

AI 驱动 软件研发 全面进入数字化时代

中国·北京 08.18-19

AI+
software
Development
Digital
summit



基于大模型的程序修复

刘逵 华为

科技生态圈峰会 + 深度研习 ——1000+ 技术团队的选择



2023K+
全球软件研发行业创新峰会
上海站

会议时间 | 06.09-10



2023K+
全球软件研发行业创新峰会
北京站

会议时间 | 07.21-22



2024K+
全球软件研发行业创新峰会
深圳站

会议时间 | 05.17-18



K+峰会详情



会议时间 | 08.18-19

AiDD AI+软件研发数字峰会
北京站



会议时间 | 11.17-18

AiDD AI+软件研发数字峰会
深圳站



AiDD峰会详情



演讲嘉宾



刘逵

华为2012软件工程应用技术实验室智能化软件工程技术专家

华为2012实验室智能化软件工程技术专家，CCF会员，CCF软件工程专委会委员。主要从事软件编码、软件测试、程序分析、代码检视等智能化软件工程技术工作，在软件工程领域发表高水平研究论文40余篇，其中CCF A类期刊/会议论文十多篇，曾任南京航空航天大学计算机科学与技术学院副教授，主持过国家自然科学基金面上项目一项，江苏省自然科学基金青年项目一项，参与国家重点研发计划项目两项，担任IEEE TSE、ACM TOSEM、EMSE、《软件学报》、ICSME、SANER等国际期刊/会议审稿人。

目录

CONTENTS

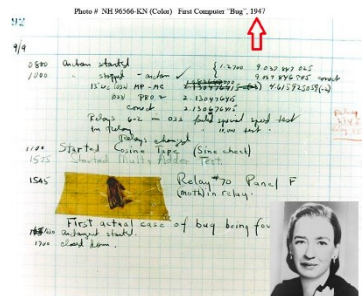
1. 软件缺陷带来的问题
2. 修复软件缺陷的挑战
3. 软件缺陷自动修复的探索与实践
4. 大模型下软件缺陷自动修复的探索
5. 总结与展望

PART 01

软件缺陷带来的问题



软件缺陷难以避免，也备受关注



Grace Hopper

第一个Bug¹

¹“CE: World’s First Computer Bug”, Sep 9, 1947, Society.



Windows 2000的发布版本伴随着**63000**个已知、在档的Bug³

³“Will bugs scare off users of new windows 2000”, 2000, Rick Lockridge, CNN.



媒体关注软件代码Bug²

²“Why we all sell code with bugs”, May 24, 2006, The Guardian.



行业技术报告⁴

⁴“So many bugs, so little time”, Jul 2010, MIT Technology Review.



软件缺陷带来的损失



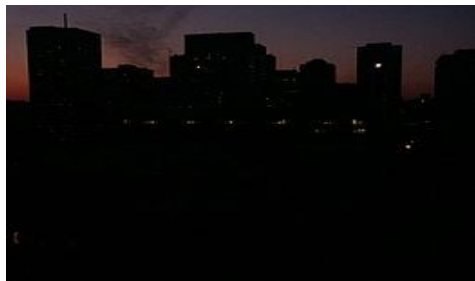
2020年富士通开发的证券交易系统存在缺陷导致四个半小时的停机事故⁵

⁵https://en.wikipedia.org/wiki/Tokyo_Stock_Exchange



胰岛素剂量控制软件存在缺陷造成病例死亡⁶

⁶Michael Zhivich and Robert K Cunningham. The real cost of software errors. IEEE Security & Privacy, 7(2):87–90, 2009.



电网管理软件存在重大缺陷造成美加大停电、近一百人死亡⁷

⁷"Northeast blackout of 2003",
https://en.wikipedia.org/wiki/Northeast_blackout_of_2003



2006年巴西天马航空3054航班空难，187人丧生，防撞系统问题⁸

⁸"TAM Airlines Flight 3054",
https://en.wikipedia.org/wiki/TAM_Airlines_Flight_3054

软件缺陷会造成巨额的损失

2002年，软件缺陷造成美国**595亿美元**的损失，占了美国当年0.6%的GDP。至少**1/3的损失**可以通过有效的**软件缺陷鉴定和修复**来避免⁹。
2016年，软件缺陷造成的损失突破到了**1100亿美元**¹⁰。

⁹. http://www.abeacha.com/NIST_press_release_bugs_cost.html

¹⁰. <https://medium.com/@ryancohane/financial-cost-of-software-bugs-51b4d193f107>

PART 02

修复软件缺陷的挑战



什么会有这么多的软件缺陷?

Support The Guardian
Available for everyone, funded by readers
Contribute → Subscribe →

Search jobs Sign in Search International edition

The Guardian

News Opinion Sport Culture Lifestyle More

World UK Environment Science Cities Global development Football Tech Business Obituaries

Inside IT Technology

Eric Sink

Thu 25 May 2006 01:56 BST



Why we all sell code with bugs

Creating quality software products means knowing when to fix bugs and when to leave well alone, writes Eric Sink

The world's six billion people can be divided into two groups: group one, who know why every good software company ships products with known bugs; and group two, who don't. Those in group 1 tend to forget what life was like before our youthful optimism was spoiled by reality. Sometimes we encounter a person in group two, a new hire on the team or a customer, who is shocked that any software company would ship a product before every last bug is fixed.

Every time Microsoft releases a version of Windows, stories are written about how the open bug count is a five-digit number. People in group two find that interesting. But if you are a software developer, you need to get into group one, where I am. Why would an independent software vendor - like SourceGear - release a product with known bugs? There are several reasons:

- We care about quality so deeply that we know how to decide which bugs are acceptable and which ones are not.
- It is better to ship a product with a known quality level than to ship a product full of surprises.

