Paper ID:

Technological rule:

To achieve effect/change in situation/context use solution/intervention

Problem Instance

Problem Understanding:

Evaluation approach:

Solution

Solution design approach:

Relevance:

Rigor:

Novelty:

Paper Title:

Generic Visual Abstract

Technological rule: To achieve effect/change in situation/context apply solution/intervention

Problem Instance

Concrete case of the problem described in the Technological Rule in terms of real data and/or stakeholders' needs

Problem Understanding: Studies to support conceptual understanding of the problem instance

Evaluation approach: Studies to evaluate the impact of the solution on the problem instance

Solution

Concrete implementation of the solution described in the Technological Rule

Solution design approach: Studies or theoretical foundations validating core design choices

Relevance: Characteristics of the context that are likely to impact applicability and potential value of the proposed solution

Rigor: Characteristics of the three knowledge creating activities (problem understanding, solution design and in context evaluation) that adds to the strength of the empirical support of the Technological Rule

Novelty: Positioning of the Technological Rule in terms of previous knowledge

Paper ID: Alimadadi2014

Technological rule: To improve developers comprehension when understanding event-based interactions in JavaScript capture and visualize low-level event-based interactions mapped to a higher level behavioural model.

Problem Instance

Understanding the dynamic, event-driven, and asynchronous behaviour of JavaScript applications

Problem Understanding: The reported challenges of understanding Javascript was confirmed by web developers at a large software company.

Solution

An inferred behavioural model of execution is presented in an interactive visualization to provide deeper understanding of entities involved in the execution and their interactions at runtime.

Solution design approach: Instrumenting Javascript to collect detailed execution traces, inferring a behavioural model of its execution showing DOM mutations.

Evaluation approach: An experiment in industry and a lab study showing improvements in task accuracy by 160% , and task completion time by 47%, gained feedback on the interactive visual interface.

Relevance: Common [JavaScript] comprehension tasks are supported by the proposed visualization approach, which could help many web developers.

Rigor: Two controlled experiments were conducted with 34 users to measure and compare task completion time and accuracy, with preferred tools selected by a control group.

Novelty: 1) Capturing and inferring temporal and causal relations between JavaScript execution, DOM mutations, and server communications; and 2) A behavioural model of program behaviour, displayed as an interactive visualization in an open source tool.

Paper Title: Understanding JavaScript Event-Based Interactions

Paper ID: Avgerino2014

Technological rule: To increase efficiency and effectiveness when using symbolic execution alternate between static and dynamic symbolic execution

Problem Understanding: The problem is well known and reported in several studies.

Problem Instance

How to increase bug detection in created benchmarks (using binaries from Debian Wheezy and Sid)

Evaluation approach: Use of benchmarks to compare with previous work, Applied MergePoint with veritesting to 33,248 binaries

Solution

Implement MergePoint with veritesting. Veritesting alternates between SSE and DSE. The alternation mitigates the difficulty of solving formulas, while alleviating the high overhead associated with a path-based DSE approach.

Solution design approach: It builds on and improves previous work (MergePoint) wrt the path explosion problem

Relevance: Lack of efficiency when using symbolic execution

Rigor: Large-scale experiment on 33,248 programs from Debian Linux. MergePoint generated billions of SMT queries, hundreds of millions of test cases, millions of crashes, and found 11,687 distinct bugs

Novelty: Tested an order of magnitude more applications than have been tested by prior symbolic execution research. We analyzed each application for less than 15 minutes per experiment. We improve open source software by finding over 10,000 bugs and generating millions of test cases.

Paper Title: Enhancing Symbolic Execution with Veritesting

Paper ID: Bell2014

Technological rule: To reduce overhead during unit testing use unit test virtualization

Problem Understanding: A motivating study was carried out: 591 OSS (selected from 1200) projects were studied

Problem Instance

Programmers start one process for each test case in order to avoid hidden dependencies through static declarations. This requires extra execution time. Programmers use TSM to solve this which decreases fault finding ability.

Evaluation approach: Authors develop a tool called Unit test virtualization – VMVM to carry out unit test

Solution

Authors introduce Unit Test Virtualization, a technique whereby the side effects of each unit test are isolated from other tests, eliminating the need to restart the system under test with every new test

Solution design approach: Design approach involved: a motivational study of 1200 open source projects showing that developers isolate tests; presentation of Unit Test Virtualization technique; Implementation of the technique with the VmVm tool

Relevance: Test case selection and isolation for developers working on large applications with long running test suites

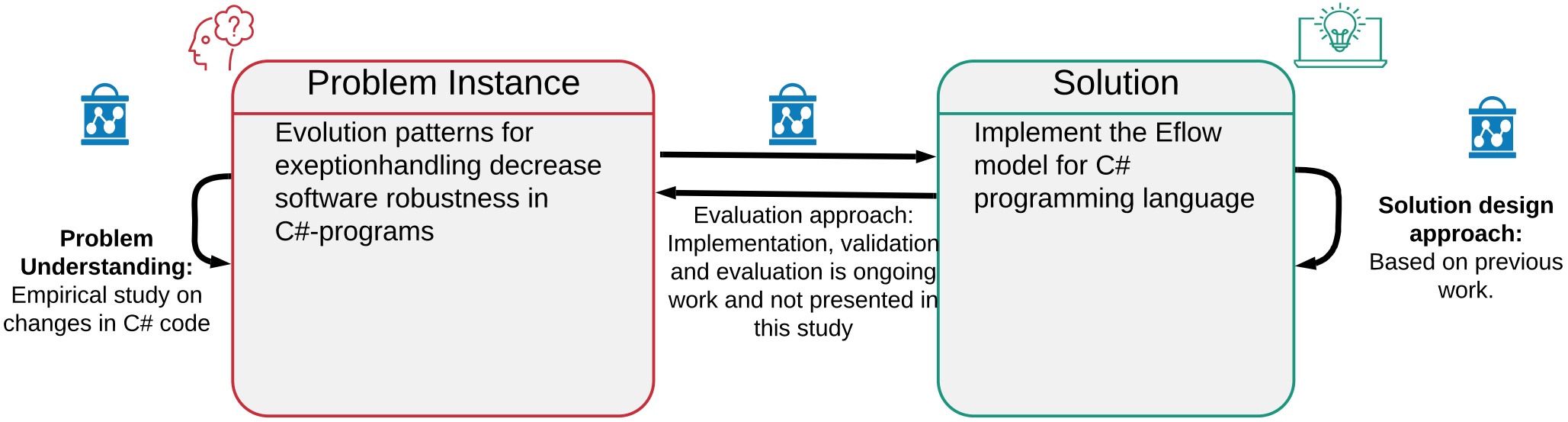
Rigor: Two Empirical studies to evaluate performance of VMVM vs. TSM wrt: reduction of execution time (RT); reduction in fault-finding ability (RF); reduction in test suite size (TS). One study conducted on 19 versions of 4 applications; a second study conducted on 20 OS Java applications.

