Selecting a research topic: Reflection and Lessons from My Research Journey

Shin Hwei Tan

Southern University of Science and Technology

Shin Hwei Tan

Website: https://www.shinhwei.com/

Email: tansh3@sustc.edu.cn

Research Interest:

- Automated Program Repair
- Software Testing
- Mobile Analysis
- Comment Analysis
- Search-Based Software Engineering

University of Illinois at Urbana-Champaign (B.S. & M.S)

National University of Singapore (PhD)

Southern University of Science and Technology (from June 2018)

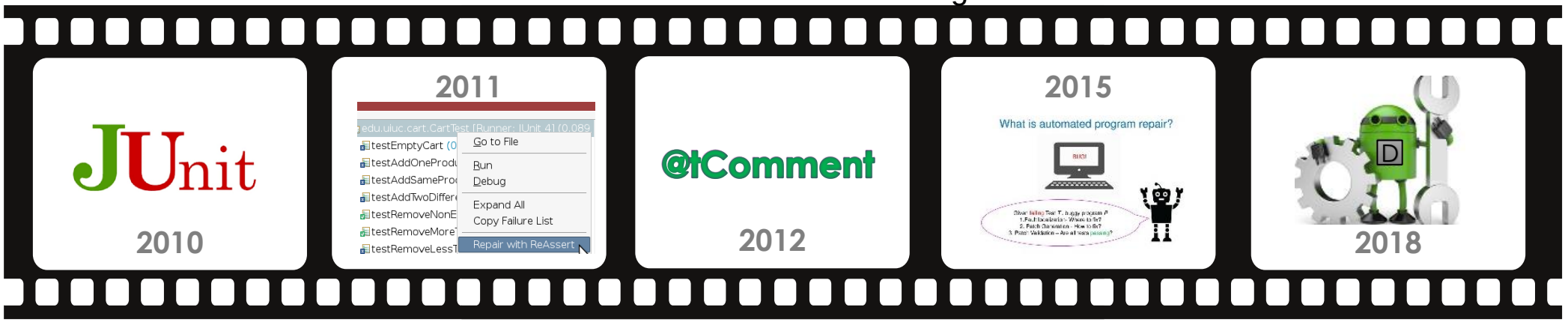
My Research Journey

Repair broken assertions

Research Topic:
Automated repair for broken assertions

Automated Program Repair

Research Topic:
Repairing software regressions



Unit Testing

Research Topic:
Theories/Parameterized tests for Junit.

Comment-Code Inconsistency

Research Topic:
Testing Comment-Code Inconsistencies

Repair Android Apps

Research Topic:
Repairing crashes in Android apps

What is your research passion?

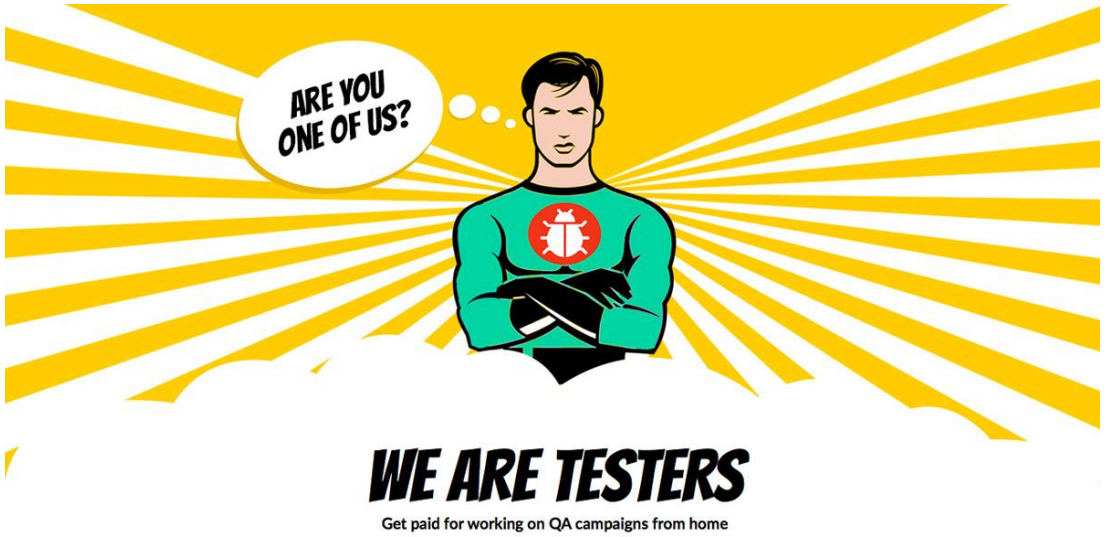
ICSE 2019: (Basic) Analysis of Accepted Papers

- Software Testing
- Program Analysis
- Android
- Data Mining
- Deep Learning
- Fuzzing
- Repair

