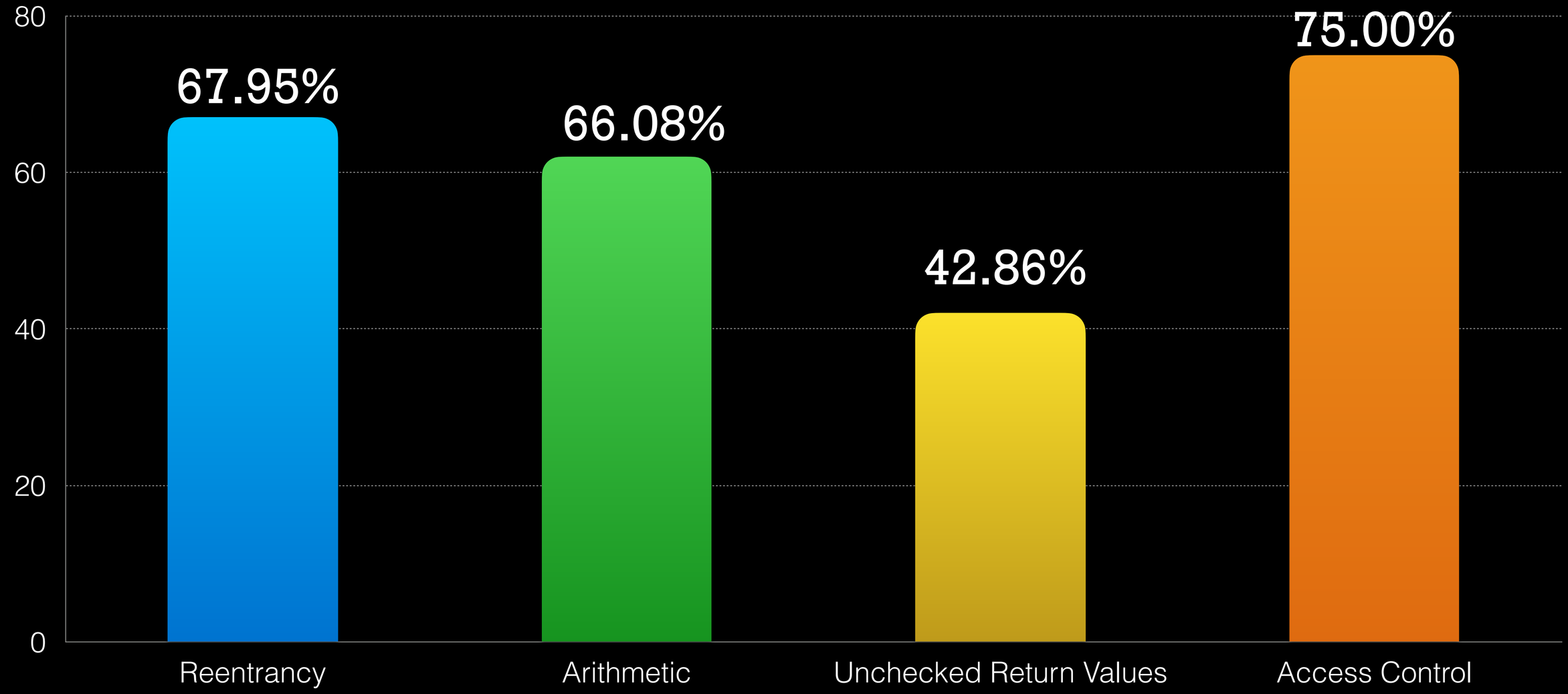
Input for RQ2

Classify according to DASP Taxonomy


I’ll follow
something
known
I’ll fix that in
my way

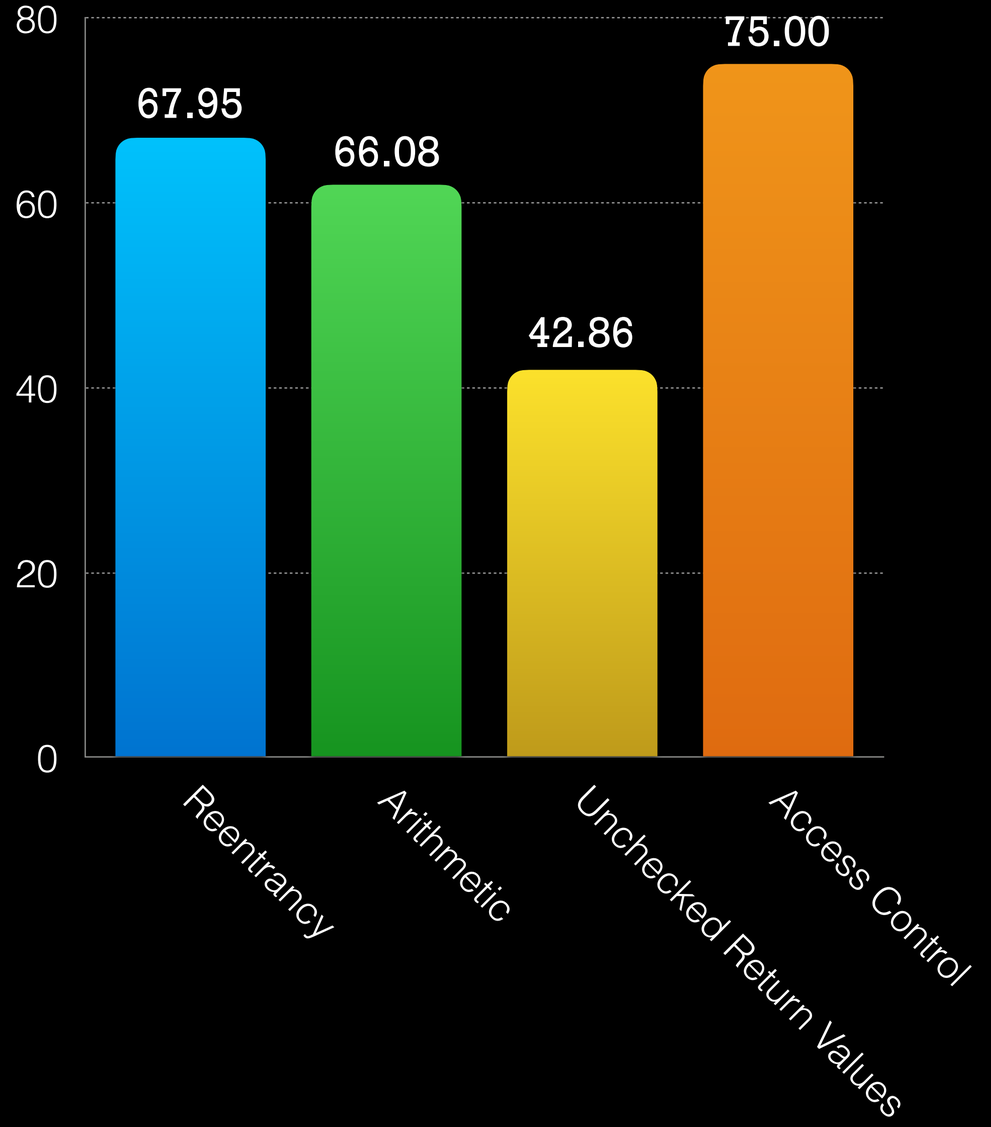
Experimental Results

RQ1: To what extent do developers adhere to the fixing guidelines provided in the literature?



Experimental Results

RQ1: To what extent do developers adhere to the fixing guidelines provided in the literature?