Novelty: 1) Motivational study of the test suites of 1,200 open source projects showing that developers isolate tests 2.) Unit Test Virtualization technique to efficiently isolate test cases 3.) Implementation and evaluation of VmVm tool to show its effiacy in reducing test suite runtime and maintaining fault-finding properties

Paper Title: Unit Test Virtualization with VMVM

Paper ID: Cacho2014

Technological rule: To improve the simultaneous satisfaction of software maintainability and reliability in built-in EHM in programming languages apply explicit exception channels, which support modular representation of global exceptional-behaviour properties.



Relevance: Built-in exception handling mechanisms affect robustness negatively. Relevant for mainstream programming languages (C#, Ruby, Python and many others) that provide built-in exception handling mechanisms

Rigor: Changes to both normal and exceptional code in 119 versions extracted from 16 software projects were analysed. Analysis was based on common models for exception handling mechanisms.

Novelty: Empirical knowledge on how programs using maintenance-driven facilities for exception handling evolve over time (impact and side effects on robustness)

Paper Title: Trading Robustness for Maintainability: An Empirical Study of Evolving C# Programs

Paper ID: Inozemtseva2014

Technological rule: When assessing test suite effectiveness in software testing of Java programs do NOT rely on coverage measures

Problem Understanding: Several studies are published, failing to come to a consenus about the relation.

Problem Instance

A proxy for the ability of a test suite to detect fault is needed. Coverage is often used, but the nature and the strength of the relation between coverage and test suite effectiveness is not clear.

Evaluation approach: Evaluated 31,000 test suites for 5 systems consisting of up to 724 KLOC, measured the statement coverage, decision coverage, and modified condition coverage, using mutation testing to evaluate their fault detection effectiveness.

Solution

Coverage, while useful for identifying under-tested parts of a program, should NOT be used as a quality target because it is not a good indicator of test suite effectiveness.

Solution design approach: Literature studies (e.g. [29, 34])

Relevance: Using coverage for understanding of the effectiveness of automatic unit tests by practitioners planning unit tests

Rigor: The recommendation is ensured by investigating a large set of (generated) mutants of 5 programs, and test cases for thee programs.

Novelty: The negative rule has previously been seen a commonly accepted fact.

Paper Title: Coverage Is Not Strongly Correlated with Test Suite Effectiveness

Paper ID: Liu2014

Technological rule: To achieve improved performance bug identification ability in Android app development apply static analysis wrt lengthy oper. in the main thread and violations of the View-holder design pattern

Problem Understanding: Study of the bug reports and related discussions of 52 performance bugs

Problem Instance

Many performance bugs (GUI bloat) in Android apps

Evaluation approach: applied PerfChecker to 29 Android applications and collect feedback from developers

Solution

PerfChecker is a tool for static code analysis wrt to most important bug types: - lengthy operations in main thread - violation of DP for list views (the ViewHolder Design Pattern) built on Soot, a widely-used Java program analysis framework [47].

Solution design approach: Study of the bug reports and related discussions of 52 performance bugs

Relevance: A previous inspection of 60,000 Android applications [11] revealed that 11,108 of them have suffered or are suffering from performance bugs. Many smartphone applications are developed by small teams without dedicated quality assurance. These developers lack viable techniques to help analyze the performance of their applications [23].

Rigor: PerfChecker was applied to analyze the latest version of all 29 Android applications, which comprise more than 1.1 million lines of Java code. Selection of OSS projects based on relevant factors, conclusions based on tests of significance.

Novelty: PerfChecker finished analyzing each application within a few minutes and detected 126 matching instances of the two performance bug patterns in 18 of the 29 applications.

Paper Title: Characterizing and Detecting Performance Bugs for Smartphone Application

Paper ID: Murphy-Hill 2014

Technological rule: To improve developer experience and tools during game and other development scenarios apply the recommendations that emerged from this study.

Problem Understanding: Literature review; 14 interviews with game devs; survey with 364 devs to contrast game dev experiences with other developers

Problem Instance

Poor developer experience and tools are identified from 14 game developers and 364 traditional software developers.

Evaluation approach: N/A

Solution

No concrete intervention proposed, but proposed recommendations such as need for testing approaches for rapidly changing requirements. Insights from game developers (e.g., more focus on user needs) may also improve other developer experience

Solution design approach: By analytical reasoning based on observed differences and similarities. Some recommendations build on insights from other areas, specifically non-deterministic program testing.

Relevance: Software developers in general wishing to improve their processes and tools.

Rigor: Triangulation of methods: surveyed a large and diverse group of game developers to validate insights from the literature and interviews. Survey and interview questions are posted online to support future replication.

Novelty: New insights on game developer challenges and practices.

Paper Title: Cowboys, Ankle Sprains, and Keepers of Quality: How is Video Game Development Different from Software Development?

Paper ID: Okur2014

Technological rule: To improve the responsiveness of asynchronous programs, where outdated or misused idioms are adopted, use automated tools to detect and fix these issues.

Problem Understanding: 1378 open source apps (Microsoft CodePlex and GitHub) were analyzed, finding many anti-patterns of misused asynchronous programming.

Problem Instance

Many instances of misuse of asynchronous programming anitpatterns are identified and studied in a corpus of open source apps.

Evaluation approach: Evaluation of the effectiveness and efficiency of tools with the code corpus used to identify the problems. Patches are suggested to the original developers (via a pull request), many of which were accepted.

Solution

Two tools are proposed: 1) a refactoring tool called Asyncifier and 2) a transformation tool called Corrector to fix misuse antipatterns

Solution design approach: A large scale study of WPApps analyzing how older idioms and misuses occur was conducted. Results led to a toolkit to addres those issues.

Relevance: Asynchronous programs that use outdated asynchronous idioms, or misuse modern idioms. The techniques they propose can be applied to other C# platforms. Relevant for developers as well as tool vendors, language and library designers, educators and researchers.

Rigor: An empirical study that quantitatively assesses that the techniques work, and a study to demonstrate applicability of their solution were carried out.

Novelty: 1) large scale empirical study to answer questions about asynchronous progamming and async/wait; 2) development and evalluation of a toolkit (two tools) that implements analysis and transformation algorithms to address challenges

Paper Title: Semih Okur, David L. Hartveld, Danny Dig, and Arie van Deursen, A Study and Toolkit for Asynchronous Programming in C#

Paper ID: Rodeghero2014

Technological rule: To improve automated source code summarization for programmers reading code apply weighting scheme for terms, based on human behavior

Problem Understanding
10 devs wrote summaries of 67 Java methods from 6 different applications; through eye-tracking, researchers analyzed the keyword types they focused on.

Problem Instance

Code comprehension is a general challenge and Programmers prefer to read the least amount of text/code to gain comprehension

Evaluation approach:
Three sets of weights (determined from the eye tracking gaze events) were evaluated in a user study, whereby the resulting list of keywords were then compared to keywords that were selected by human experts.