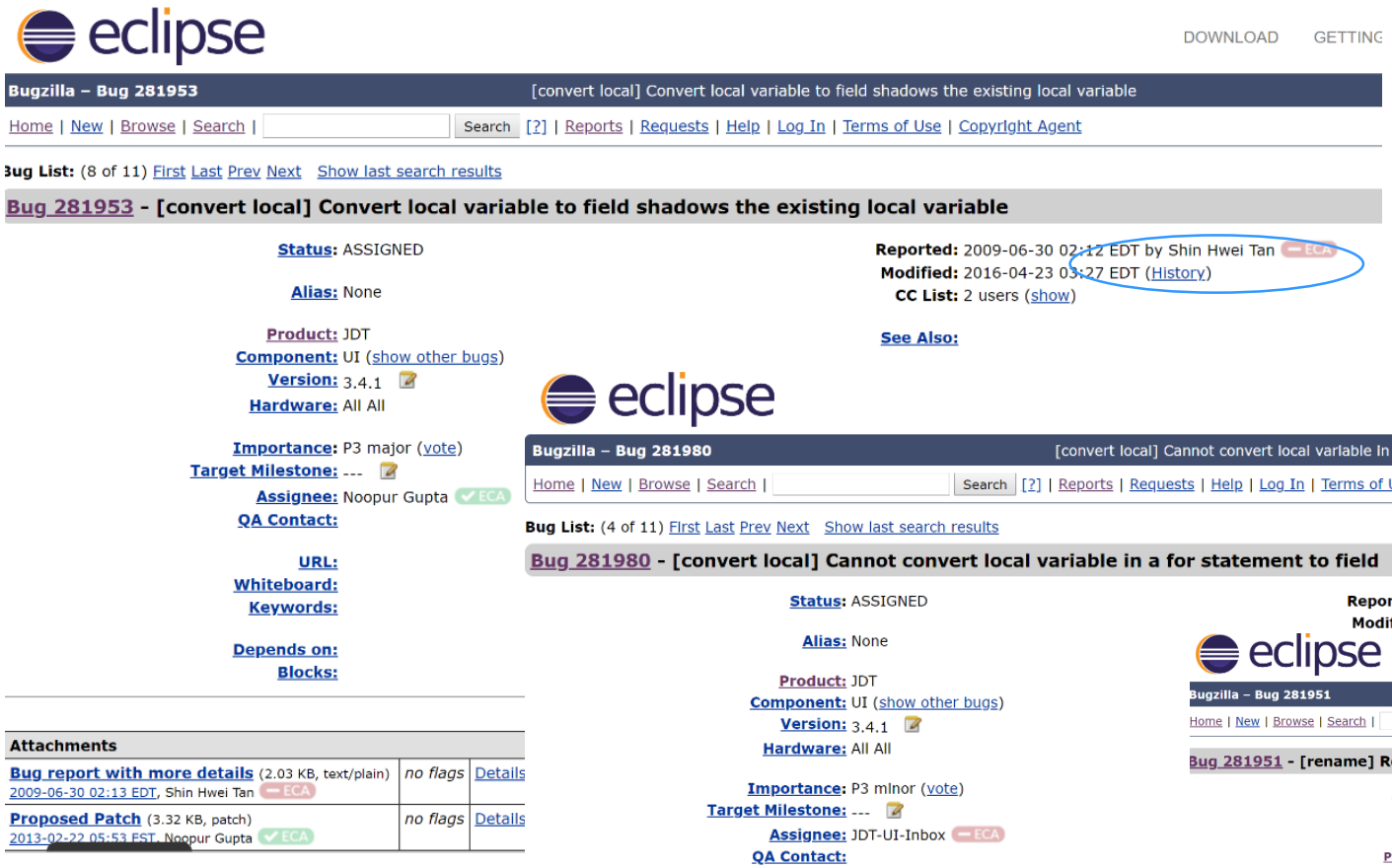
My passion

Cool Hackers

Software Testers



Why software testing?



It all started with some bug reports…

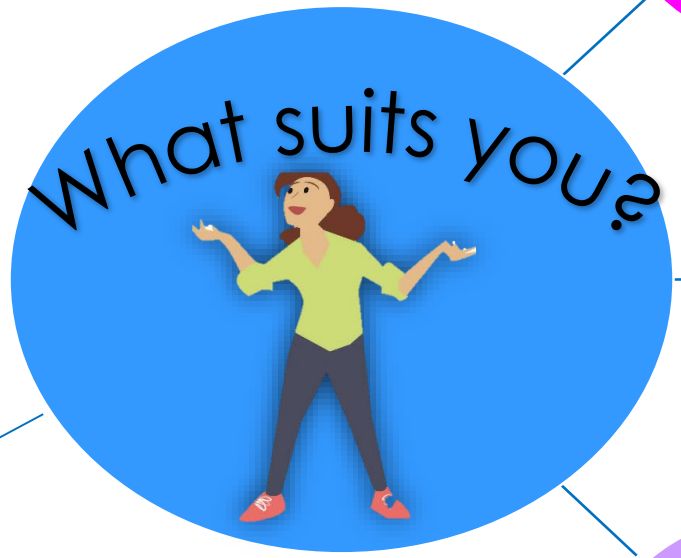
First step into the open-source community…

Passion is the first step, what's next?

How to select a topic?

Let's have a personality test!

Let's have a personality test!



D

- Do you like reading comments and discussion in forums?

R

- Do you like reading research papers?

T

- Do you like trying different techniques and tools?

P

- Do you like writing code for big project?

D

- Do you like reading comments and discussion in forums?

R

- Do you like reading research papers?

T

- Do you like trying different techniques and tools?

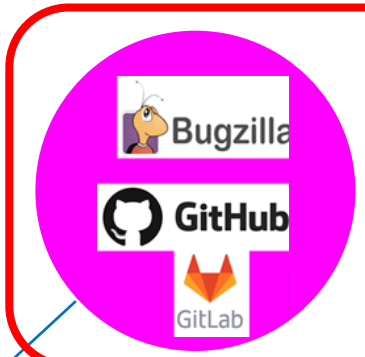
P

- Do you like writing code for big project?

- D for reading discussion
- R for reading papers
- P for coding projects
- T for trying tools

- D+R?
- P+T?
- R+P?

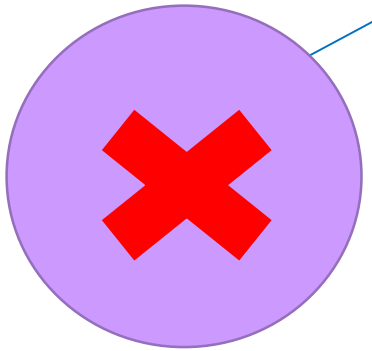
Finding Research Idea



- Read issues reported by developers in open-source project

- Read papers and answer the questions:
  - What is the potential future improvement for this work?