Read The Guardian without interruption on all your devices

Subscribe now

most viewed

Greta Thunberg in Twitter spat with G rail firm

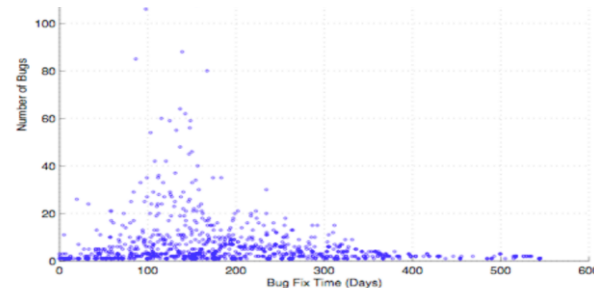
Two standard alcohol drinks a day no longer health officials say

Trump threatens C with 'years in jail' o Russia report

It's an Uggs game: v

2006年, Sunghun Kim等人的技术报告指出: 修复一个独立的软件缺陷的中间时间是200天²。

Sunghun Kim and E James Whitehead Jr. How long did it take to fix bugs? In Proceedings of the 3rd international workshop on Mining software repositories, pages 173-174. ACM, 2006.



软件缺陷修复时间分布

Intelligent Machines

So Many Bugs, So Little Time

Tools that find serious bugs automatically could lead to safer, more stable software.

by Erica Naone

Jul 27, 2010

Several talks at the Black Hat security conference this week in Las Vegas will focus on tools that could make software safer by automatically searching for bugs—and pinpointing the ones that could be most dangerous.

Bug hunting used to be a painstaking process. Researchers found one at a time, figured out what caused it and what dangers it posed, and revealed it, to a software vendor or publicly, so that it could be fixed. But in recent years, popular software has improved, and bugs aren't so easy to find. On top of that, commercial programs are increasingly large and complex, making it time-consuming to manually search for potential bugs. However, new software tools are helping to automate the process, which may mean programs that work more



Popular

01. A new way to make quadratic equations easy

02. Tidal forces carry the mathematical signature of gravitational waves

50.1%
Programmers total work time spent doing admin*

49.9%
Programming time spent debugging**

人工修复软件缺陷的挑战

Debugging的巨额花费:

2001年, Boehm等人的技术报告指出: 修复缺陷开销逐年猛增¹。

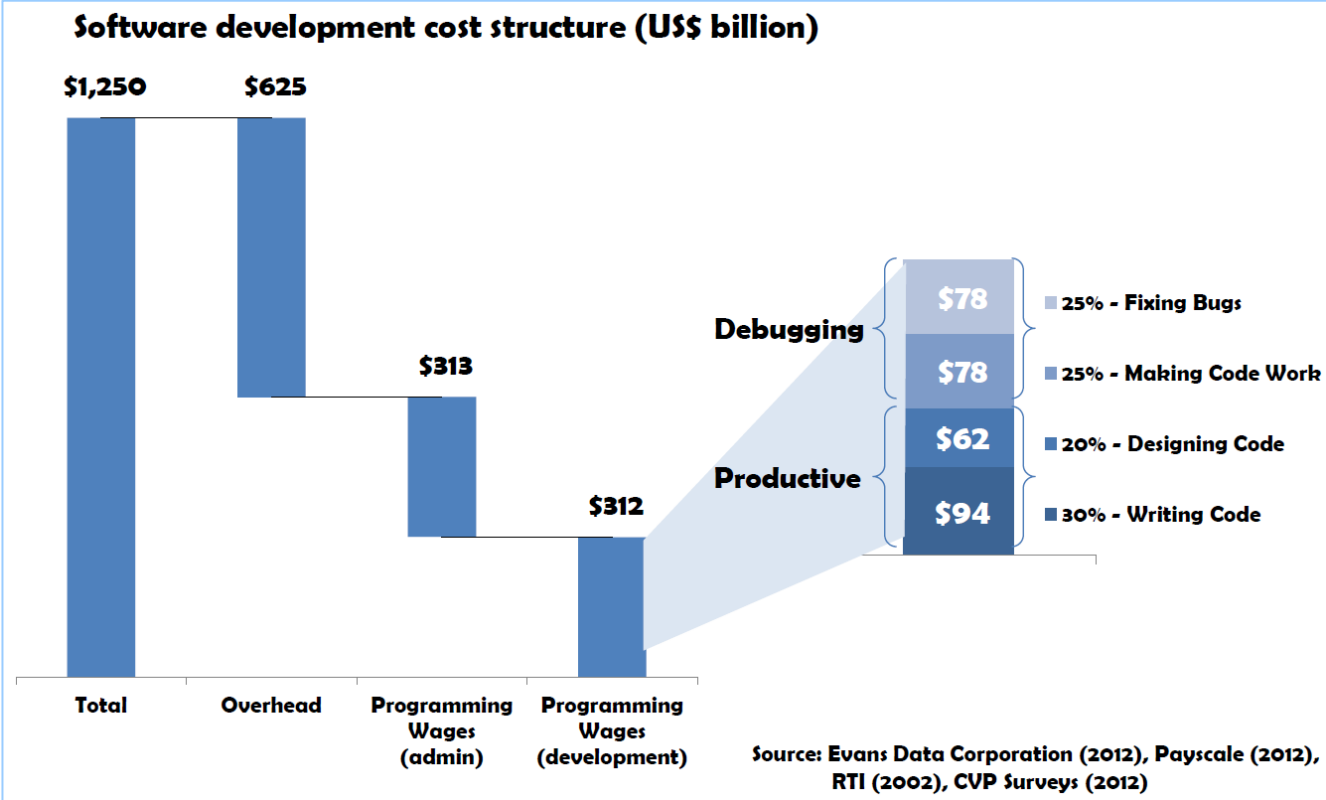
2003年, Seacord等人出版的一书中指出: “在一款软件产品发布以后, 其**90%的开销**都将花费对该软件产品的**Debugging**上”²。

2013年, 剑桥大学的技术报告声称: 程序员在Debugging上需要花费**一半软件开发时间和软件开发费用**³。

1. Barry Boehm and Victor R. Basili. Software Defect Reduction Top 10 List. Computer, Volume 34, Issue 1, January 2001, pp 135-137.

2. Robert C Seacord, Daniel Plakosh, and Grace A Lewis. Modernizing legacy systems: software technologies, engineering processes, and business practices. Addison-Wesley Professional, 2003.