Solution

Extend existing automated source code summary tools with term weighting, based on observations of human behavior

Solution design approach:
Automated code summaries were created by adding weights to keywords in the VSM tf/idf approach.

Relevance: Summaries of code are useful for programmers to aid comprehension

Rigor: Human experts read Java methods and ranked the top five most-relevant keywords from each method and compared to the automated summarization

Novelty: Weighting factors in automated code summarization, using VSM, based on empirical study of human behavior

Paper Title: Improving Automated Source Code Summarization via an Eye-Tracking Study of Programmers

Paper ID: Lavallee2015

Technological rule: To find factors to take into account by software quality assurance in large scale software projects take specific management actions to mitigate identified organizational problems that may impact software quality

Problem Understanding: 10-month study based on non-participant observation of a software development team's weekly status meetings, validated with manager interviews.

Problem Instance

To understand how organizational factors negatively impact software quality, a study was conducted at a large telecommunications company with over 40 years in business where it was observed that despite good developers, the outcome on software quality was not always positive.

Evaluation approach: N/A. They do not evaluate their recommendations.

Solution

To address the observed flaws they suggest the following corrective actions: a) encourage face-to-face meetings, b) don't manage multiple versions of components, c+d) follow standardized processes for acquisition of third-party components, e) change processes gradually, f+g) manage scope, quality and deadlines, h) balance formal processes and human interaction, i,j) promote knowledge sharing.

Solution design approach: They rely on their experience, insights gained during the study and the literature to arrive at recommendations to mitigate the problems they observed

Relevance: Large-scale, general software quality problems

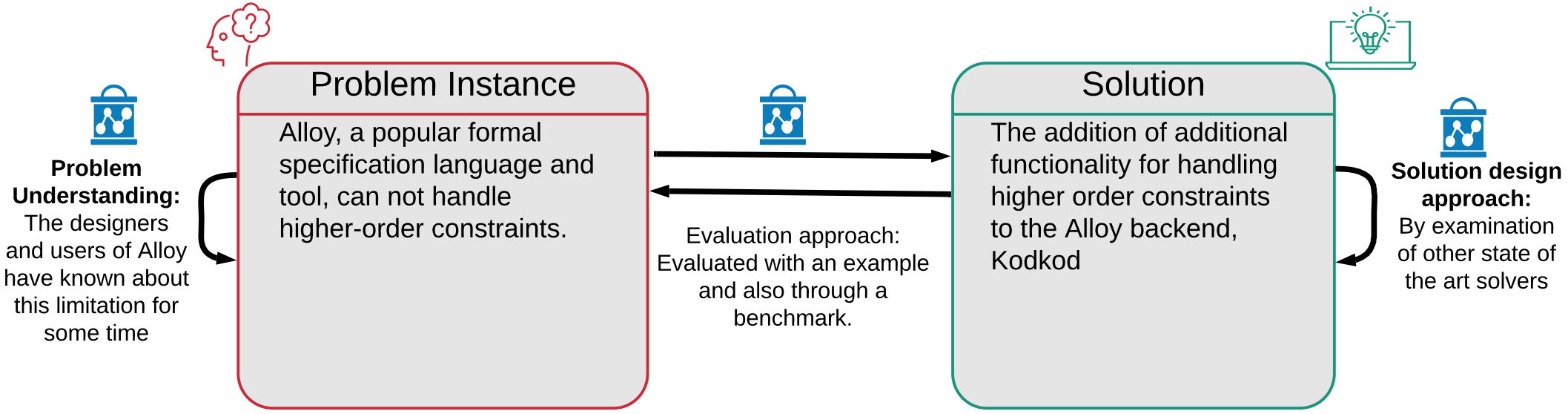
Rigor: Ten months of observation of an in-house software development project within a large telecommunications company to understand the problems.

Novelty: The problem description, based on a year of observations that shows organizational factors may impact software quality

Paper Title: Why Good Developers Write Bad Code: An Observational Case Study of the Impacts of Organizational Factors on Software Quality

Paper ID: Millicevic2015

Technological rule: To handle higher-order quantifiers in a generic and model-agnostic way, when using Alloy use the updated version that supports higher order constraints



Relevance: It is releveant to designers and users of Alloy who needs to handle higher order constraints..

Rigor: The paper validates the approach on a community accepted benchmark of reasonably complex problems. The benchmark process considers three competing tools and one state of the art synthesis tool

Novelty: Alloy* is the first general-purpose constraint solver capable of solving formulas with higher order quantification. Existing solvers either do not admit such quantifiers, or fail to produce a solution in most cases.

Paper Title: "Alloy: A General-Purpose Higher-Order Relational Constraint Solver*

Paper ID: Nistor2015

Technological rule: To fix performance bug problems caused by unnecessary loop execution use CARAMEL, which adds a break within loops

Problem Understanding: Literature references to papers on performance bugs and a classification of CondBreak fixes

Problem Instance

Performance bugs are programming errors that slow down program execution. The study aims to study how likely developers are to fix a performance bug.

Evaluation approach: CARAMEL was evaluated on 11 JAVA and 4 C/C++ applications through two implementations respectively CARAMEL-J and CARAMEL-C

Solution

CARAMEL, a novel static technique that detects and fixes performance bugs that have non-intrusive fixes likely to be adopted by developers. Typically, these bugs are fixed by adding one line of code inside the loop, i.e., if (cond) break, called a CondBreak fix.

Solution design approach: A set of RI (result instructions) have been classified into 6 types. They have been used to implement the CARAMEL algorithm.

Relevance: Programming errors due to performance bugs. Relevant for programmers for which performance is an issue

Rigor: The validation pointed out that CARAMEL, used on 11 JAVA applications, and 4 C/C++ applications identified 61 performance bugs in the JAVA applications and 89 in C/C++ ones. Of these, 10+24 were not found before.