- Identify frequently encountered problems during software development



- Identify the weakness of existing tools

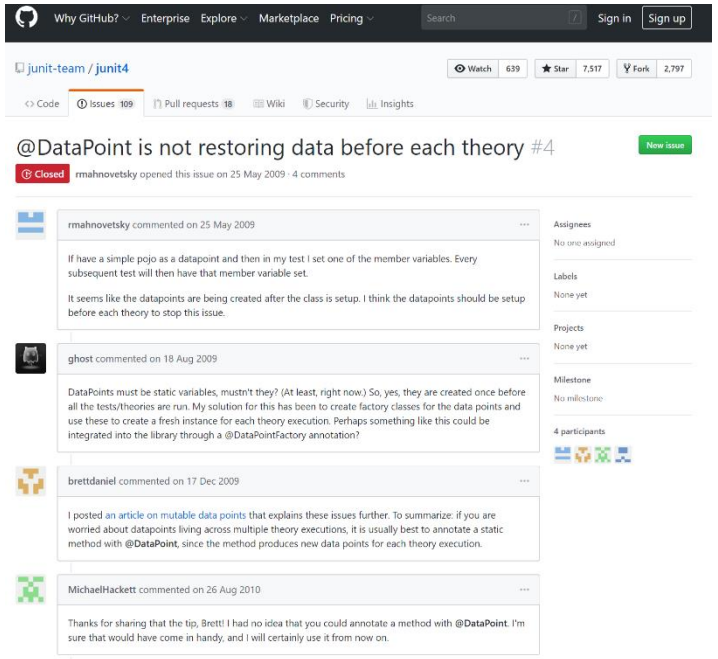
Bugzilla

GitHub

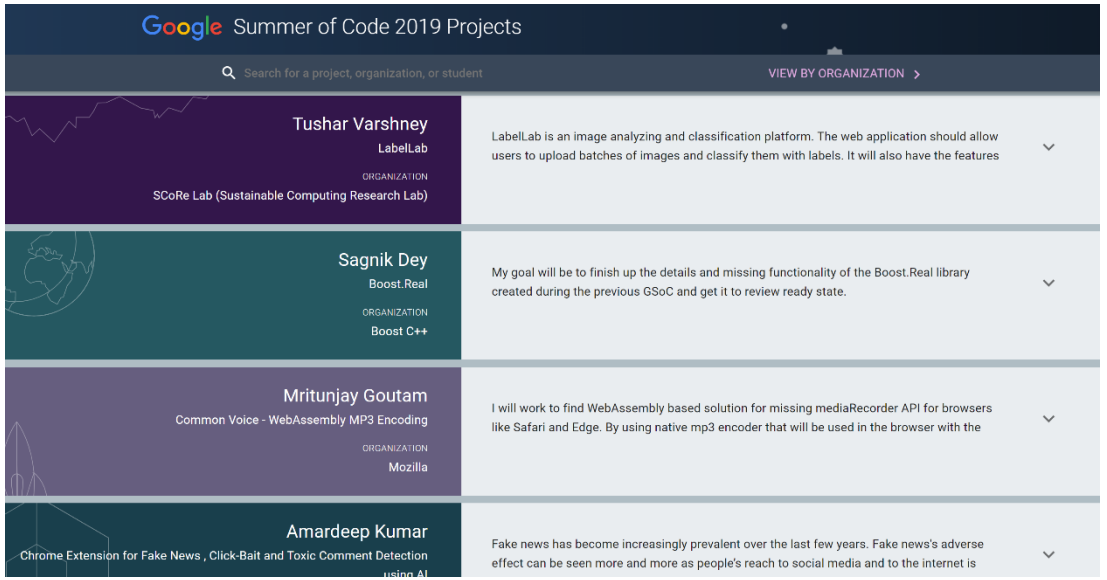
GitLab

Listening to the voices of developers
How to find a concrete problem?

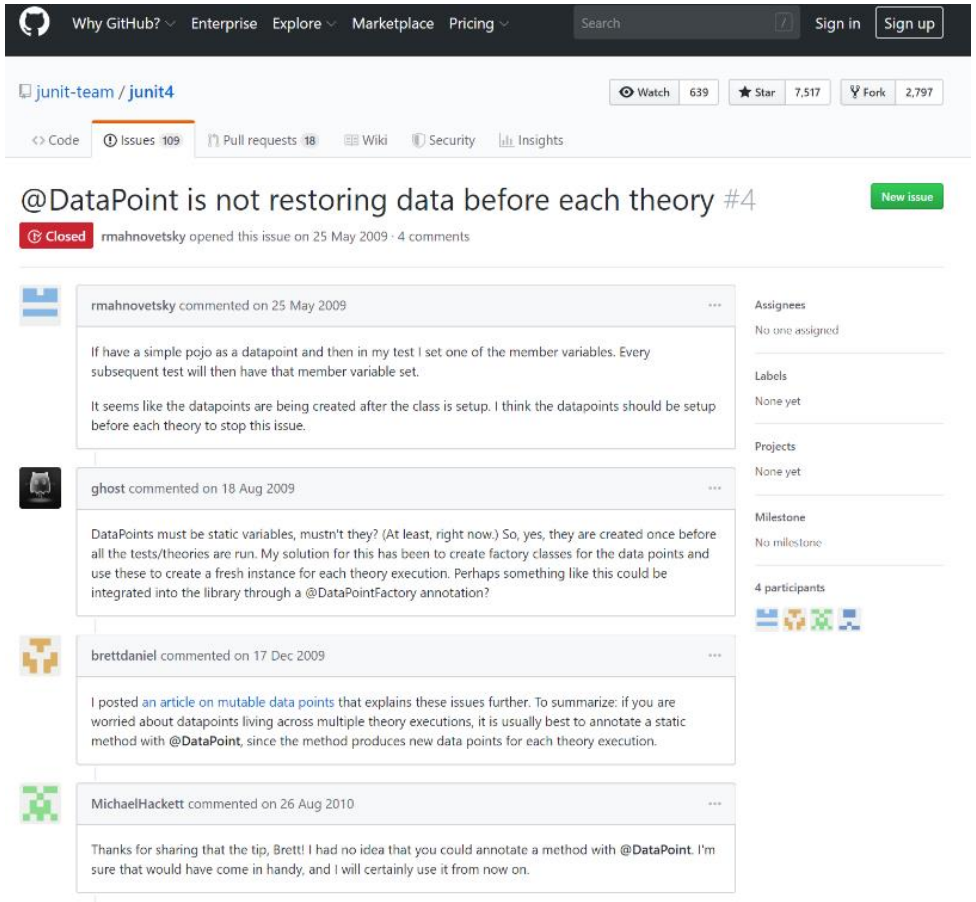
GitHub Issues

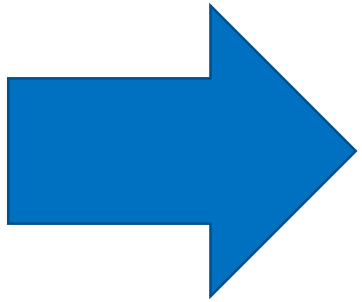


Google Summer of Code Project Description



Project:Theories/Parameterized tests for JUnit Starting from GitHub Issues





Project: Theories/Parameterized tests for JUnit

Communicating with developers

http://docs.yahoo.com/info/terms/

Tan Shin Hwei 9 years ago

David, Berin, Mike and Brett,

Thanks for all the helpful comments.

Regarding the default CloneStrategy, Sang and I started with the following strategies:
1. Strategy for Collection Class
2. Using the clone() method for DataPoint(s) that implements Cloneable
3. Using Interfaces of DataPoint(s)
4. Using the copy constructor of the DataPoint(s) class
5. Using the copy constructor of the super class of the DataPoint.

Below is the method that contains the CloneableStrategy:

```
public Object copyDataPoint(Object toBeReplicated) throws Exception {
try {
Method method= toBeReplicated.getClass().getMethod("clone",
new Class[0]);
//invoke the clone method
return method.invoke(toBeReplicated, new Object[0]);
} catch (Exception e) {
throw new CopyStrategyFailureException();
}
}
```

When we implemented the above default strategies, we had a hard time in deciding which strategy should be used first. I think the same problem will occur if both copy constructor and cloning are available. In that case, I think the user should be able to chose the desired strategy.

Berin mentioned the performance issue within the method getCopyStrategyInvokedObject. I agree that creating a new instance for every object will have a significant impact on performance. I will discuss it with Sang and try to fix the problem.