3. Tom Britton, Lisa Jeng, Graham Carver, Paul Cheak, and Tomer Katzenellenbogen. Reversible debugging software. Technique Report in University of Cambridge, UK, 2013.



US\$156 Billion per year
Wages-only estimated cost of debugging*

US\$312 Billion per year
TOTAL estimated cost of debugging including wages and overheads**



Meta OPT-175B训练过程也遇到很多Bug

OPT训练过程中的故障统计

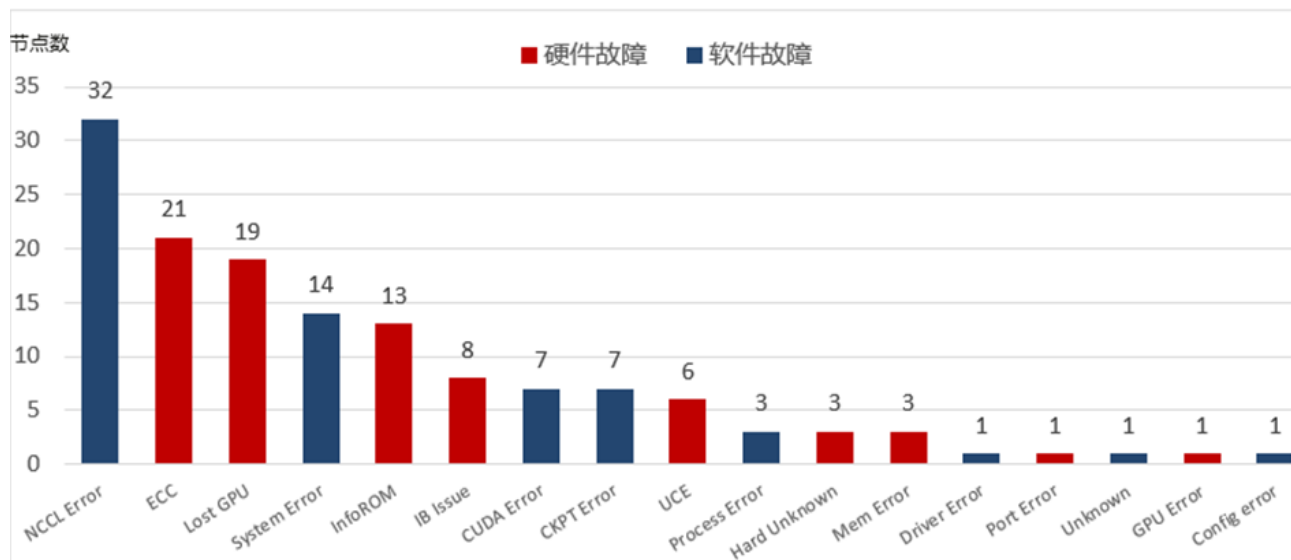
描述：OPT训练在**992个80GB显存A100**上训练，历时**3个月**，核心训练时长**56天**（2022年），结构大小（175B）和GPT3类似，过程中平均**每天约2.7个故障**，其中主要为软/硬件故障（92%）、损失发散。

- 软件：NCCL算子、计算算子
- 硬件：Lost GPU、ECC
- 模型：损失发散

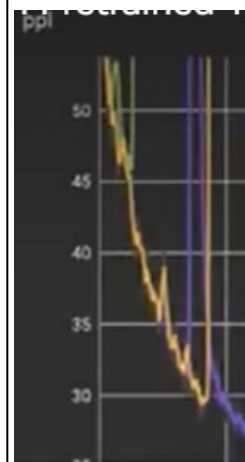
OPT故障处理

- 训练故障解决方式大多依靠任务重启。
- 集群在一定的容错能力，部分故障不中断训练，事后根据CKPT重启

故障类型	故障的节点数	处理方法
硬件故障	75 (49%)	检测节点，替换或重启节点，并利用CKPT重启任务
软件故障	66 (43%)	替换或重启节点，并利用CKPT重启任务
未知故障	6 (4%)	更换节点，CKPT重启任务
代码Bug	4 (4%)	修改并重新运行
部署错误	1 (1%)	向运维人员求助，并要求提供更多机器

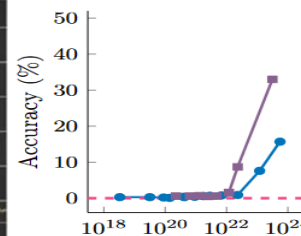
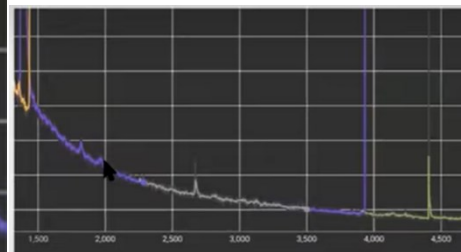