Novelty: (i) detect performance bugs whose fixes clearly offer more benefits than drawbacks to developers; (ii) identify a family of performance bugs; (iii) CARAMEL as tecnique for detecting performance bugs that have CondBreak fixes;

Paper Title: CARAMEL: Detecting and Fixing Performance Problems That Have Non-Intrusive Fixes

Paper ID: Siegmund2015

Technological rule: N/A N/A N/A

Problem Understanding: Conducting a literature review and a survey among 79 program-committee and editorial-board members of 11 major software-engineering venues

Problem Instance

The SE community is lacking consensus on internal and external validity

Evaluation approach: **N/A**

Solution

There is no proposed intervention

Solution design approach: N/A

Relevance: N/A

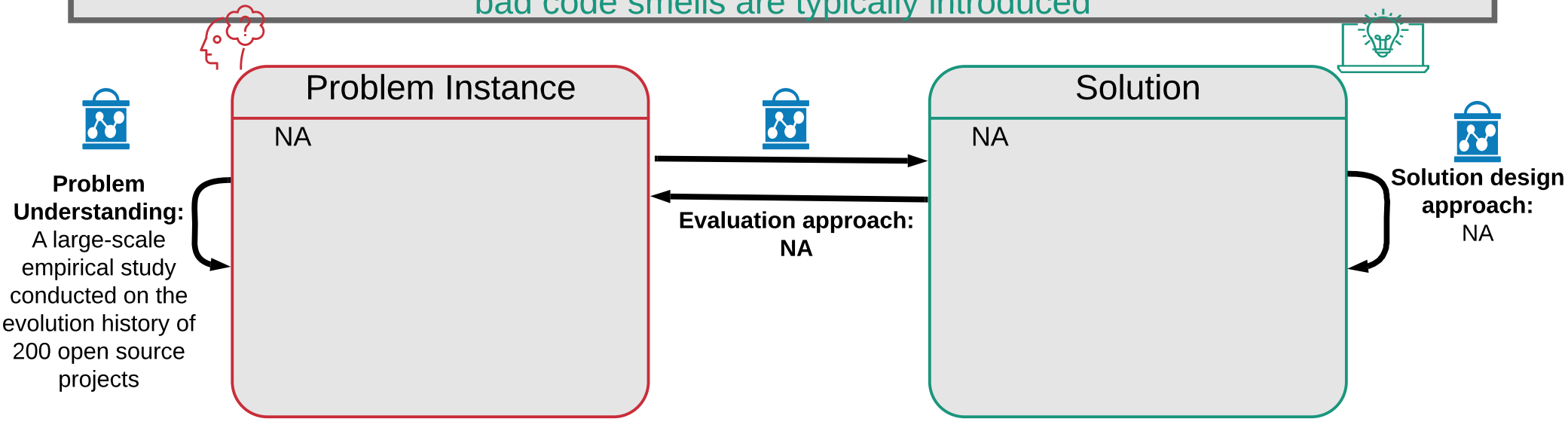
Rigor: SE experts were asked through an online survey.

Novelty: Description of the situation

Paper Title: Views on Internal and External Validity in Empirical Software Engineering: Janet Siegmund, Norbert Siegmund, and Sven Apel"

Paper ID: Tufano2015

Technological rule: To better plan activities for improving design and source code quality when developing software utilise the knowledge gained in this study about when and why bad code smells are typically introduced



Relevance: The study context is the change history of 200 projects belonging to three software ecosystems, namely Android, Apache, and Eclipse

Rigor: Validity is ensured because a large set of 200 projects concerning the analysis of code smells and oftheir evolution has been investigated. Projects were extracted from three ecosystems: android, apache and eclips.

Novelty: First comprehensive empirical investigation into when and why code smells are introduced in software projects.

Paper Title: When and Why Your Code Starts to Smell Bad

Paper ID: Waterman2015

Technological rule: To understand architecture development in agile development settings use a Theory that identified forces in the context of a project that affect agility and their relation to project strategies for agility

Problem Instance

Understanding how much time up front should be spent on architecture planning in agile settings (several instances are studied)

Problem Understanding: Understanding is obtained from literature, with additional insights gained from the grouned theory study.

Evaluation approach: N/A

Solution

Identifies forces and strategies in an inductive manner to address challenges in architecture development in agile projects

Solution design approach: Grounded theory research

Relevance: Researchers and practitioners that are involved in architecture design

Rigor: Use of grounded theory with 44 participants to give insights on the problem instances.

Novelty: The presented theory.

Paper Title: How much up-front? A grounded theory of Agile Architecture

Paper ID: Bersani2016

Technological rule: In order to scale the size of traces when checking a logged event trace against a temporal logic specification use lazy semantics

Problem Understanding: Previous research

Problem Instance

When checking a logged event trace against a temporal logic specification, known algorithms like MTL (Metric Temporal Logic) do not scale with respect the length of the trace and the size of the time interval of the formula to be checked.

Evaluation approach: Integrate lazy semantics and modified distributed trace checking algorithm in the MTLMapREDUCE Tool

Solution

Use Lazy Semantics, a new semantics for MTL, to address memory scalability issues

Solution design approach: 1) Propose a new semantics for MTL 2) provide a parametric decomposition of MTL formulae 3) introduce a new trace checking algorithm; 4) evaluate the proposed algorithm

Relevance: Problems related to tracking large scale log data for events. The technique is relevanto to who inspects server logs, crash reports, and test traces in order to analyse problems at runtime.

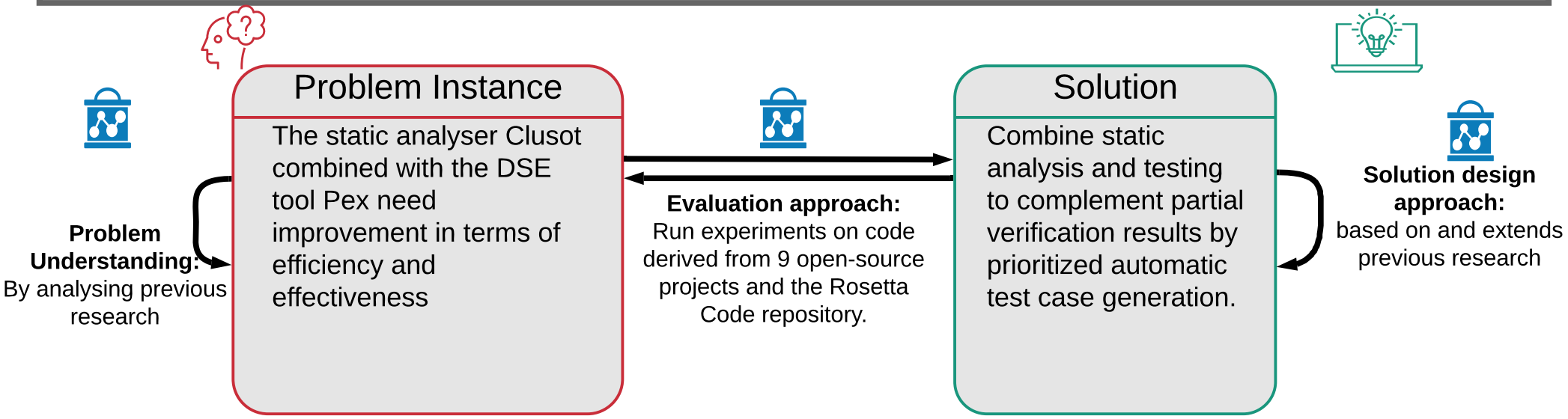
Rigor: Evaluation of properties, using 20 synthesized traces with 50 million elements each

Novelty: Use a new semantics for MTL called Lazy Semantics

Paper Title: Efficient Large-scale Trace Checking Using MapReduce

Paper ID: Christakis2016

Technological rule: To reduce testing time and improve coverage in programs that are parially verified based on unsound assumptions guide DSE toward unverified program executions



Relevance: Relevant for .net developers

Rigor: Experiments used 101 methods written in C# from nine open-source projects and from solutions to 13 programming tasks on the Rosetta Code Repository

Novelty: In contrast to existing work, our technique supports the important case that the verification results are obtained by an unsound (manual or automatic) static analysis.