To: ***@yahoogroups.com; ***@yahoogroups.com
From: ***@gd-ais.com

22 Replies
2 Views
Switch to linear view
Disable enhanced parsing
Permalink to this page

Thread Navigation

Loritsch, Berin C. 9 years ago
David Saff 9 years ago
Loritsch, Berin C. 9 years ago
Brett Daniel 9 years ago
Berin 9 years ago
Brett Daniel 9 years ago
Mike Forsberg 9 years ago
Loritsch, Berin C. 9 years ago
Loritsch, Berin C. 9 years ago
David Saff 9 years ago
Brett Daniel 9 years ago
Loritsch, Berin C. 9 years ago
Tan Shin Hwei 9 years ago
Cédric Beust ♔ 9 years ago
Loritsch, Berin C. 9 years ago
Brett Daniel 9 years ago
Loritsch, Berin C. 9 years ago
David Saff 9 years ago
David Saff 9 years ago
David Saff 9 years ago
David Saff 9 years ago
David Saff 9 years ago
Loritsch, Berin C. 9 years ago

Creating Pull Request

stan6 / junit
forked from junit-team/junit4

Watch 0 | Star 1 | Fork 2,800

Code | Pull requests 0 | Projects 0 | Security | Insights

Immutable DataPoint(s) extension

master

Browse files

Shin Hwei Tan committed on 13 Mar 2010 — 1 parent 1bbab71 — commit dcfa3c41446c469a4edc7513e9e7a9b610b04020

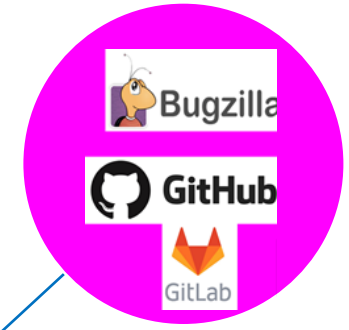
Showing 16 changed files with 672 additions and 13 deletions.

Unified | Split

25 src/main/java/org/junit/experimental/theories/CopyStrategy.java

```
@@ -0,0 +1,25 @@
+ package org.junit.experimental.theories;
+
+
+ /**
+  * CopyStrategy is an interface provided to guarantee immutability among DataPoint.
+  * Any class that implements this interface can invoke copy constructor on the
+  * passed in DataPoint object.
+  *
+  * @author      <a href="mailto:sybaik2@illinois.edu">Sang Yong Baik</a>
+  * @author      <a href="mailto:stan6@illinois.edu">Shin Hwei Tan</a>
+  * @since       JUnit 4.8b3
+  */
+ public interface CopyStrategy {
+
+         /**
+          * Replicates the passed in DataPoint object by user specific means.
```

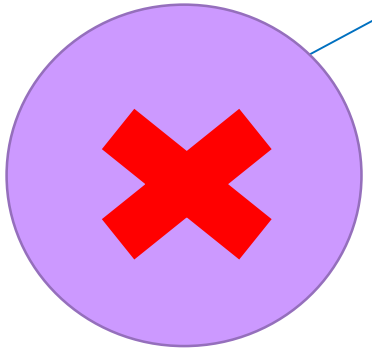
Finding Research Idea



- Read issues reported by developers in open-source project

- Read papers and answer the questions:
  - What is the potential future improvement for this work?

- Identify frequently encountered problems during software development



- Identify the weakness of existing tools

Reading paper is fun but it could be dangerous!

- You could be reading passively without thinking!
- You could end up being depressed thinking that all great researches have been conducted by someonelse!

Reading Papers

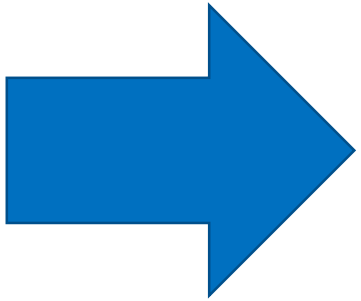
When reading papers, answer the following questions for each:

1) Is there any technical contribution (e.g., new algorithm) of the paper? If yes, what is the technical contribution?
2) What is the main novelty of the paper? Does it study a new domain or does it improve on existing solutions?
3) What are the challenges/problems that the paper tries to solve?
4) What are the good things ("Pros") about the paper? Gives 3 pros of the paper.
5) What are the bad things/ things to improve ("Cons") of the paper? Gives 3 cons of the paper.
6) Could you think about any possible future works that are not listed? Gives 3 future possible improvement for the paper.

Project:@tComment Starting from Paper

/* iComment: Bugs or Bad Comments? */

Lin Tan[†], Ding Yuan[†], Gopal Krishna[†], and Yuanyuan Zhou[†‡]
[†]University of Illinois at Urbana-Champaign, Urbana, Illinois, USA
[‡]CleanMake Co., Urbana, Illinois, USA
{lintan2, dyuan3, gkrishn2, yyzhou}@cs.uiuc.edu



ABSTRACT

Commenting source code has long been a common practice in software development. Compared to source code, comments are more *direct*, *descriptive* and *easy-to-understand*. Comments and source code provide relatively redundant and independent information regarding a program's semantic behavior. As software evolves, they can easily grow out-of-sync, indicating two problems: (1) bugs - the source code does not follow the assumptions and requirements specified by correct program comments; (2) bad comments - comments that are inconsistent with correct code, which can confuse and mislead programmers to introduce bugs in subsequent versions

Keywords

comment analysis, natural language processing for software engineering, programming rules, and static analysis

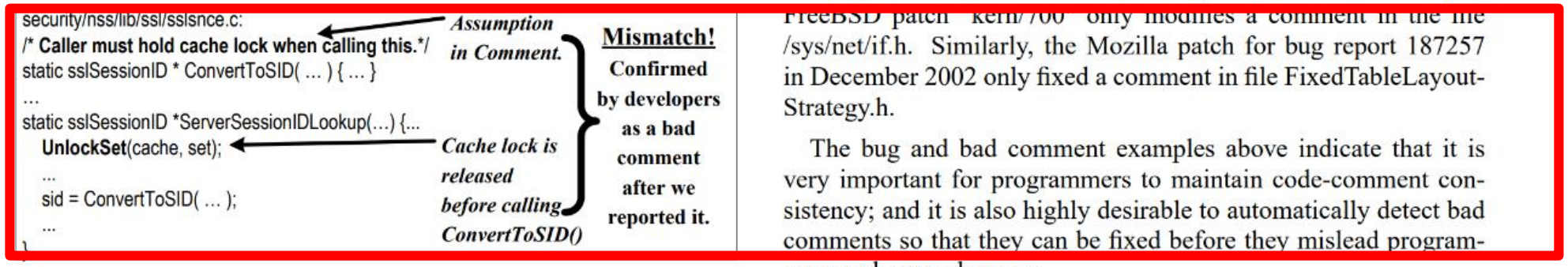
1. INTRODUCTION

1.1 Motivation

Despite costly efforts to improve software-development methodologies, software bugs in deployed code continue to thrive and contribute to a significant percentage of system failures and security

Project:@tComment
What the problem and solution?

Problem: Inconsistent Code and Comment
Solution: Static analysis to detect inconsistencies



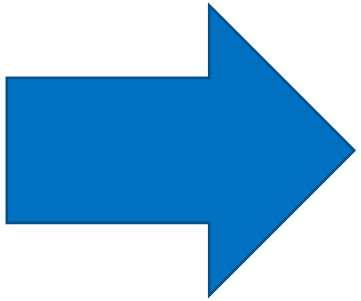


Figure 2: A new misleading bad comment detected by our tool in the *latest* version of Mozilla. It has been confirmed by the Mozilla developers, who replied us "I should have removed that comment about needing to hold the lock when calling ConvertToSID".