Scalable Metric 使能模型训练监控，这块目前完全依赖经验



GPU训练由于精度误差导致训练抖动，任务训练不搭预期

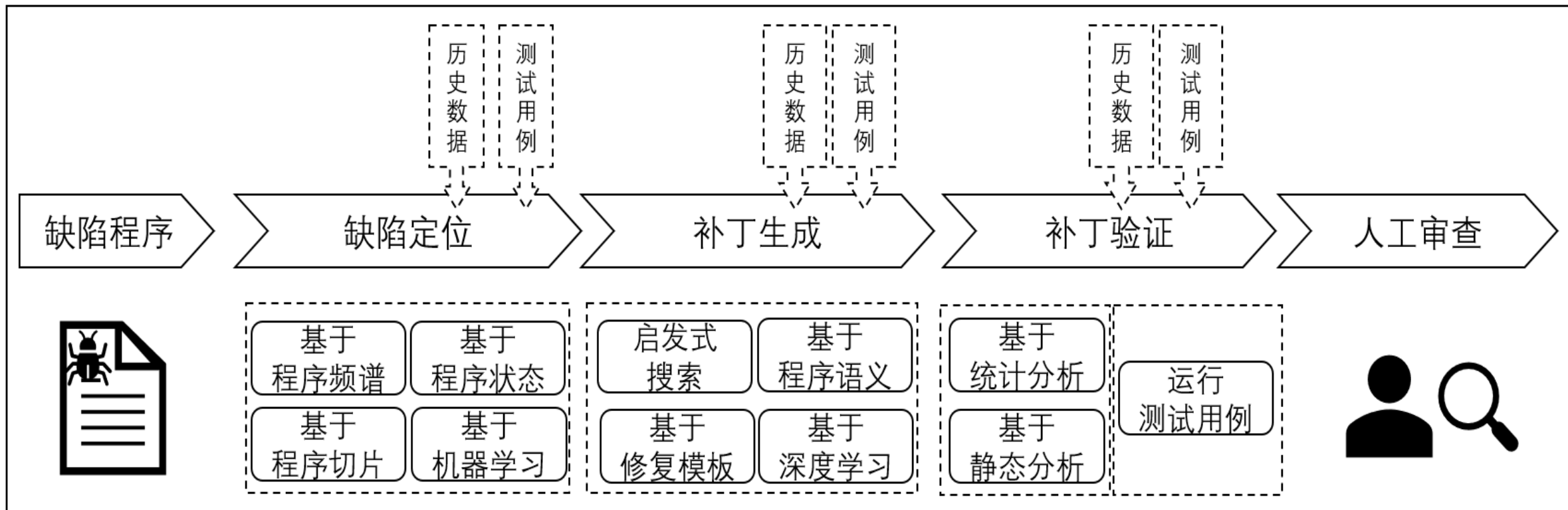
大模型能力涌现性加大了模型训练监控的难度。



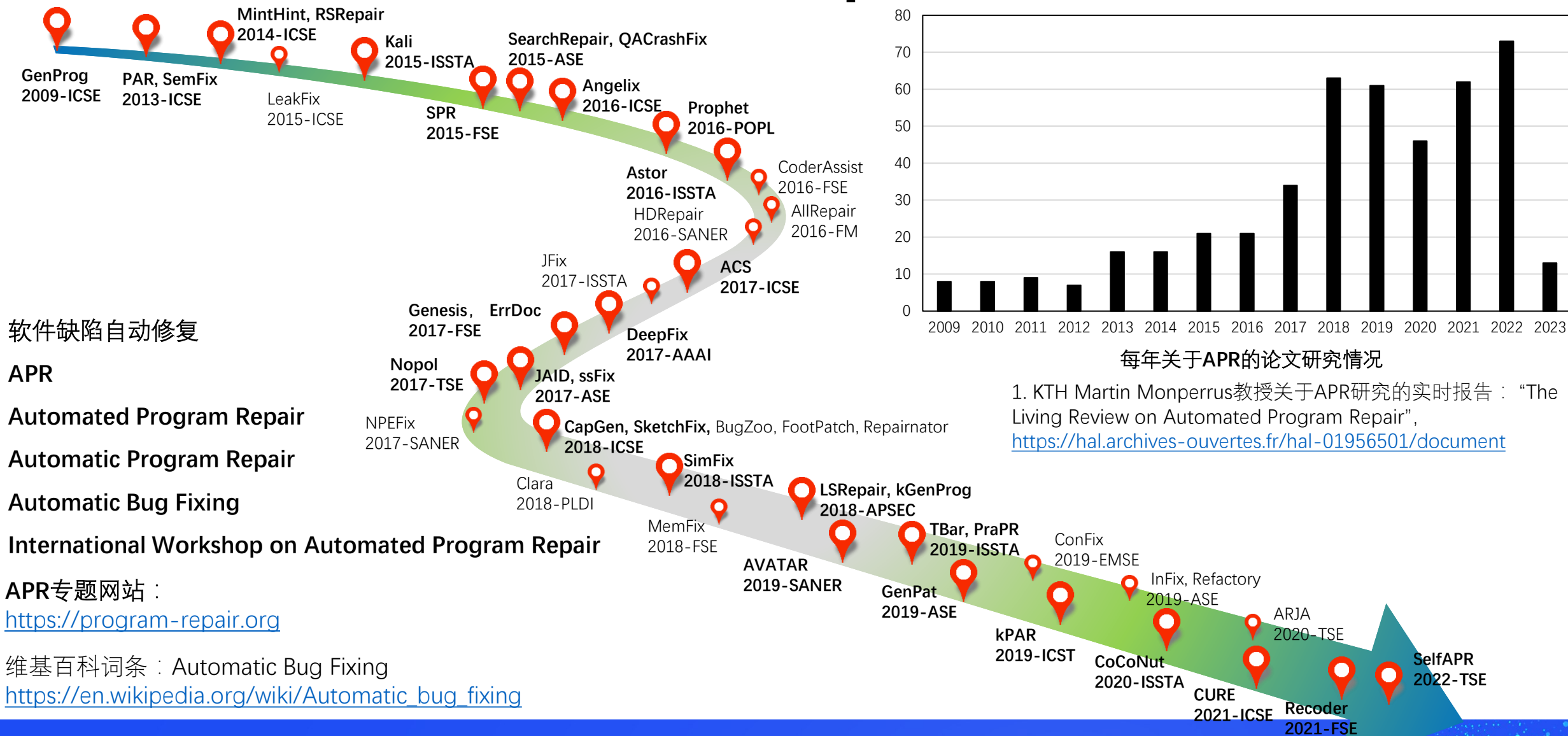
PART 03

软件缺陷自动修复的探索与实践

缺陷自动修复基本流程



软件缺陷自动修复的Road Map



软件缺陷自动修复

APR

Automated Program Repair

Automatic Program Repair

Automatic Bug Fixing

International Workshop on Automated Program Repair

APR专题网站：

<https://program-repair.org>

维基百科词条：Automatic Bug Fixing

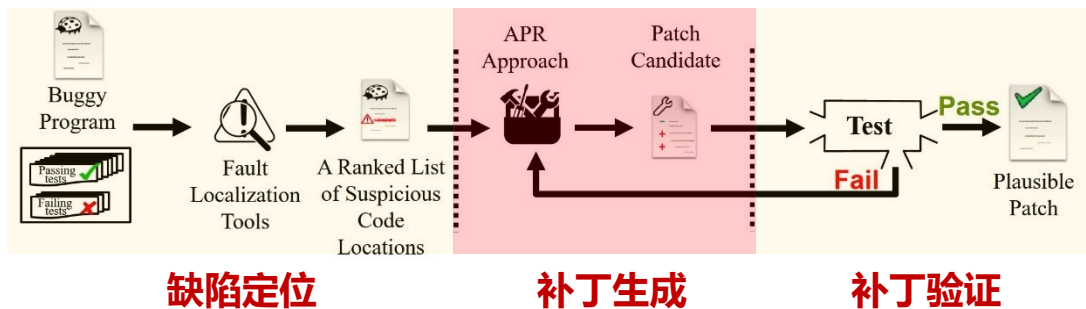
https://en.wikipedia.org/wiki/Automatic_bug_fixing