Paper Title: Guiding Dynamic Symbolic Execution toward Unverified Program Executions

Paper ID: Gousios2016

Technological rule: To improve the pull-based contribution model in open source projects, follow recommendations that may improve the process.

Problem Instance

When using the pull-based model in open source projects, there is a lack of knowledge on the work practices used and challenges faced by contributors. Concrete problem instances are explored through a survey.

Problem Understanding: Survey sent to 4,172 conttributors with questions about contributing in the pull model

Evaluation approach: N/A

Solution

No intervention is proposed for the specific instances however several recommendations are provided: reduce response time, enhance awareness, improve communication and quality assessment of contributions.

Solution design approach: The survey led to the suggested recommendations.

Relevance: Open source projects that accept contributions from the community wishing to improve their process.

Rigor: Carefully designed survey, large sample of contributors, wide range of projects, pilot of survey

Novelty: Identified challenges in pull-based development and recommendations for improving the pull-based process.

Paper Title: Work Practices and Challenges in Pull-Based Development: The Contributor's Perspective

Paper ID: Hasan2016

Technological rule: To improve the energy efficiency of software developed in Java use per-method energy profiles

Problem Instance

Higher than necessary energy consumption was observed in six applications and libraries due to selection of Collections and methods in their implementation.

Problem Understanding: Studied energy consumption profiles for methods used for three commonly used Collections datatypes: List, Map and Set.

Evaluation approach: Combined energy impact of all invoked API methods are measured for 6 real applications to estimate improvement potential.

Solution

Create per method energy profiles of popular Java Collection classes to help developers estimate energy needs and choose a different alternative if necessary.

Solution design approach: They found the solution by closely studying the problem of energy efficiency of Java Collections.

Relevance: Developers wishing to use green programming techniques (in particular to select among methods in Java Collections)

Rigor: They study if what they find applies in other cases by modifying those applications and checking if using the profiles actually helps improve the energy consumption various applications (e.g., Google Gson, XStream and K-9 Mail).

Novelty: A new approach for profiling Java Collections in terms of energy consumption.

Paper Title: Energy Profiles of Java Collections Classes

Paper ID: Madsen2016

Technological rule: To reduce instrumentation overhead for computing crash paths, for Javascript bugs that are difficult to diagnose from error messages and stack traces alone, compute crash paths by distributing the instrumentation overhead to a crowd of users, thereby reducing the impact of instrumentation at run time

Problem Understanding: A case study

Problem Instance

Error messages and stack traces are insufficient to debug real-world specific problematic issues in JavaScript applications reported on Github

Evaluation approach: Evaluated on 10 real-world issues where diagnosis was problematic. Measured instrumentation overhead and number of times a crash needs to be encountered to recover the complete crash path.

Solution

A prorotype implementation CROWDIE was used to debug the issues

Solution design approach: Inspired by similar crowdsourcing techniques for debugging that were pioneered by Liblit et al. and by Orso et al.

Relevance: Error messages and stack traces provide insufficient information for diagnosing problems in deployed Java applications

Rigor: Tool is used to debug 10 real world issues.

Novelty: The notion of crash paths is novel as well as application of crowd-based techniques to compute them.

Paper Title: Feedback-Directed Instrumentation for Deployed Javascript Applications

Paper ID: Menendez2016

Technological rule: In order to detect non-termination bugs with peephole optimizations use ALIVE tool that LLVM developers can use to check non-termination before they commit a new peephole optimization

Problem Understanding: Related work and previous research on compiler optimizations

Problem Instance

Peephole optimizations that represent a persistent sourcee of bugs in mainstream compilers.

Evaluation approach: Experimentation of the prototype termination checker for Alive InstCombine optimizations

Solution

The research proposes a methodology to identify compiler non-termination bugs by extending Alive (a domain specific language for specifying peephole optimizations) to provide a global view of InstCombine optimizations in LLVM

Solution design approach: N/A

Relevance: Relevant for relevant for LLMV developers when addressing non-termination bugs that arise when a suite of peephole optimizations is executed until a fixed point.

Rigor: 184 optimization sequences involving 38 optimizations that cause non-terminating compilation in LLVM with Alive-generated C++ code have been identified in the experimentation.

Novelty: 1) new class of compiler non-termination bugs resulting from lack of a global view of peephole optimizations in LLVM; 2) a methodology to detect compiler non-termination bugs building on top of Alive, which checks the correctness of each individual InstCombine transformation, 3) non-increasing source in the self-composition of a sequence of optimizations as the necessary condition for non-termination, 4) a technique to generate concrete inputs to demonstrate non-termination errors to aid debugging

Paper Title: Termination-Checking for LLVM Peephole Optimizations

Paper ID: Rizzi2016

Technological rule: To increase efficient replication and aggregation of knowledge in software engineering research, developing tools, change academic publication and rewarding structures

Problem Understanding: The conclusions of a sample of papers citing KLEE was investigated after six minor improvements were made to KLEE.

Problem Instance

Research built on KLEE omits the fact that the veracity of the new techniques depends on whether the underlying systems really embody the techniques they implement

Evaluation approach: N/A

Solution

Recommendations to align incentives for tool maintence work: Cite Papers and Tools, Increase Institutional Support, promote Competition and create tool maintenance tracks at conferences.

Solution design approach: By analyzing the papers published on KLEE, and the incentive structure in academia.

Relevance: Demonstrating the value of technique A may involve direct comparison with or building on technique B. The implementations of both A and B will then play a crucial role in the validity of the conclusions

Rigor: Previous studies based on KLEE were replicated after its improvement. Original authors were involved to give feedback.

Novelty: Empirical evidence that the strong emphasis on introducing "new" techniques may lead to wasted effort and questionable research conclusions.

Paper Title: On the Techniques We Create, the Tools We Build, and Their Misalignments: a Study of KLEE

Paper ID: Ye2016

Technological rule: To improve information retrieval in SE where there is a lexical gap between text documents and code use word embeddings based on the distributional hypothesis

Problem Understanding: Literature proves that existing baseline methods do not effectively link text and code when there is a lexical gap in a bug localization benchmark.

Problem Instance

Bug localization tasks in an Eclipse corpus of code as well as linking questions to code snippets in StackOverflow.

Evaluation approach: Solutions has been evaluated on the Eclipse bug localization benchmark dataset.

Solution

Authors propose to use word embeddings based on the distributional hypothesis to improve API recommendations during tasks such as bug localization, and recommendations for APIs in StackOverflow.