Comments and source code provide relatively **redundant and independent** information about a program's semantic behavior, cre-

databases and found that at least 62 bug reports in FreeBSD [4] are only about incorrect and confusing comments. For example, FreeBSD patch "kern/700" only modifies a comment in the file /sys/net/if.h. Similarly, the Mozilla patch for bug report 187257 in December 2002 only fixed a comment in file FixedTableLayoutStrategy.h.

The bug and bad comment examples above indicate that it is very important for programmers to maintain code-comment consistency; and it is also highly desirable to automatically detect bad comments so that they can be fixed before they mislead programmers and cause damages.

To the best of our knowledge, *no tool has ever been proposed to automatically analyze comments written in natural language and detect inconsistencies between comments and source code*. Almost all compilers and static analysis tools simply skip comments as if they do not exist, losing the opportunity to use comments to their maximum potential as well as to detect bad comments.

Project:@tComment
What is the possible future work?

My Passion

- Software Testing
  - I like finding bugs instead of writing programs!

Current Paper that I read

- Comment –Code Inconsistencies

What if we combine both?

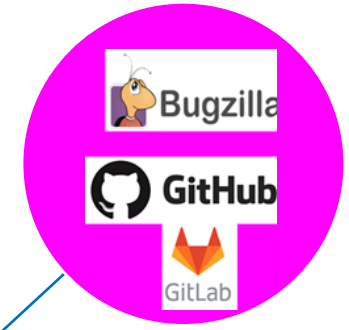
Project:@tComment Proposing a new idea

✉ Darko, I have spent several sleepless nights thinking about the topics for my Master thesis. Below are the ideas that I have:

- What is the relationship between testing and comment?
- …

✉ There is no need to spend sleepless nights thinking about topics.
>>What is the relationship between testing and comment?
This question is interesting

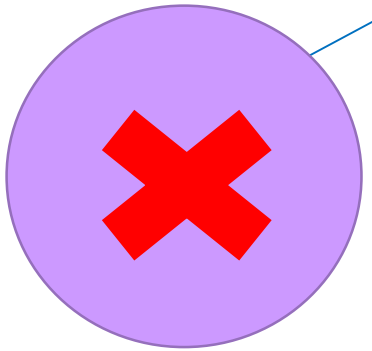
Finding Research Idea



- Read issues reported by developers in open-source project

- Read papers and answer the questions:
  - What is the potential future improvement for this work?

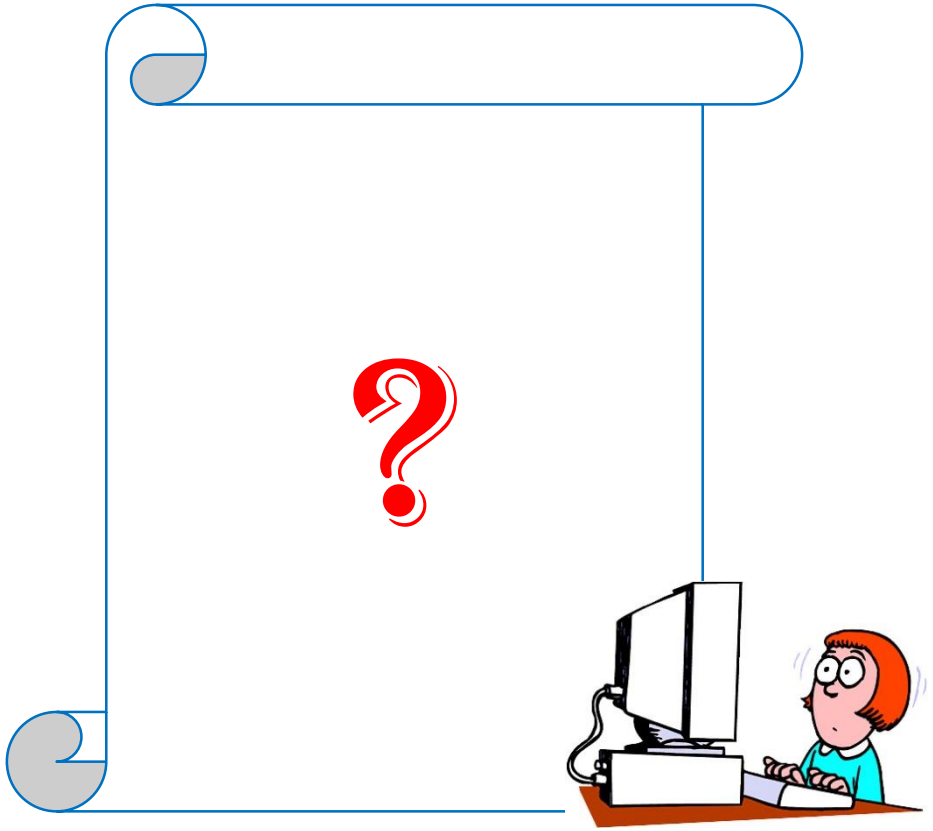
- Identify frequently encountered problems during software development



- Identify the weakness of existing tools

What are the problems you faced as a developer during software development?

ICSE 2019: (Basic) Analysis of Accepted Papers

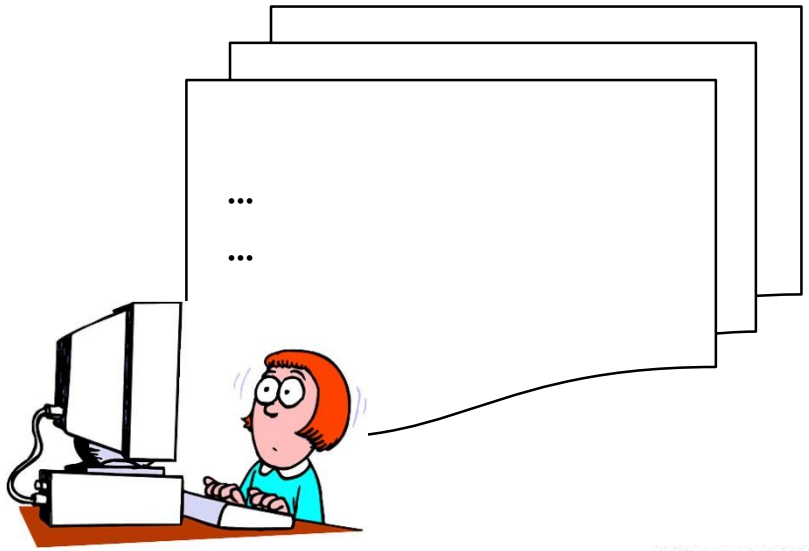


What are the problems that I faced as a developer during software development?

ICSE 2019: (Basic) Analysis of Accepted Papers

- How to find bugs?
  - What happen if code evolves but comments are not updated?
- How to fix broken tests?
  - What happen if code evolves but tests are not updated?
- How to fix bugs?
  - What happen if we find a bug?