AI驱动软件研发全面进入数字化时代

iDDD AI+软件研发数字峰会
AI*software,Development Digital summit

缺陷自动修复技术的学术研究探索

软件缺陷自动修复技术的基本流程



缺陷定位: 根据分析缺陷导致的非正常现象与代码之间的关系, 找到引发该缺陷的类、函数、**代码行**¹。

- 基于统计的缺陷定位技术
- 基于程序状态的缺陷定位技术
- 基于数据挖掘的缺陷定位技术
- 基于模型的缺陷定位技术
- 基于信息检索的缺陷定位技术
- 基于学习的缺陷定位技术 (机器学习、深度学习)
- 基于程序切片的缺陷定位技术 (静态、动态、双重切片等等)
- 基于静态分析的缺陷定位技术 (SpotBugs, Infer, ErrorProne)
- **基于频谱的缺陷定位技术** (需要测试用例, Ochiai等等)

$$Suspicious_{Ochiai}(s) = \frac{T_f(s)}{\sqrt{(T_f(s) + T_p(s)) * (T_f(s) + T_f(\neg s))}}$$

1. Wong et al. A Survey on Software Fault Localization. IEEE TSE, 42(8), 707-740 (2016).

补丁生成: 通过既定的自动化方法, 自动地修改可疑的缺陷代码, 生成潜在的缺陷补丁。

- **基于启发式的补丁生成技术** (e.g., GenProg, 2009-ICSE, SimFix, 2018-ISSTA)
- **基于修复模板的补丁生成技术** (e.g., PAR, 2013-ICSE, TBar, 2019-ISSTA)
- **基于语义约束的补丁生成技术** (e.g., SemFix, 2013-ICSE, Nopol, 2017-TSE)
- **基于学习的补丁生成技术** (e.g., DeepFix, 2017-AAAI, Recoder, 2021-FSE)

更多信息可参详:

- 姜佳君等, 软件缺陷自动修复技术综述, 软件学报, 2021, 32(9):2665-2690
- Liu et al., On the Efficiency of Test Suite based Program Repair, ICSE-2020.

补丁验证: 通过自动化、智能化方法验证补丁的**可行性、正确性**。

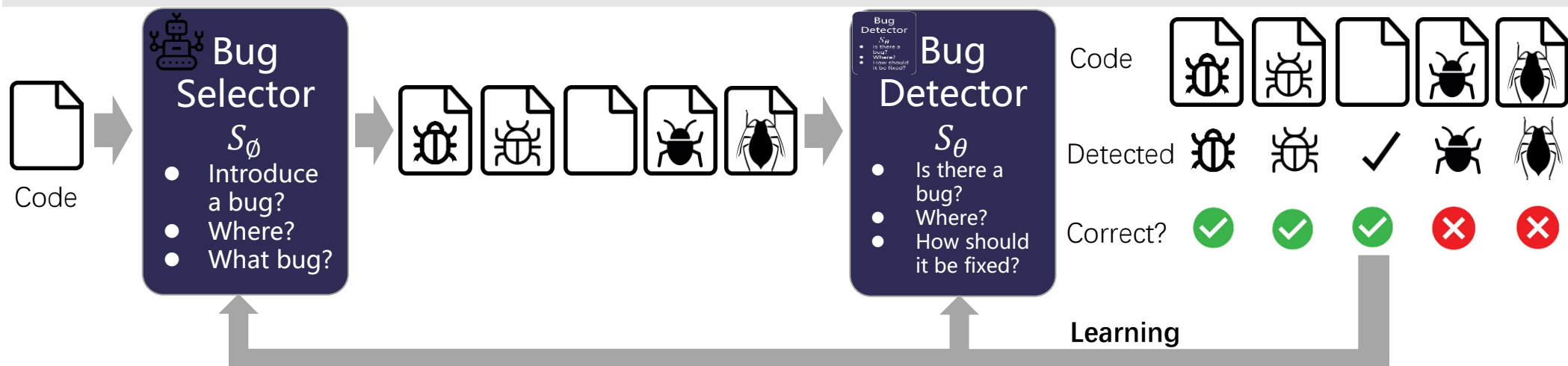
- 编译, APR工具容易生成无意义(nonsensical)的补丁, 存在语法错误, 无法通过编译。
- 测试, APR工具会产生无效的(invalid)补丁, 虽然可以通过编译, 但是不能让打过补丁的程序通过测试。
- 持续验证, APR工具产生的补丁虽然可以通过所有的测试, 但仍然无法确保该补丁的正确性, 会产生很多**貌似合理/过拟合的 (plausible) 补丁**。
 - ◆ 动态分析的方法, e.g., PatchSim, Xiong et al., 2018-ICSE.
 - ◆ 基于学习的方法, e.g., BATS, Tian and Liu et al, 2022-TOSEM.