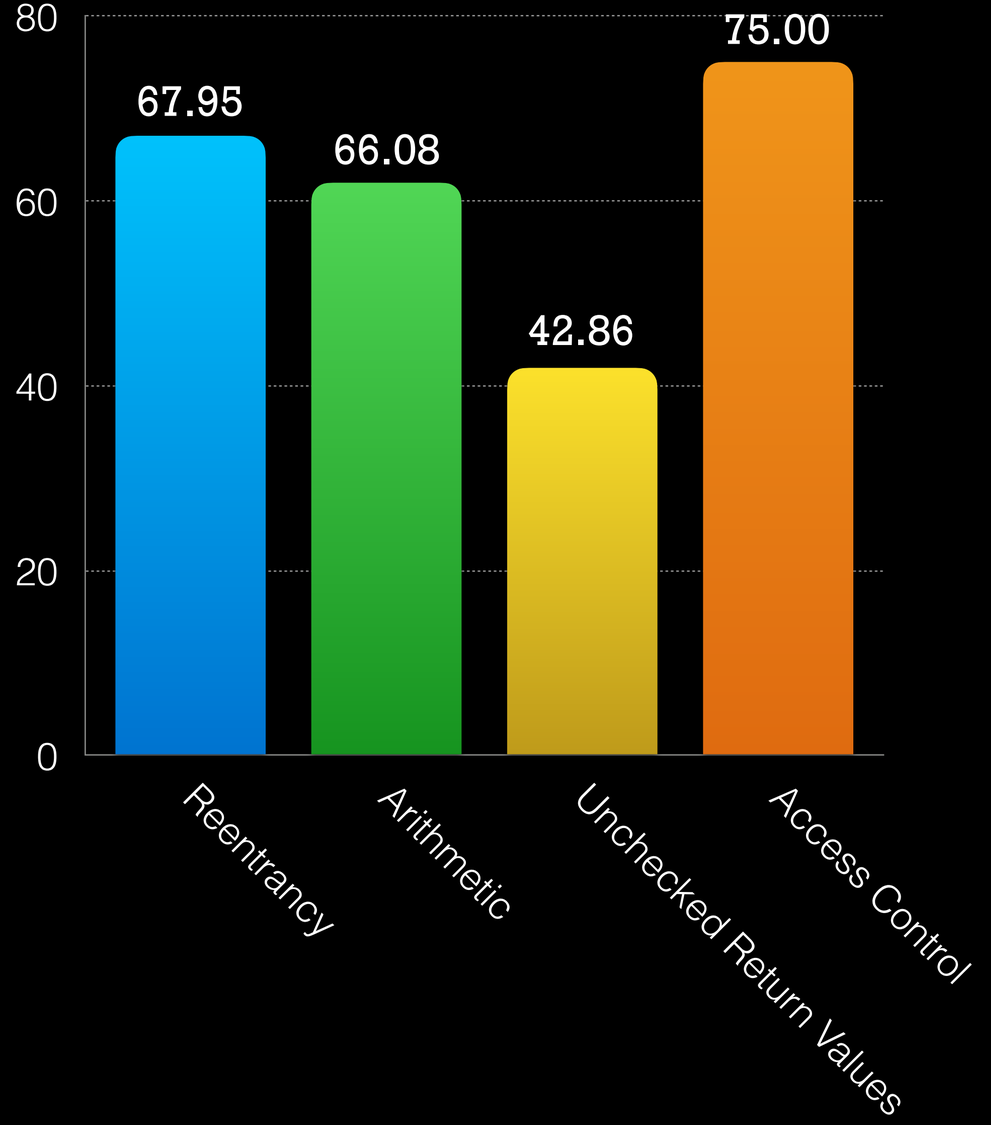


DOS, Bad Randomness, Short Address, Time Manipulation and Front-Running fixes do not follow literature guidelines

Only guidelines related to the most studied vulnerabilities are followed

Experimental Results

RQ1: To what extent do developers adhere to the fixing guidelines provided in the literature?



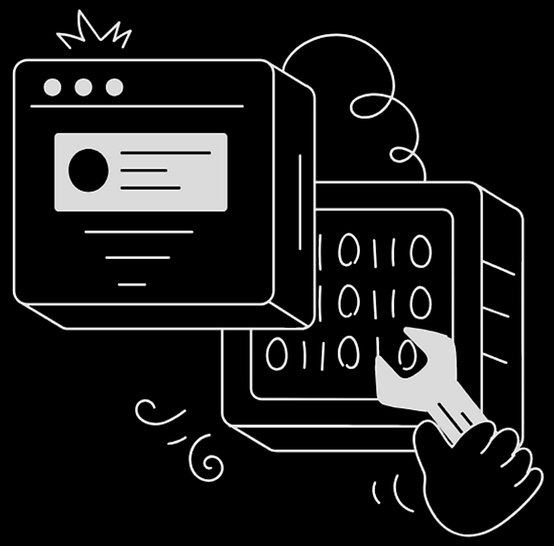
In total, 221 commits (60.55%) followed known academic strategies

The remaining 143 commits served as input for *RQ2*


Sometimes, worse is better.
New fixes inside