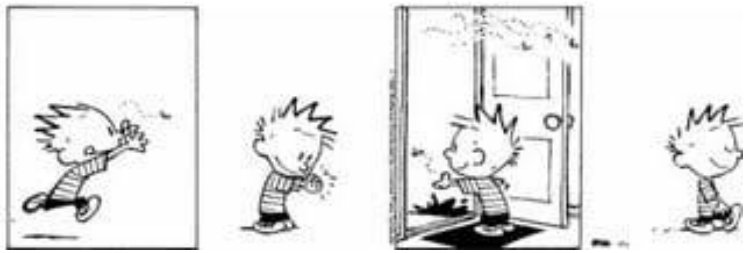
Identify frequently encountered problems



Test 1	✗	✓	Bug Fix
Test 2	✓	✓	
Test 3	✓	✗	

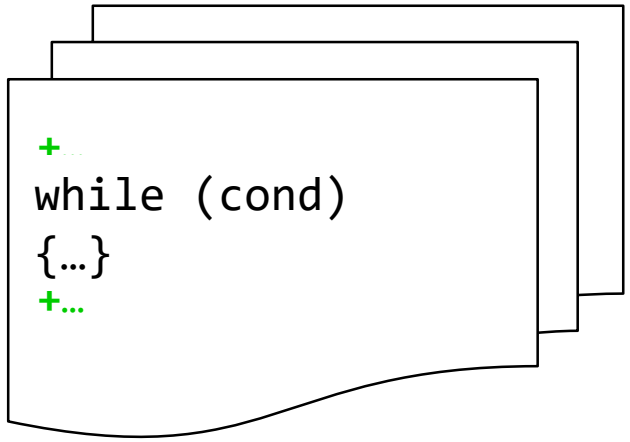
Regression!

Regression:
"when you fix one bug, you introduce several newer bugs."



How do developer repair regression?

Program Changes



Test Suite

Test 1 ✓

Test 2 ✓

Test 3 ✓

Regression Fixed!

Repair Goal: Ensure all tests in the test suite passing after the repair.

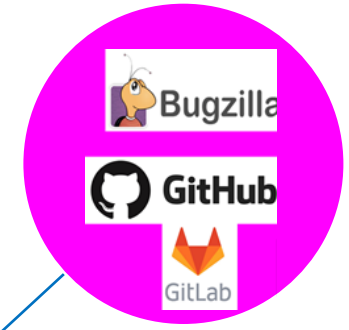
Project:relifix
Repairing Software Regression

relifix: Automated Repair of Software Regressions

Shin Hwei Tan and Abhik Roychoudhury
National University of Singapore
{shinhwei,abhik}@comp.nus.edu.sg

***Abstract*—Regression occurs when code changes introduce failures in previously passing test cases. As software evolves, regressions may be introduced. Fixing regression errors manually is time-consuming and error-prone. We propose an approach of automated repair of software regressions, called *relifix*, that considers the regression repair problem as a problem of reconciling problematic changes. Specifically, we derive a set of code transformations obtained from our manual inspection of 73 real software regressions; this set of code transformations uses syntactical information from changed statements. Regression repair is then accomplished via a search over the code transformation**

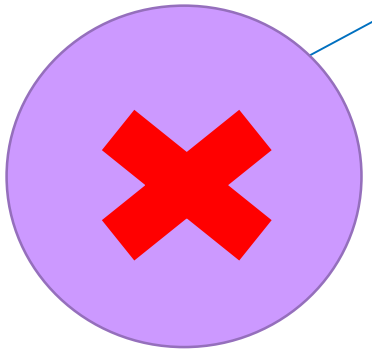
Nguyen et al. employed symbolic execution and component-based program synthesis for discovering the code required for fixing the buggy program [44]. Kim et al. proposed an automated patch generation approach (i.e., PAR) that utilizes common fix patterns learned from manual inspection of human patches [35]. Recent study shows that statements or expressions required for fixing exist in previous commits of the programs [28], [41]. However, existing automated program repair techniques have not fully exploited information from the software change history for automated repair of regressions



- Read issues reported by developers in open-source project

- Read papers and answer the questions:
  - What is the potential future improvement for this work?

- Identify frequently encountered problems during software development



- Identify the weakness of existing tools

Identify Weakness of Existing Tools

GenProg

```
static void BadPPM(char* file) {
    fprintf(stderr, "%s: Not a PPM file.\n", file);
-    exit(-2);
}
```

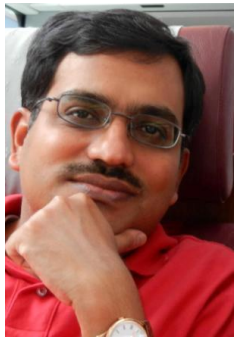
SPR

```
+ if ((type != 0))
+    return;
zend_error((1<<3L),"Uninitialized string offset:",...);
```

Project:Anti-patterns Collaborating with your advisor

Instead of looking at correct patches from human-written patches in approaches like PAR, identify rules for filtering "bad patches" generated by automatically generated patches.

There are pattern-based approaches like PAR while we are looking at the opposite. Let's call it anti-patterns!



Project: Anti-patterns in Search-Based Program Repair

Anti-patterns in Search-Based Program Repair

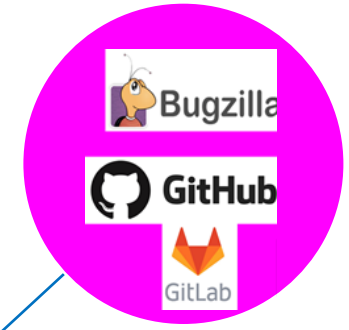
Shin Hwei Tan[*,†] Hiroaki Yoshida[‡] Mukul R. Prasad[‡] Abhik Roychoudhury[†]
[†]National University of Singapore, Singapore
[‡]Fujitsu Laboratories of America, Inc., Sunnyvale, CA, USA
{shinhwei,abhik}@comp.nus.edu.sg
{hyoshida,mukul}@us.fujitsu.com

ABSTRACT

Search-based program repair automatically searches for a program fix within a given repair space. This may be accomplished by retrofitting a generic search algorithm for program repair as evidenced by the GenProg tool, or by building a customized search algorithm for program repair as in SPR. Unfortunately, automated program repair approaches the promise of automatically suggesting fixes to "easy-to-fix" programming errors, thereby relieving substantial burden from programmers on the manual effort of debugging and generating fixes.

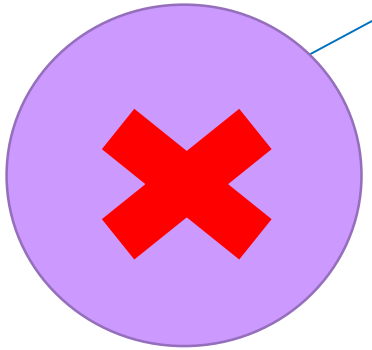
A major challenge in automated program repairs arises from the "incomplete specification" of intended behavior. Indeed, any repair technique tries to patch errors so as to

What is your secret of finding research idea?



- Read issues reported by developers in open-source project

- Read papers and answer the questions:
  - What is the potential future improvement for this work?



- Identify the weakness of existing tools

- Identify frequently encountered problems during software development

What is your research vision?
What will the future of software development?