业界的缺陷自动修复技术探索与实践

公司	APR技术	技术出处
微软	BugLab – 基于自监督学习的缺陷自动定位与修复技术。	NeurIPS-2021
FaceBook	全栈式缺陷预测、定位与修复技术	ISSTA-2016, OOPSLA-2019, ICSE-2019
谷歌	TriCoder 缺陷自动定位与补丁推荐	
富士通	缺陷自动定位与修复技术	ASE-2017, ICSE-2019
华为	可解释的缺陷预测与定位、可适配的缺陷自动修复	

微软BugLab: 基于自监督学习的程序自动定位与修复技术



受生成对抗网络的启发，基于自监督学习的缺陷定位与修复，无需带标签的训练数据，专注于Python代码4种Syntax Error缺陷，能修复26%的缺陷，但无法修复复杂的缺陷。

Allamanis, M., Jackson-Flux, H., & Brockschmidt, M. "Self-Supervised Bug Detection and Repair". NeurIPS, 2021



谷歌TriCoder: 缺陷自动定位与补丁推荐系统

TriCoder: 可扩展的程序分析平台, 旨在打造数据驱动的程序分析生态系统, 利用静态分析工具定位软件缺陷, 并生产补丁推荐。涵盖30个静态分析工具, 支持20种编程语言。每天可生产93000的分析结果, 只需2-3个维护人员。

- 生态化的分析平台: 采用微服务架构, 便于扩展; 允许公司中各团队开发面向特定语言、特定框架或特定需求的分析插件
- 低误报: 分析器的误报率必须低于10% (平台可以下线误报率超标的静态分析器)
- 修复辅助: 提供缺陷修复建议, 方便程序员采纳 (每天700次)
- 数据驱动: 持续收集修复反馈, 优化提升检测与修复能力
- workflow集成: 与代码检视工具顺畅衔接

```

package com.google.devtools.staticanalysis;

public class Test {

    ▼ Lint      Missing a Javadoc comment.
    Java
    4:02 AM, Aug 21
    Please fix Not useful

    public boolean foo() {
        return getString() == "foo".toString();
    }

    ▼ ErrorProne String comparison using reference equality instead of value equality
    StringEquality
    1:03 AM, Aug 21
    Please fix (see http://code.google.com/p/error-prone/wiki/StringEquality)
    Suggested fix attached: show Not useful
}

public String getString() {
    return new String("foo");
}
}

```

Tricoder的检查器会针对告警提供合理修复建议

"请修复"按钮

检视人员可以通过单击该按钮要求开发者必须修复该缺陷代码。在所有评论(手动和自动发现的)都得到解决之前, 检视人员通常不会批准合入代码变更

"无用"按钮

检视人员可以点击表示他们不认同分析发现的结果。点击动作会自动附带提交bug跟踪器中的错误, 并将其指向分析器所属团队

```

//depot/google3/java/com/google/devtools/staticanalysis/Test.java

package com.google.devtools.staticanalysis;

public class Test {
    public boolean foo() {
        return getString() == "foo".toString();
    }

    public String getString() {
        return new String("foo");
    }
}

```

存在缺陷的代码

```

package com.google.devtools.staticanalysis;

import java.util.Objects;

public class Test {
    public boolean foo() {
        return Objects.equals(getString(), "foo".toString());
    }

    public String getString() {
        return new String("foo");
    }
}

```

建议修复后的代码

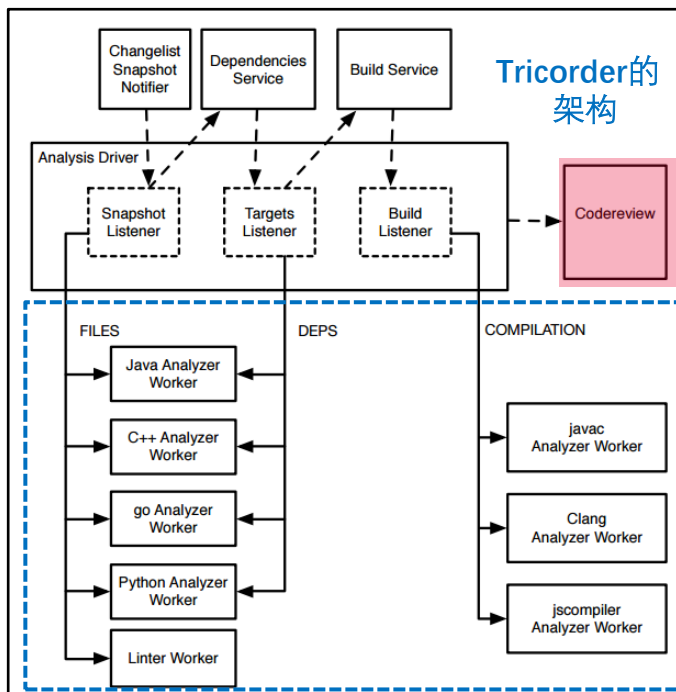
Apply Cancel

数据驱动

Tricoder团队跟进这些"无用"的点击标记, 计算"请修复"与"无用"之间的点击比率。如果分析器的比例超过10%, 那么Tricoder团队会禁用该分析器直到作者对其进行改进

应用规模

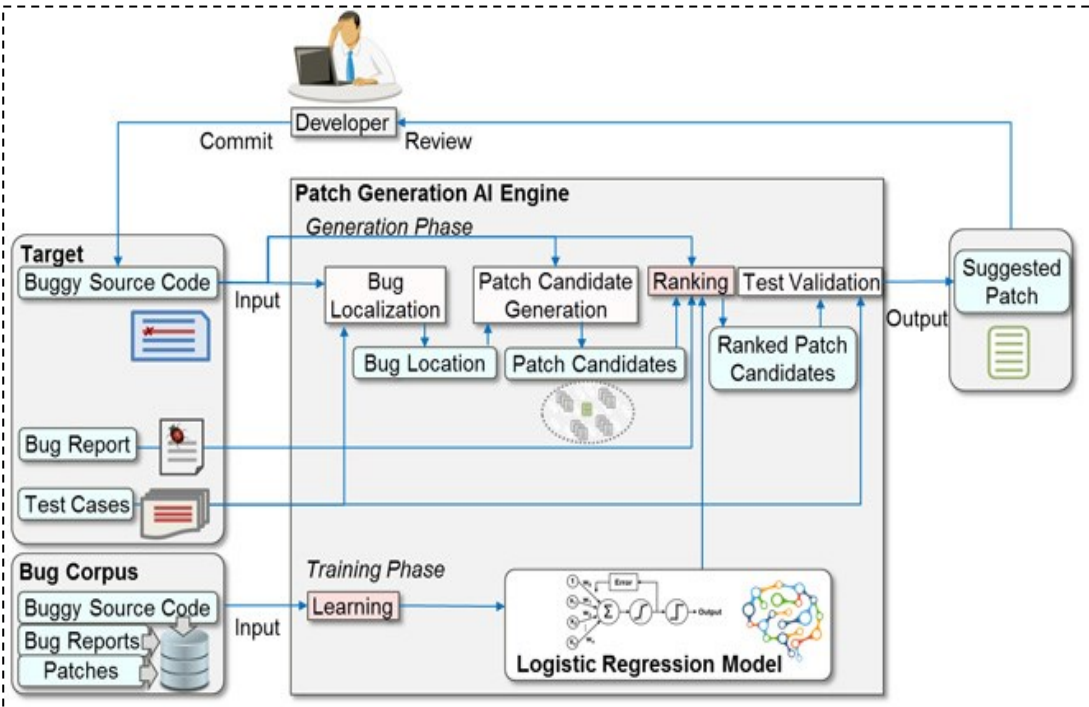
截至2018年1月, Tricoder平均每天分析约50000次的需要代码检视的变更, 其中检视人员点击"请修复"超过5000次, 作者采纳自动修复方案的约为3000次; Tricoder分析器每天收到250次"无用"点击反馈。