Experimental Procedure

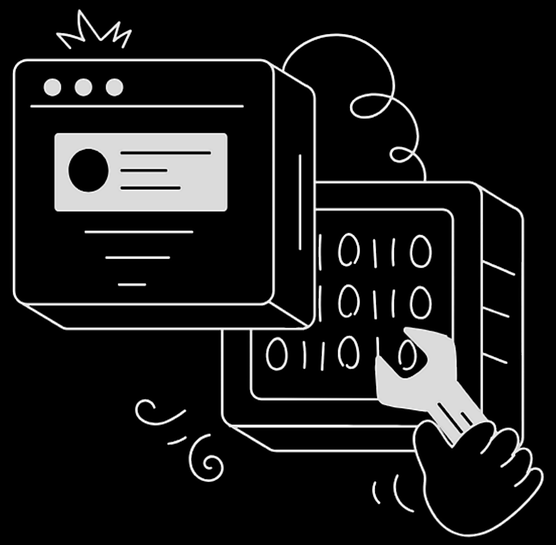
RQ2: What are the valid fixing approaches beyond those documented in the literature?



Analysis of each change made by developers

Experimental Procedure

RQ2: What are the valid fixing approaches beyond those documented in the literature?

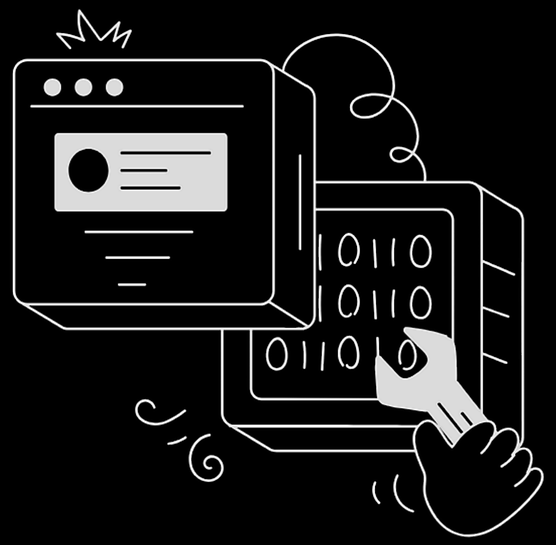


Analysis of each change made by developers

Conflict resolution

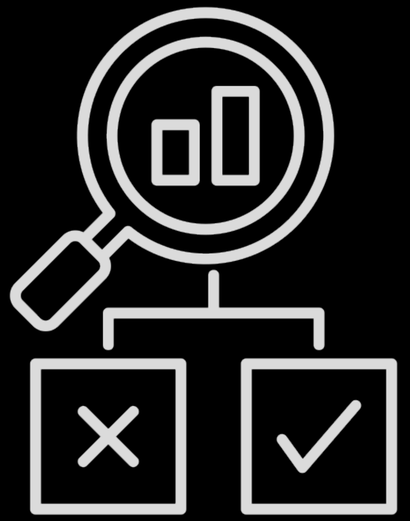
Experimental Procedure

RQ2: What are the valid fixing approaches beyond those documented in the literature?