Solution design approach: Proposed solution is inspired by the distribution hypothesis (in NL processing) that co-occurrences of words (whether NL or code) may be related.

Relevance: Relevant for tool designers, in situations where there is a lexical gap between search queries (usually expressed in textual form) and retrieved files (normally source code files).

Rigor: train word embeddings on API documents, tutorials and reference documents, then aggregate in order to estimate semantic similiarties between documents

Novelty: Word embeddings have been applied in various NLP tasks, but not for linking code files and textual queries in SE.

Paper Title: From word embeddings to document similarities for improved information retrieval in software engineering

Paper ID: Yu2016

Technological rule: To improve test effectiveness in testing of virtual devices in full system simulators (FSS) use VDTest to obtain test cases and carry out testing

Problem Understanding: Referring to several studies that have shown that software faults in virtual devices can cause security vulnerabilities [2–4]

Problem Instance

Testing of virtual devises, i.e., Full System Simulators, is challenging

Evaluation approach: In an empirical study, comparing VDtest to two other test approaches on 11 devices using both open source and commercial FSSs, we found 64 faults, 83% more than random testing.

Solution

VDTest: an approach for testing FSS's including test case generation and testing

Solution design approach: The design is presented without trace of how it was conducted.

Relevance: Testing of virtual devices

Rigor: Empirical evaluation of VDtest to virtual devices from three FSSs that are widely used in both industry and academia

Novelty: The design of VDtest.

Paper Title: VDTest: An Automated Framework to Support Testing for Virtual Devices

Paper ID: Floyd 2017

Technological rule: To understand how human brain processes software engineering tasks when making subjective human judgments use medical imaging techniques (fMRI - functional magnetic resonance imaging)

Problem Understanding: Controlled experiment monitored with fMRI technique to gain understanding.

Problem Instance

Code comprehension, code review and review using functional magnetic resonance imaging

Evaluation approach: N/A

Solution

Use medical imaging techniques to understand code comprehension and code review and the relationship to natural language and expertise.

Solution design approach: N/A

Relevance: Code review and comprehension vs prose review activities

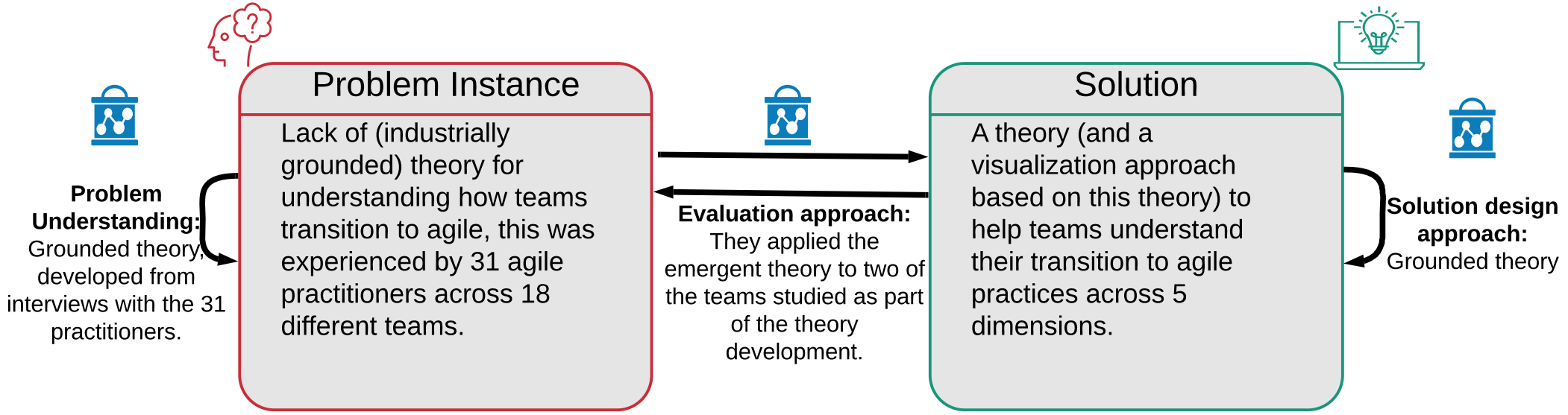
Rigor: In a controlled experiment involving 29 participants, authors examine code comprehension, code review and prose review using functional magnetic resonance imaging.

Novelty: Measure brain activity for carrying out different tasks

Paper Title: Decoding the representation of code in the brain: An fMRI study of code review and expertise

Paper ID: Hoda2017

Technological rule: To gain more understanding of a team's work practices, when transitioning to agile, use the 5 dimensions proposed in this theory.



Relevance: Industrial teams wishing to understand and improve their transition to agile

Rigor: Constant comparisons and theoretical coding were used as part of the grounded theory approach.

Novelty: A theory of 5 dimensions (culture, software practices, team practices, management approach & reflective practices) to model how a team is transitioning to agile

Paper Title: Becoming Agile: A Grounded Theory of Agile Transitions in Practice

Paper ID: Landman2017

Technological rule: To improve or enable static analysis of Java projects, when reflective code is used, follow the guidelines provided in this paper.

Problem Instance

In a constructed corpus of diverse 461 open source Java projects, 78% of the projects contain code (reflections) challenging state-of-art static analysers.

Problem Understanding: Measured how widespread this problem occurs in a corpus of code.

Evaluation approach: N/A: They do not apply the tactics to the problem instance they study.

Solution

Java devs: a) factor out reusable reflective code in type polymorphic methods, b) avoid use of dynamic proxies, c) use local variables or fields to store references to meta objects over collections, d) avoid loops over bounded collections of meta objects; e) test for preconditions rather than using exceptions.

Solution design approach: From an analysis of corpus and existing approaches and their limitations.

Relevance: For Java software engineers prioritizing on robustness and for static analysis tool builders wishing to improve their tools.