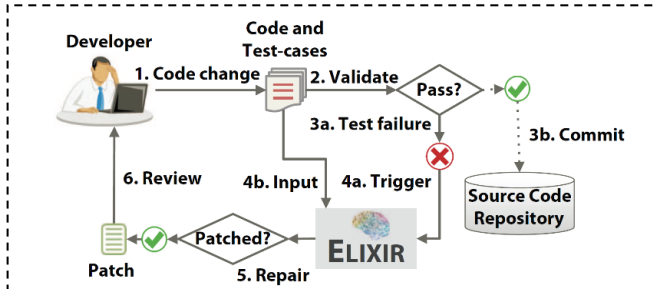
开放的分析平台

Tricoder设计理念旨在易于扩展, 这个平台既包含了开源分析工具, 也有Tricoder团队开发的工具, 当前的更多来自公司内部团队自研工具。截至2018年1月, 该平台共包括146个分析器, 其中125个来自Tricoder团队之外

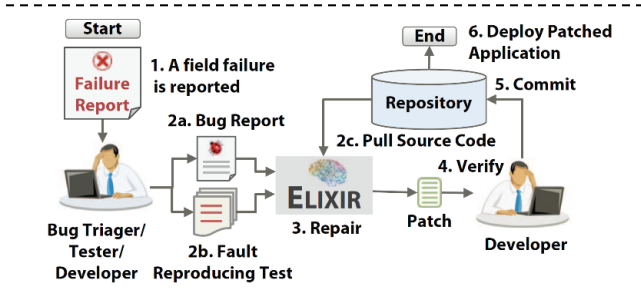
富士通：全栈式缺陷自动定位与修复技术



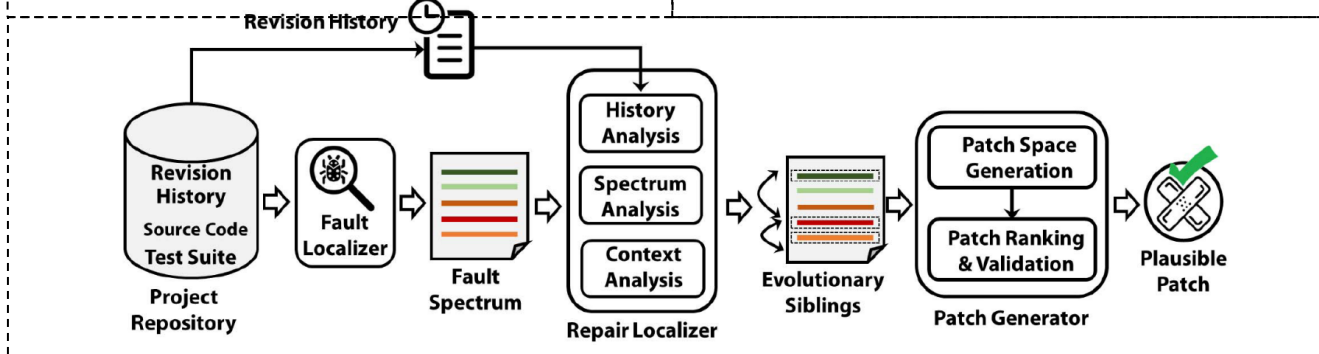
Fujitus全栈式缺陷自动定位与修复技术



支持软件开发期间的缺陷定位与修复



支持软件运维期间的缺陷定位与修复



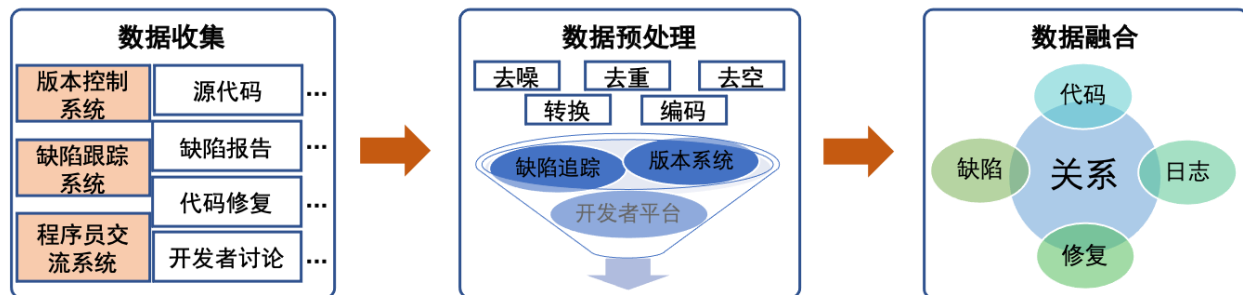
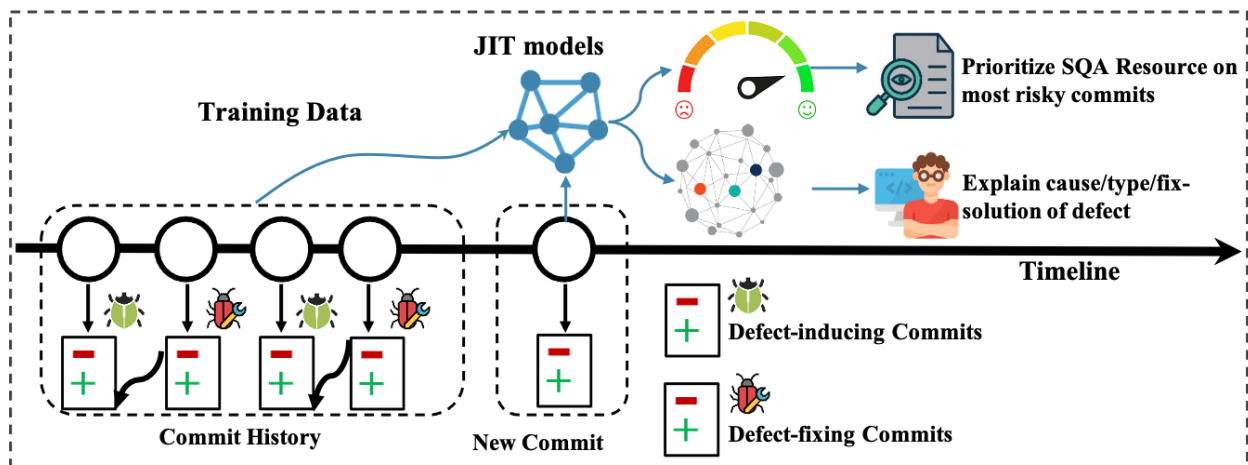
支持多程序段的缺陷定位与修复

- 整体方案