Analysis of each change made by developers

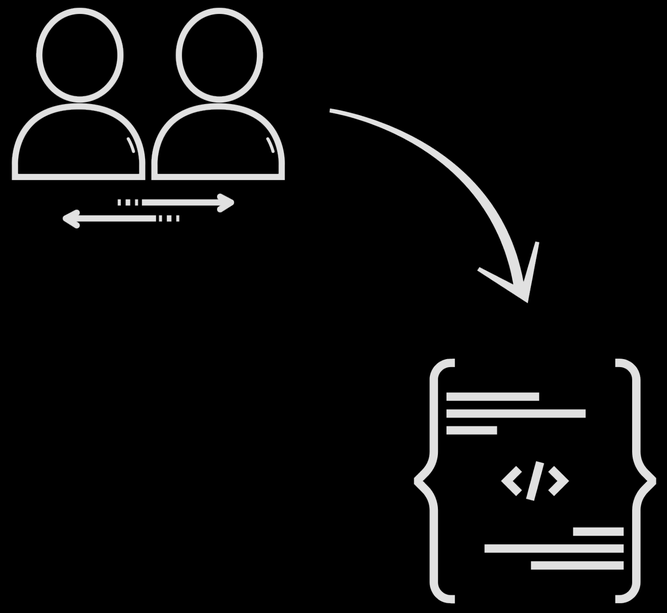
Conflict resolution



Further Validations

Experimental Procedure

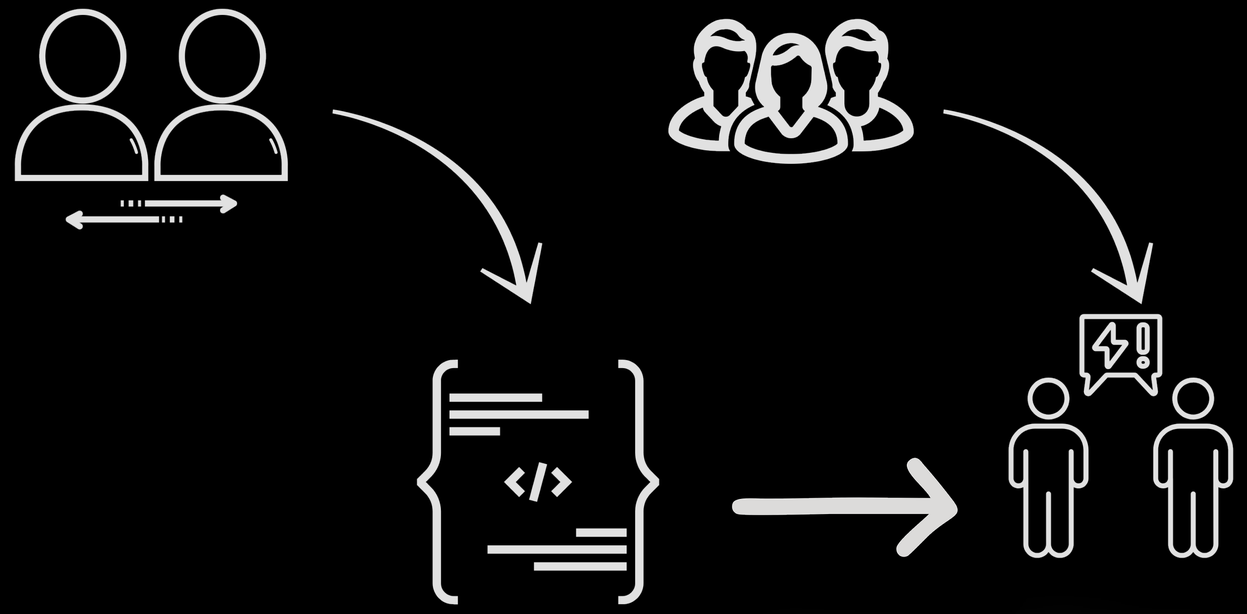
RQ2: What are the valid fixing approaches beyond those documented in the literature?



Each change is analyzed by two independent evaluators.