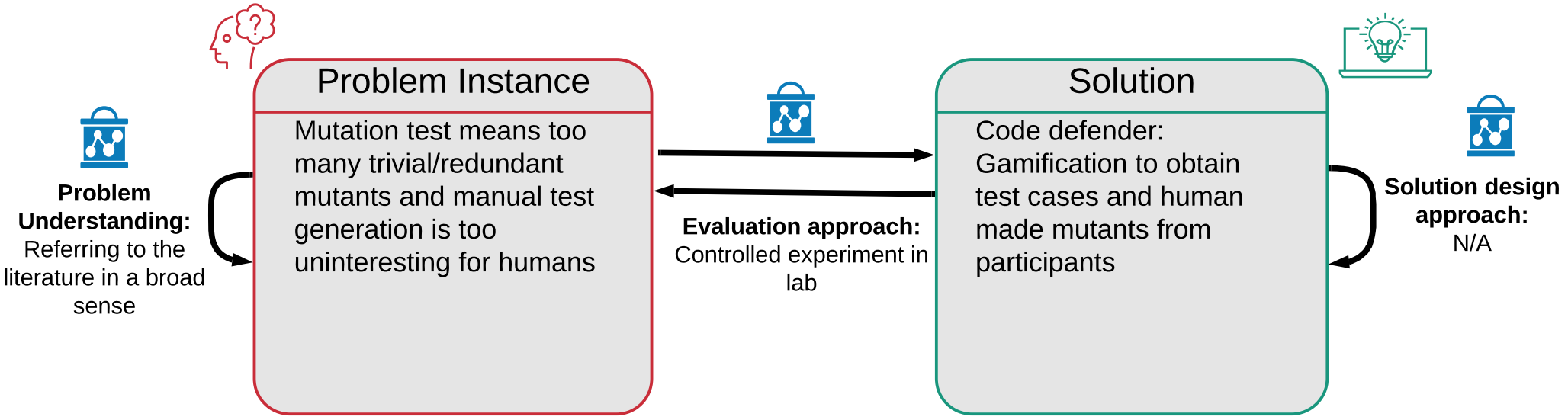
Rigor: A litterature review of existing approaches for static analysis and a survey of real world projects' use of the Reflection API

Novelty: New strategies for dealing with analysis/authoring of reflective code

Paper Title: Challenges for Static Analysis of Java Reflection – Literature Review and Empirical Study

Paper ID: Rojas2017

Technological rule: To increase efficient replication and aggregation of knowledge in software engineering research, developing tools, change academic publication and rewarding structures



Relevance: A very general class of issues with respect to mutation testing and automated test generation

Rigor: The evaluation is conducted in an artificially created instance, where rigirous experimentaiton procedures are applied.

Novelty: Code Defender, the idea of using gamification in order to obtain better results in carrying out the otherwise rather dull tasks of formulation mutants and test cases.

Paper Title: Code Defenders: Crowdsourcing Effective Tests and Subtle Mutants with a Mutation Testing Game

Paper ID: Shi2017

Technological rule: To reduce wasteful text executions, when tests are poorly placed in modules, use historical build information & test dependencies to guide which tests should be run.

Problem Understanding
The authors mathematically formalize the problem of wasteful test executions that occur when tests are poorly placed in project modules.

Problem Instance

Many large software projects have imprecise dependency graphs that lead to wasteful test executions which impacts on developer productivity

Evaluation approach:
TestOptimizer was applied to 5 large proprietary projects, calculating time saved by potential movements. Validation was carried out with developers who accepted 84% of the suggestions.

Solution

TestOptimizer, a technique which uses a greedy algorithm to suggest test movements between test nodes.providing a ranked list of suggestions to devs allowing them to choose and reduce the expected number of test executions.

Solution design approach:
A greedy algorithm is proposed to reduce # of test executions by suggesting test movements that consider historical build information & test dependencies

Relevance: Relevant for developers during regression testing of large systems when tests may be poorly positioned, leading to wasteful test executions at build time.

Rigor: Case study with 5 large systems was carried out.

Novelty: Proposal of TestOptimizer, a tool for integration in a build environment for making suggestions on how to reduce wasteful test executions. The tool can also be used to reduce impact of flaky tests.

Paper Title: Optimizing Test Placement for module level regression testing, A.Shi, S.Lahiri, N.Bjorner, J.Czerwonka

Paper ID: Tsantalis2017

Technological rule: To enable the refactoring of Type-2 and Type-3 clones having behavioral differences that cannot be parameterized with regular parameters when refactoring Java programs use Lambda expressions

Problem Understanding: A previous study on clones detected by four different clone detectors

Problem Instance

Created a large and diverse dataset of clones extracted from 9 open osurce projects. 46,765 clone pairs having statement gaps (Type-3) and/or expression differences (Type-2) that were assessed as non-refactorable in previous work.

Evaluation approach: Correctnes and appicability analysed on a data set of 50k clones extracted from 9 OSS projects.

Solution

in Java, merge everything that is common into one chunk of code and describe differences between clones in anonymous lambda-functions, when refactoring Type-2 and Type-3 clones

Solution design approach: Not reported

Relevance: Java-program with type 2 and type 3 clones

Rigor: By applying it and evaluating its effect

Novelty: A technique and tool that utilizes Lambda expressions to enable the refactoring of Type-2 and Type-3 clones

Paper Title: Clone Refactoring with Lambda Expressions

Paper ID: Fan2018

Technological rule: To identify and fix framework exceptions in a more timely manner in android apps use an enhanced dynamic testing tool to detect bugs and a framework exception localization tool to explain the root cause of framework exceptions.

Problem Understanding: Studied 16,245 exception traces from 2,486 Android apps to understand how to debug and fix the exceptions.

Problem Instance

Lack of understanding and tools to understand root causes of why android apps crash.

Evaluation approach: They discover and fix specific previously unknown crashes in Gmail and Google+.

Solution

A taxonomy for characterizing framework exceptions; and improved tools for bug detection and localization.

Solution design approach: Examined the source code, fixes, automated bug reports and Stackoverflow posts of 270K issues and exmained how existing techniques work on these issues.

Relevance: Studied a large corpus of open source Android apps to show framework exceptions account for most crashes, and applied their prototype solution to 27 previously unknow bugs, localizing the root causes in those bugs.

Rigor: Used their tools for revealing and speeding up the fix of of previously unknown bugs in real world apps.

Novelty: Show extent of framework exceptions in Android Apps, contributing a dataset and two prototype tools for revealing and localizing such exceptions.

Paper Title: Large-Scale Analysis of Framework-Specific Exceptions in Android Apps

Paper ID: Loncaric2018

Technological rule: To synthesize data structures that track subsets and aggregations of multiple related collections alternate steps of query synthesis and incrementalization

Problem Understanding: Observing that others tried to solve the problem

Problem Instance

Four example cases are described, implicitly pointing to a problem

Evaluation approach: Cozy's effect on development time, correctness, and efficiency was evaluated for the four real cases

Solution

A tool Cozy, was implemented that can synthesize implementations for complex multi-collection data structures from high-level specifications. It iteratively improves the data structure specification using two cooperating components: a query synthesizer that selects a better representation and implementation for each query method and an incrementalization step that ensures the new representation is kept up-to-date when an update method is called.

Solution design approach: Not reported - focus on the final solution

Relevance: Data structure problems, especially in domains like user interfaces or web services where software must manage some internal state and also handle asynchronous events.

Rigor: Proof of concept demonstrated in four real cases

Novelty: It is a new technique for data structure synthesis that overcomes many of the limitations of previous work

Paper Title: Generalized Data Structure Synthesis

Paper ID: Rath2018

Technological rule: To improve the traceability of commits to issues due to missing links during and after the commit process use the Link Classifer

Problem Understanding: Related work and analysis of six open source projects from Git and Jira that point out lack of traceability between commits and issue IDs

Problem Instance

Many projects suffer from a lack of traceability between commits and issues. Several open source projects are studied that exhibit this challenge.