The power of imagination

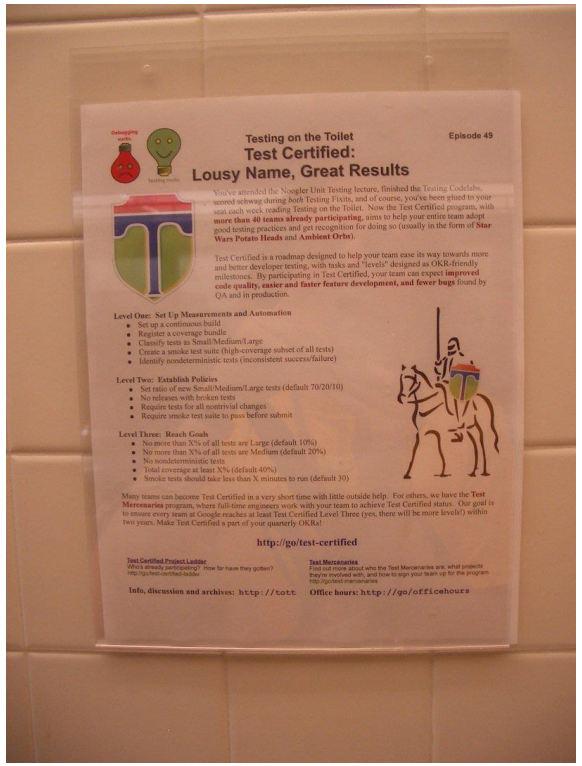
Let's think for a minute

Imagine that you are a programmer living in the year 2029
How would programming in the future look likes?

Programming in Toilet?

Programming in Bed?

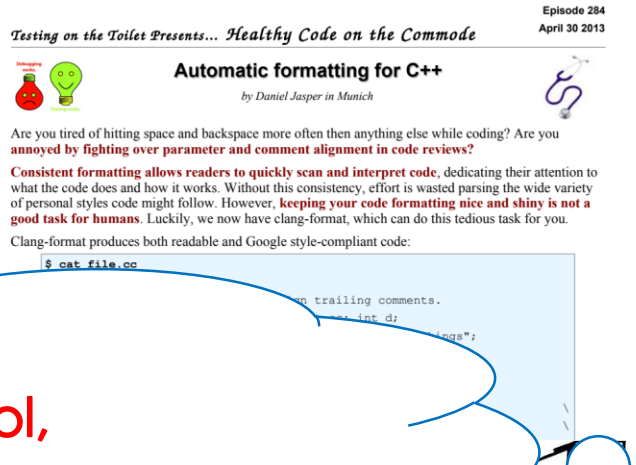
PROGRAMMING IN TOILET?



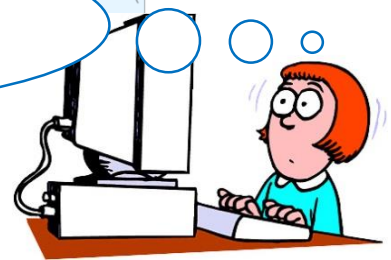
Do Developers Discover New Tools On The Toilet?

Emerson Murphy-Hill, *Google, LLC*, emersonm@google.com
Edward K. Smith*, *Bloomberg*, esmith404@bloomberg.net
Caitlin Sadowski, *Google, LLC*, supertri@google.com
Ciera Jaspan, *Google, LLC*, ciera@google.com
Collin Winter*, *Waymo*, collinwinter@waymo.com
Matthew Jorde, *Google, LLC*, majorde@google.com
Andrea Knight, *Google, LLC*, aknight@google.com
Andrew Trenk, *Google, LLC*, atrenk@google.com
Steve Gross, *Google, LLC*, stevegross@google.com

***Abstract*—Maintaining awareness of useful tools is a substantial challenge for developers. Physical newsletters are a simple technique to inform developers about tools. In this paper, we evaluate such a technique, called Testing on the Toilet, by performing a mixed-methods case study. We first quantitatively evaluate how effective this technique is by applying statistical causal inference over six years of data about tools used by thousands of developers. We then qualitatively contextualize these results by interviewing and surveying 382 developers, from authors to editors to readers. We found**



Instead of promoting the usage of new tool, could we actually program in toilet?



PROGRAMMING IN SLEEP?

Fixing Bugs in Your Sleep: How Genetic Improvement Became an Overnight Success

Saemundur O. Haraldsson*
University of Stirling
Stirling, United Kingdom FK9 4LA
soh@cs.stir.ac.uk

John R. Woodward
University of Stirling
Stirling, United Kingdom FK9 4LA
jrw@cs.stir.ac.uk

Alexander E.I. Brownlee
University of Stirling
Stirling, United Kingdom FK9 4LA
sbr@cs.stir.ac.uk

Kristin Siggeirsdottir
Janus Rehabilitation Centre
Reykjavik, Iceland
kristin@janus.is

ABSTRACT

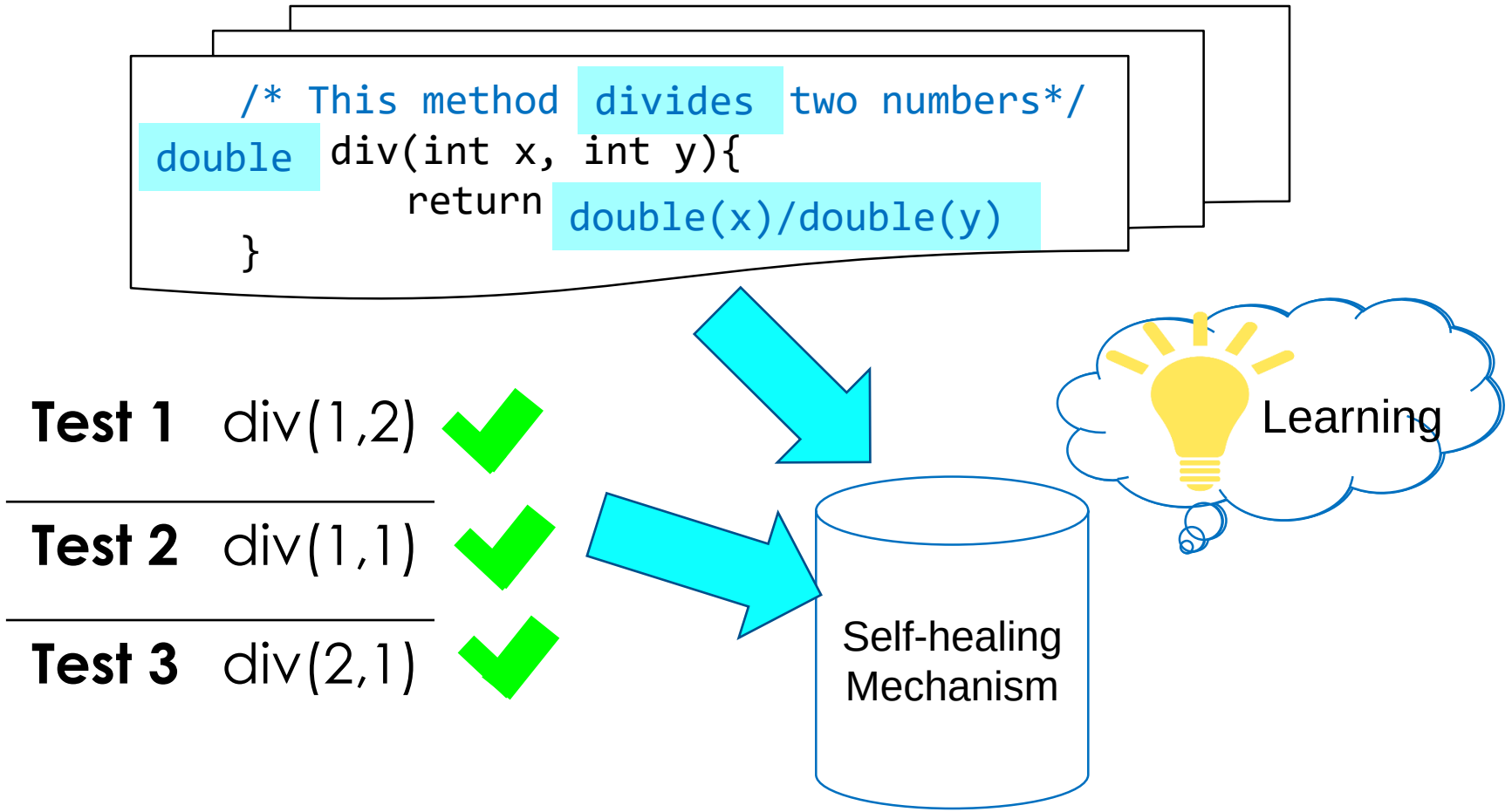
We present a bespoke live system in commercial use with self-improving capability. During daytime business hours it provides an overview and control for many specialists to simultaneously schedule and observe the rehabilitation process for multiple clients. However in the evening, after the last user logs out, it starts a self-analysis based on the day's recorded interactions. It generates test data from the recorded interactions for Genetic Improvement to fix any recorded bugs that have raised exceptions. The system has already been under test for over 6 months and has in that time