Experimental Procedure

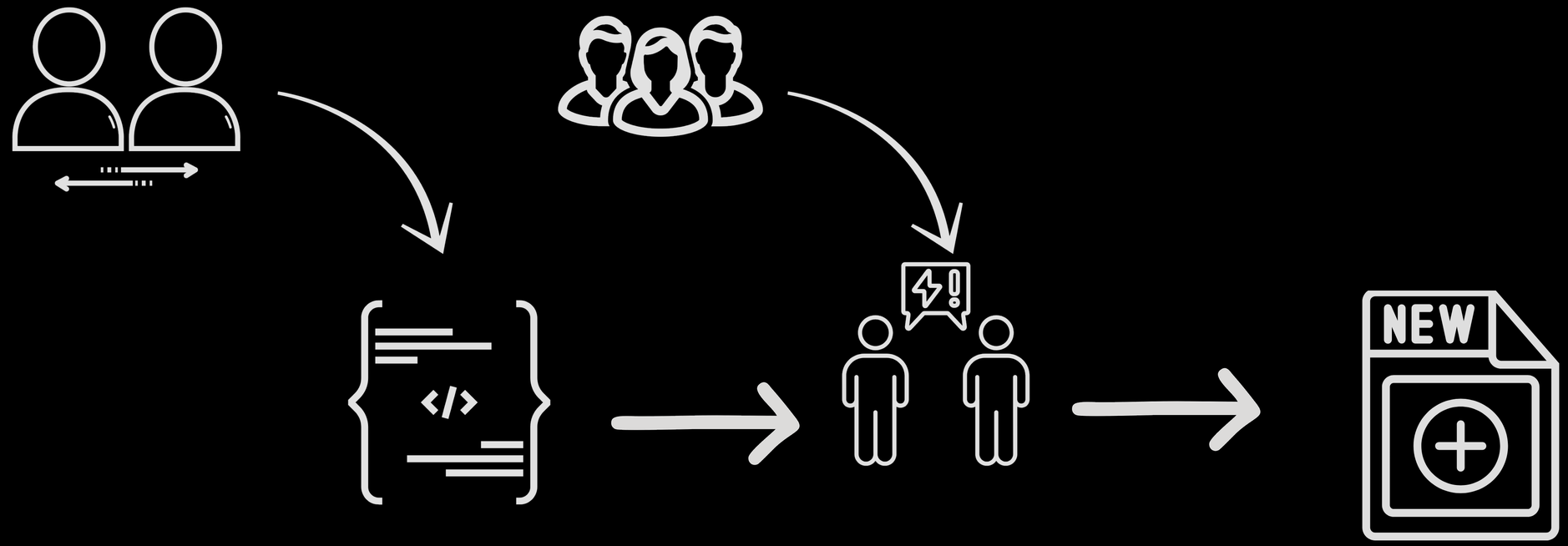
RQ2: What are the valid fixing approaches beyond those documented in the literature?



Each conflict is resolved
through discussion,
involving three evaluators

Experimental Procedure

RQ2: What are the valid fixing approaches beyond those documented in the literature?



New Employable Fixes

Experimental Results

RQ2: What are the valid fixing approaches beyond those documented in the literature?

Experimental Results

RQ2: What are the valid fixing approaches beyond those documented in the literature?

35 New Fixes Strategies

With 27 different Approaches

Experimental Results

RQ2: What are the valid fixing approaches beyond those documented in the literature?

35 New Fixes Strategies

With 27 different Approaches

Covering all the categories of the DASP TOP 10 but not Unknows

Further Validations of the Found Fixing Procedures

Further Validations of the Found Fixing Procedures

- Effectiveness: How applicable do you think the fix is to similar or recurring cases?