Evaluation approach: Applied their classifier to 6 open source projects and evalute it in 2 scenarios: 1) a recommender to suggest which issues relate to a commit; 2) augment existing traceability links between commits and issues

Solution

A trained classifier for identifying links between commits and issues, based on commit activity and textual information in source code, commit messages and issues.

Solution design approach: Develop a model of relations between source code, commits and issues, and consider different information retrieval techniques and classifiers to find associations between source code, commit messages and issues

Relevance: They analyzed six OSS showing that an average of only about 60% of commits were linked to specific issues.

Rigor: The proposed solution is applied to the projects to show that it can find links that already exist as well as recommend non existing links that were manually verified.

Novelty: The approach proposed - Link Classifier - that trains a classifier to recommend links when commits are made and augments an existing set of commits and issues with automatically identified links.

Paper Title: Traceability in the wild: automatically augmenting incomplete trace links.

Paper ID: Ren2018

Technological rule: To effectively localize the problematic files for unreproducible builds use the automated framework to analyse differences between binaries built in different build environments

Problem Understanding: Analytic reasoning based on Debian's documentation and bug tracking system

Problem Instance

In Debian's repository there are 2,342 packages that are not reproducible. The localization of problematic files are challenging due to an "information barrier" (of the diff-log) and due to a "diverse course-barrier"

Evaluation approach: RepLoc used to analyse 671 real-world unreproducible Debian packages in four different categories

Solution

Implemented a tool (RepLoc)

Solution design approach: By carefully investigating the localization challenges that are due to issues described in the Debian documentation and by applying the Proposed IR approach

Relevance: Relevant in projects where localisation of problematic files is currently inefficient

Rigor: Implemented a prototype and conducted extensive experiments over 671 real-world unreproducible Debian packages in 4 different categories. RepLoc achieves accuracy rate of 47% (only accounting for the top most rated files). If they expand the examination to top 10 ranked files in the list, the accuracy rate becomes 79%

Novelty: The first work to address the localization task for unreproducible builds

Paper Title: Automated Localization for Unreproducible Builds

Paper ID: Sousa2018

Technological rule: In order to analyze symptoms that manifest in code elements during source code analysis use the grounded theory that explains how developers identify design problems in source code

Problem Understanding: The literature assumes that developers only refer to one symptom during identification of design problems.

Problem Instance

There is a lack of knowledge into how developers recognize design problems in software especially related to source code

Evaluation approach: N/A

Solution

A theory is proposed to help bring understanding on how developers could use different symptoms (if this information were available) to identify design problems.

Solution design approach: A quasi-experiment with developers from 5 companies was carried out to understand how they would use it to make decisions about design problems if provided with certain kinds of information

Relevance: Field study with developers from 5 companies that may encounter design problems.

Rigor: Quasi experiment with professionals from 5 software companies to build a grounded theory that explains how developers identify design problems in practice

Novelty: Grounded theory of how developers diagnose design problems

Paper Title: Identifying design problems in the source code

Paper ID: vanTonder2018

Technological rule: To automatically repair resource leaks, memory leaks and null dereferences in source code that lacks test cases or developer annotations use separation logic to generate patches based on code fragments from the program under repair

Problem Understanding: They refer to specific worked examples and to the literature

Problem Instance

Automated program repair tools do not work well for bugs that either do not have test cases, or are difficult to test deterministically (due to memory leaks, resource leaks and null dereferences)

Evaluation approach: Evaluate their tool FootPatch on 8 C programs and 3 Java programs averaging 64 kLOC

Solution

A static analysis and verification tool (based on separation logic) that finds and repairs bugs (through source code patches)

Solution design approach: Through formalizing the notion of repair with respect to heap based property violations, they formalize the search procedure to discover candidate patch code from existing program fragments

Relevance: They apply their tool (FootPatch) by correctly fixing 55 bugs (11 previously unknown) in 11 selected multi-language projects in an efficient manner

Rigor: Evaluation on real software projects

Novelty: A new static automatic program repair technique based on Separation logic that generates patches for certain types of bugs (resource leaks, memory leaks and null deferences) that does not rely on preexisting test cases or developer annotations

Paper Title: Static Automated Program Repair for Heap Properties

Paper ID: Wang2018

Technological rule: In order to face the problem of testing strategies when conducting conclolic testing adopt the Optimal Strategy and use the Greedy Algorithm for identifying the optimal policy with low complexity

Problem Understanding: Identify existing heuristics from literature

Problem Instance

Concolic execution provides a combination of symbilic execution and concrete testing. However, what is the optimal concolic testing strategy is still an open problem

Evaluation approach: Conduct an evaluation of the Greedy Algorithm by using the approach on programs in GNU Scientific Library

Solution

They develop a framework which helps to define and compute the optimal concolic testing strategy based on Markov decision processes

Solution design approach: Authors conduct experiments to compare the performance of the optimal strategy vs. heuristic-based ones, to investigate whether existing heuristics are reasonably effective

Relevance: Results of experiments point out that all existing heuristics have significantly higher costs than the optimal cost

Rigor: Experiments that compare the performance of the optimal strategy vs heuristics based ones

Novelty: Proposal of a framework to derive optimal concolic testing strategies, and analyze existing heuristics; proposal of a Greedy algorithm to approximate the optimal strategy.

Paper Title: Towards Optimal Concolic Testing

Paper ID: Yan2018

Technological rule: To mitigate for Zero-day Use-After-Free (UAF) vulnerabilities In standards compliant C-programs apply pointer-analysis-based static analysis

Problem Understanding: Problem reported by National Vulnerability Database http://nvd.nist.gov. Further understanding is gained through analysis of pointers' allocation and de-allocation in C/C++

Problem Instance

UAF vulnerabilities: 138 known in the C test cases in Juliet Test Suite (JTS) and at least 85 previously unknown in10 widely-used open-source C applications, totaling over 3 MLOC

Evaluation approach: Implemented CRed in LLVM-3.8.0, comparing it with 4 state-of-the-art static tools, using all C test cases in Juliet Test Suite and 10 open-source C applications

Solution

They introduce a new pointer-analysis-based static analysis, CRed, for finding UAF bugs in multi-MLOC C source code efficiently and effectively

Solution design approach: Formulated the problem of detecting UAF bugs statically, then investigated challenges faced with existing static techniques for analyzing C/C++ source code

Relevance: Zero-day Use-After-Free (UAF) vulnerabilities are increasingly popular and highly dangerous, but few mitigations exist

Rigor: Investigation of four alterative solutions was done

Novelty: Three contributions: (i) a spatio-temporal context reduction technique, (ii) a multi-stage analysis for filtering out false alarms efficiently, and (iii) a path-sensitive demand-driven approach for finding the points-to information required

Paper Title: Spatio-Temporal Context Reduction: A Pointer-Analysis-Based