1 INTRODUCTION

Genetic Improvement (GI) [38] is a growing area within Search Based Software Engineering (SBSE) [23, 24] which uses computational search methods to improve existing software. Despite its [illegible] within academic research the practical usage of GI has not [illegible] like with many SBSE applications, the software in- [illegible] period for new ideas where they come to t[illegible] as cost effective solutions. GI is i[illegible] for the latter as it present[illegible]

During daytime business hours it provides an overview and control for many specialists to simultaneously schedule and observe the rehabilitation process for multiple clients. However in the evening, after the last user logs out, it starts a self-analysis based on the day's recorded interactions. It generates test data from the recorded interactions for Genetic Improvement to fix any recorded bugs that have raised exceptions.

My Research Vision: Self-healing Software



My Research Journey

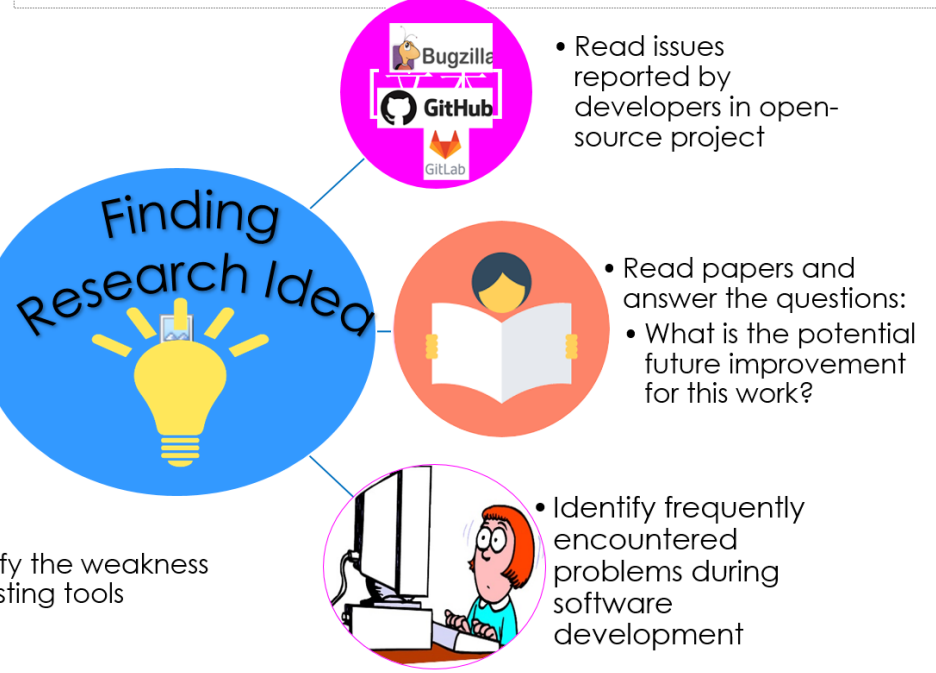
Repair broken assertions
Research Topic: Automated repair for broken assertions

Automated Program Repair
Research Topic: Repairing software regressions

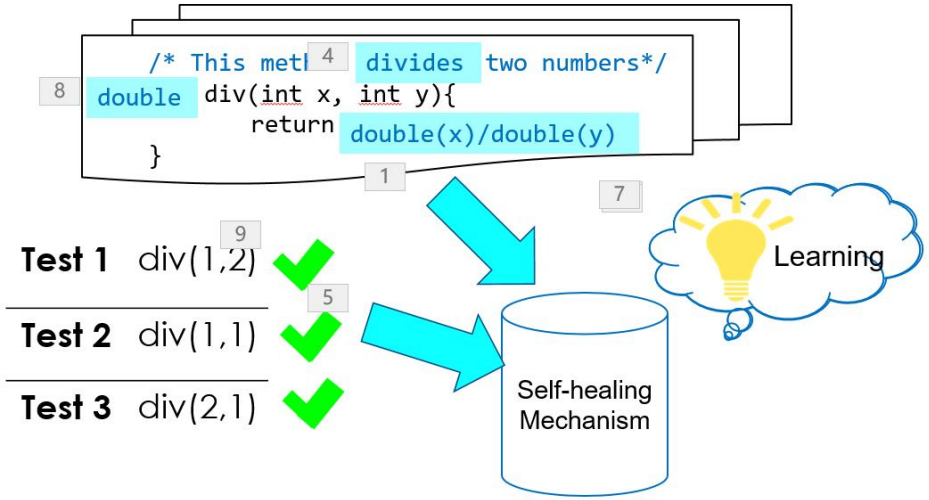
Unit Testing
Research Topic: Theories/Parameterized tests for Junit.

Comment-Code Inconsistency
Research Topic: Testing Comment-Code Inconsistencies

Repair Android Apps
Research Topic: Repairing crashes in Android apps



My Research Vision: Self-healing Software



HOW ABOUT YOUR RESEARCH JOURNEY?