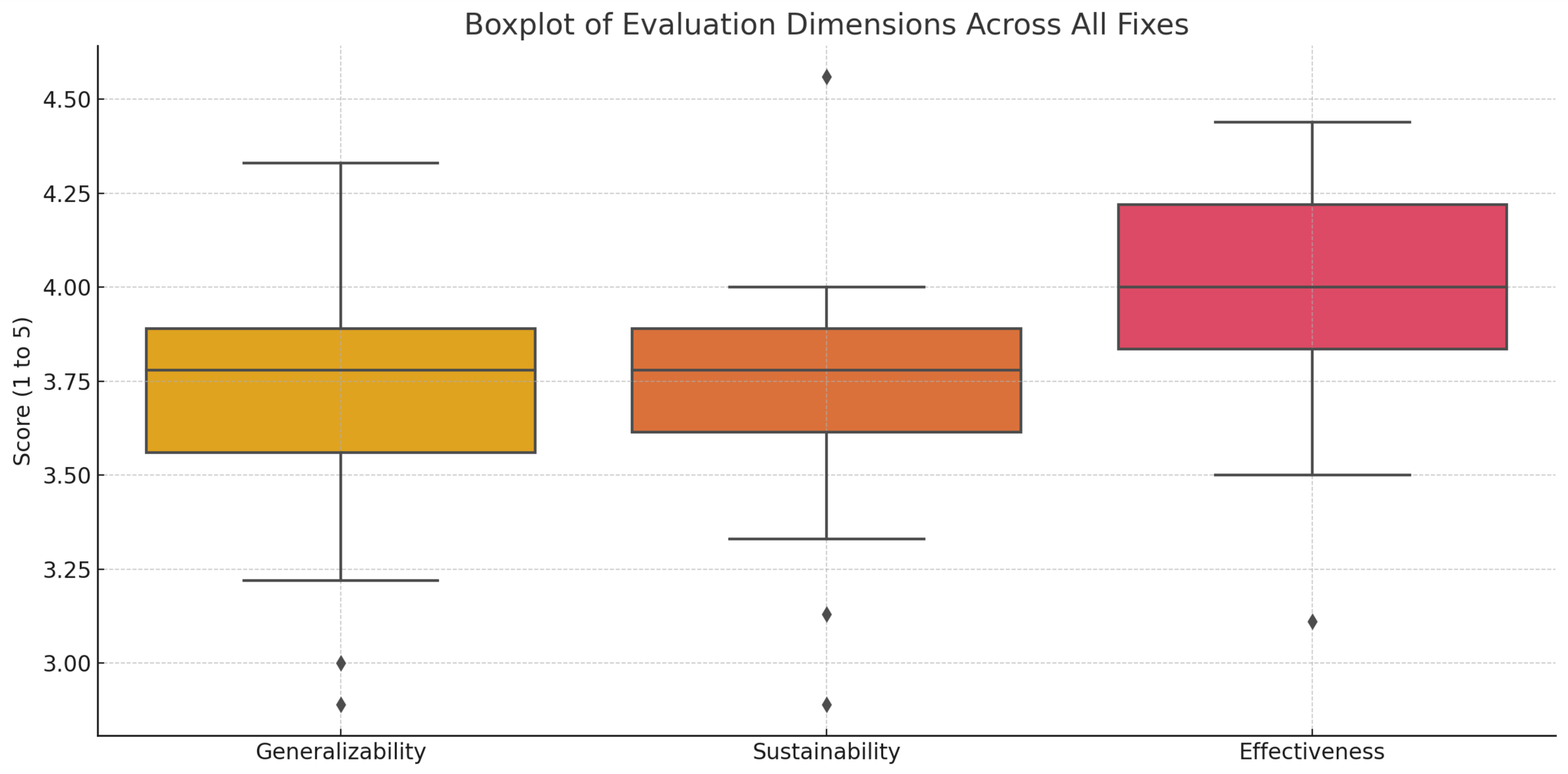
Further Validations of the Found Fixing Procedures

- Effectiveness: How applicable do you think the fix is to similar or recurring cases?
- Generalizability : To what extent do you believe the fix can remain effective and manageable over time, even as the context or codebase changes?

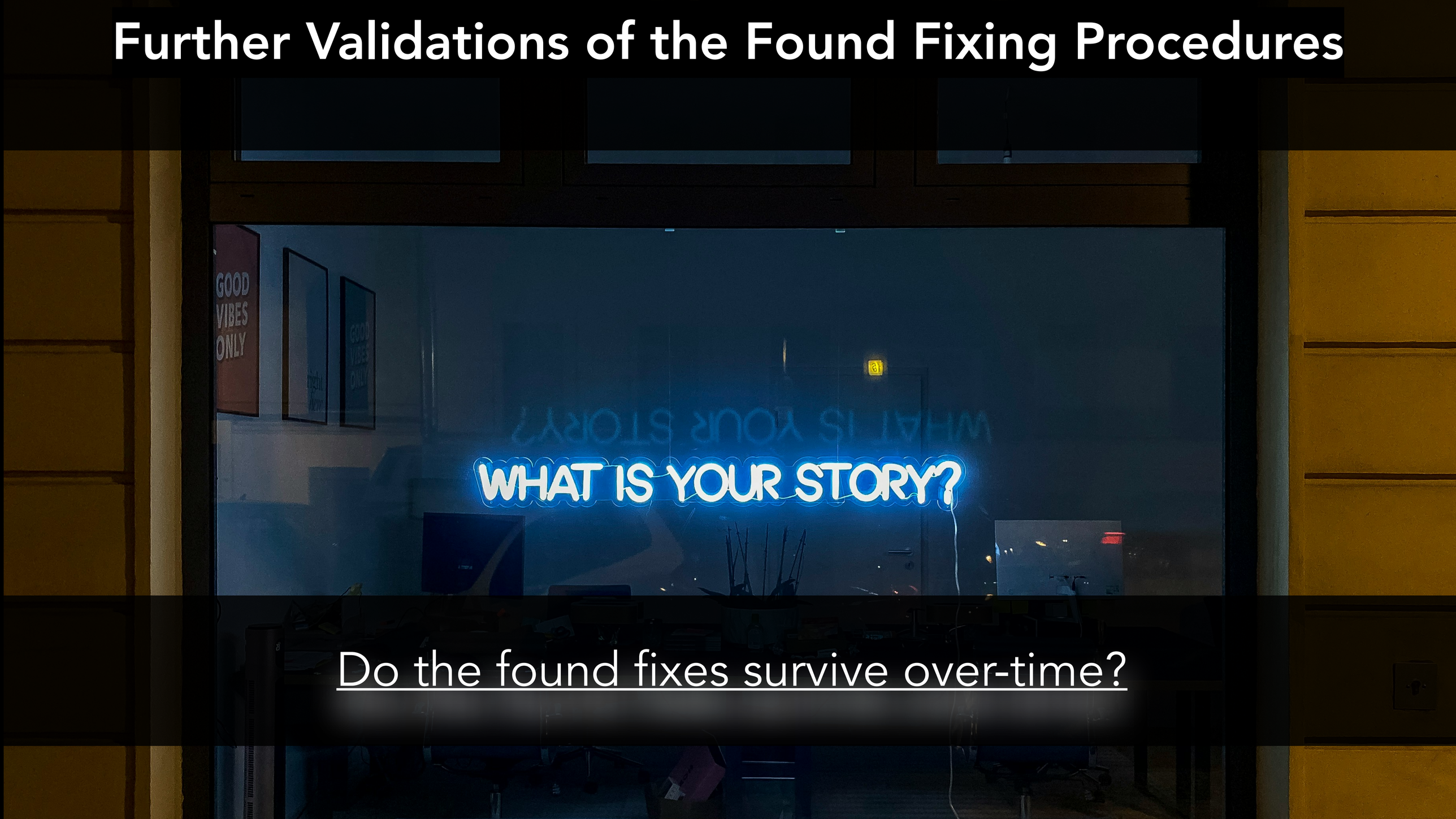
Further Validations of the Found Fixing Procedures

- Effectiveness: How applicable do you think the fix is to similar or recurring cases?
- Generalizability : To what extent do you believe the fix can remain effective and manageable over time, even as the context or codebase changes?
- Long-term Sustainability : How effectively do you think the fix resolves the identified vulnerability?

Further Validations of the Found Fixing Procedures

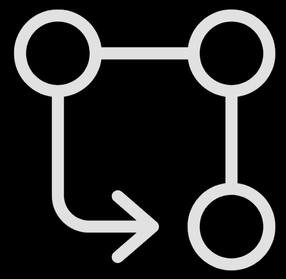


Further Validations of the Found Fixing Procedures

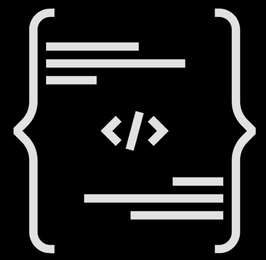


Do the found fixes survive over-time?

Further Validations of the Found Fixing Procedures

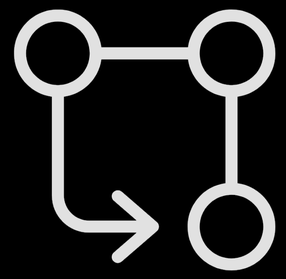


For each commit, check all the later commit that modify the file involved

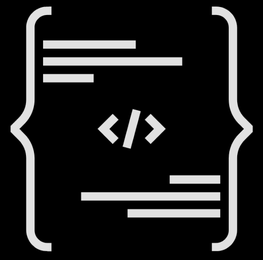


If at least one of the lines modified by the fixing commit changes

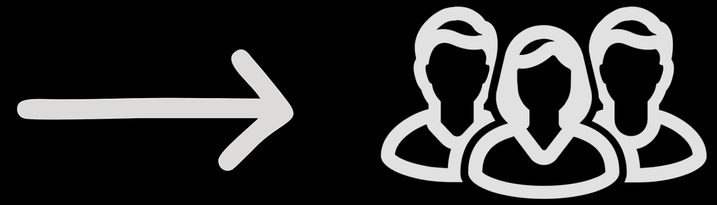
Further Validations of the Found Fixing Procedures



For each commit, check all the later commit that modify the file involved

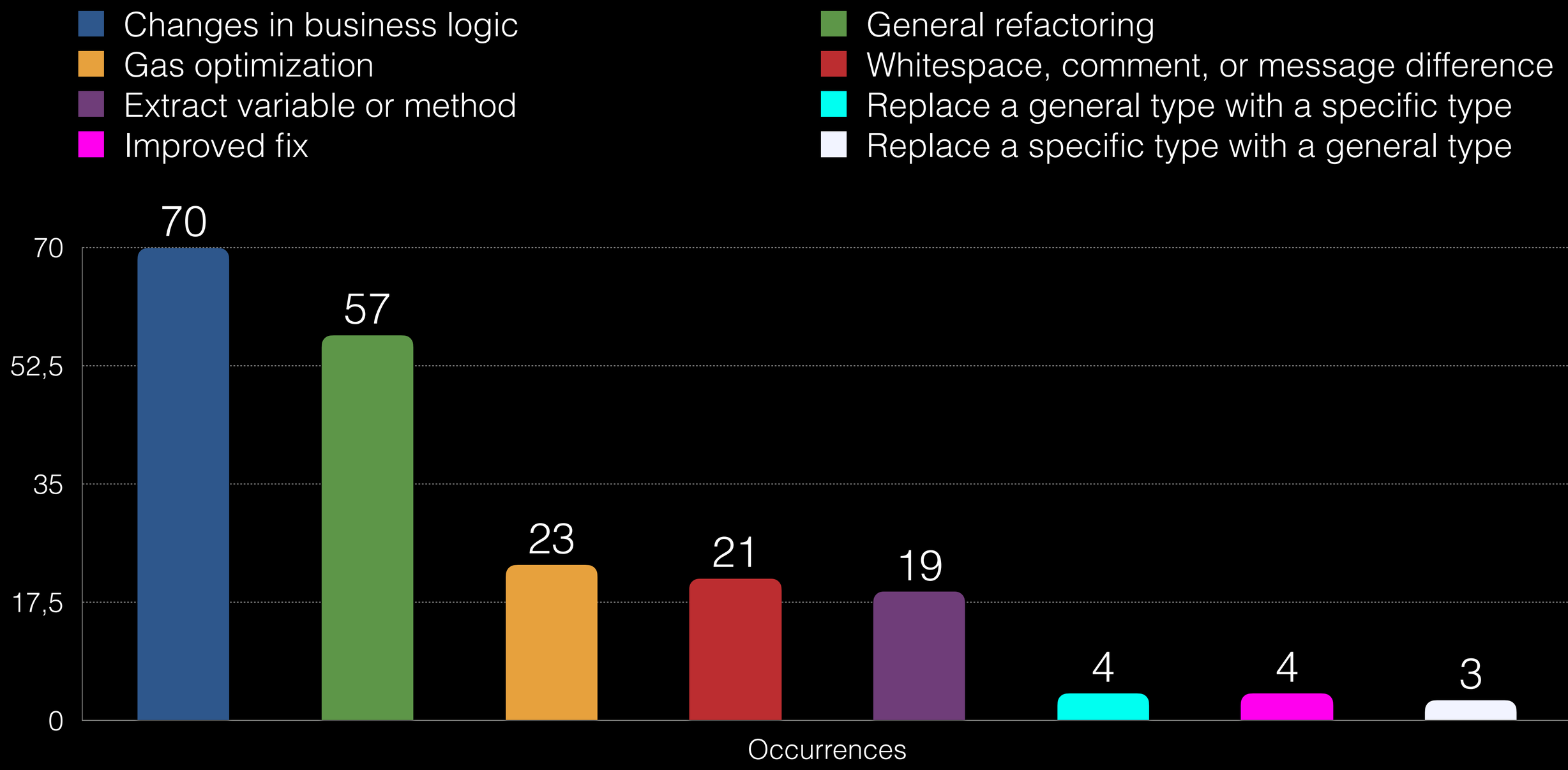


If at least one of the lines modified by the fixing commit changes



Manual checking to state if the fixes survived to later commits

Further Validations of the Found Fixing Procedures



Check out the Preprint!

Bridging the Gap: A Comparative Study of Academic and Developer Approaches to Smart Contract Vulnerabilities

Francesco Salzano · Lodovica Marchesi · Cosmo Kevin Antenucci · Simone Scalabrino · Roberto Tonelli · Rocco Oliveto · Remo Pareschi

Received: date / Accepted: date

Abstract In this paper, we investigate the strategies adopted by Solidity developers to fix security vulnerabilities in smart contracts. Vulnerabilities are categorized using the DASP TOP 10 taxonomy, and fixing strategies are extracted from GitHub commits in open-source Solidity projects. Each commit was selected through a two-phase process: an initial filter using natural language processing techniques, followed by manual validation by the authors. We analyzed these commits to evaluate adherence to academic best practices. Our results show that developers often follow established guidelines for well-known vulnerability types such as Reentrancy and Arithmetic. However, in less-documented categories like Denial of Service, Bad Randomness, and Time

F. Salzano
University of Molise, Italy
E-mail: francesco.salzano@unimol.it

L. Marchesi
University of Cagliari, Italy
E-mail: lodovica.marchesi@unica.it

C. K. Antenucci
University of Molise, Italy
E-mail: c.antenucci2@studenti.unimol.it

S. Scalabrino
University of Molise, Italy
E-mail: simone.scalabrino@unimol.it

R. Tonelli
University of Cagliari, Italy
E-mail: roberto.tonelli@unica.it

R. Oliveto
University of Molise, Italy
E-mail: rocco.oliveto@unimol.it

R. Pareschi
University of Molise, Italy
E-mail: remo.pareschi@unimol.it

Status: Submitted to Empirical Software Engineering and revised after revisions

Preprint available

Bridging Academia and Practice in Smart Contract Security

Thank you for the attention!

* francesco.salzano@unimol.it

🎓University of Molise

📚 University of Cagliari

Francesco Salzano*🎓

Lodovica Marchesi 📚

Cosmo Kevin Antenucci 🎓

Simone Scalabrino 🎓

Roberto Tonelli 📚

Rocco Oliveto 🎓

Remo Pareschi 🎓

7th Distributed Ledger Technologies Workshop (DLT2025)