 - ❑ 采用基于生成并验证式修复方法，在算法设计阶段，富士通基于一组真实世界广泛、有差异的缺陷集训练了一个Ranking模型，以期更早地输出正确修复方案
- 实用分析
 - ❑ ELIXIR在富士通业务线最初集成时，得到的正确修复仅占10%，效果很差
 - ❑ 在完成对ELIXIR的失效分析后，新发布的修复工具正确率达到了40%
 - ❑ 实现了Eclipse ELIXIR插件
- 观察
 - ❑ 目标程序的测试用例与运行环境可得时，动态缺陷修复是一种可行方案
 - ❑ 动态缺陷修复是困难的，需要在E2E各个环节精细设计，针对不同的业务需要专有的设计

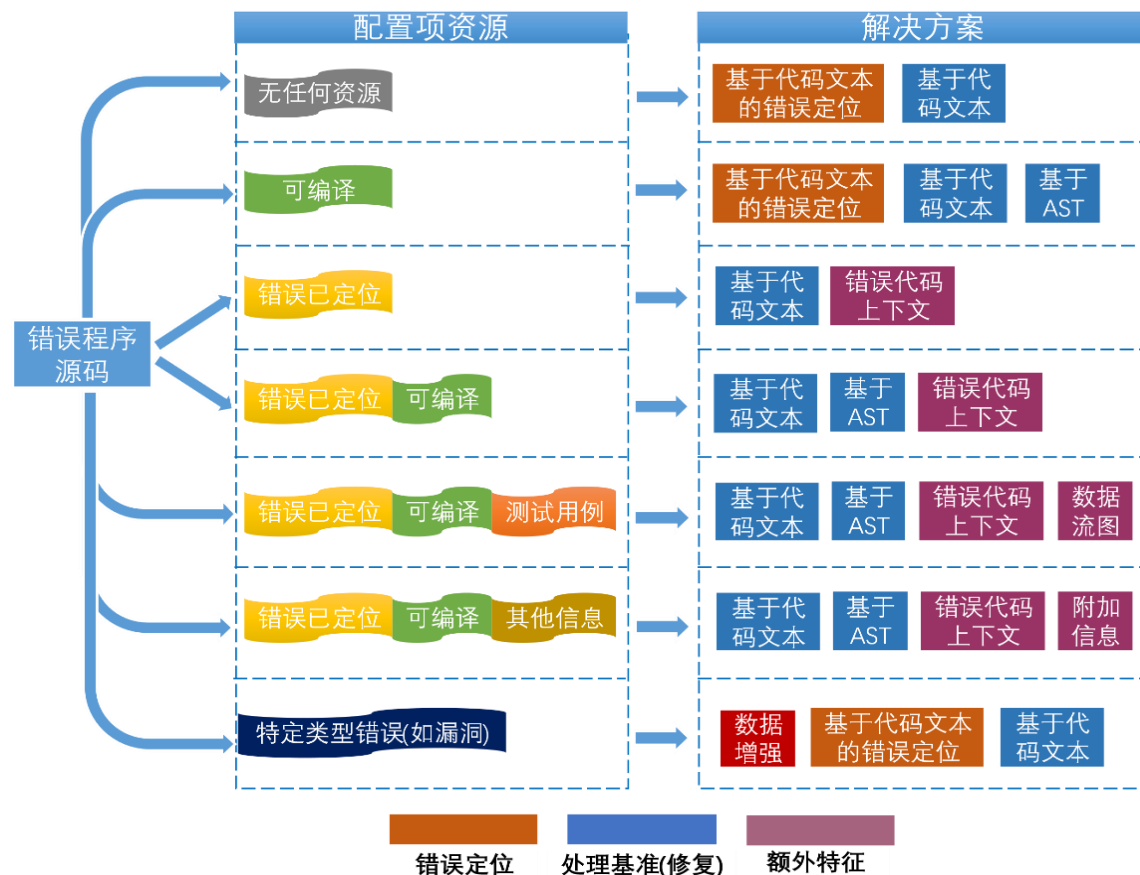
Elixir: Effective object-oriented program repair, ASE-2017
 Harnessing evolution for multi-hunk program repair, ICSE-2019

华为的软件缺陷自动修复探索与实践

可解释的缺陷预测与定位



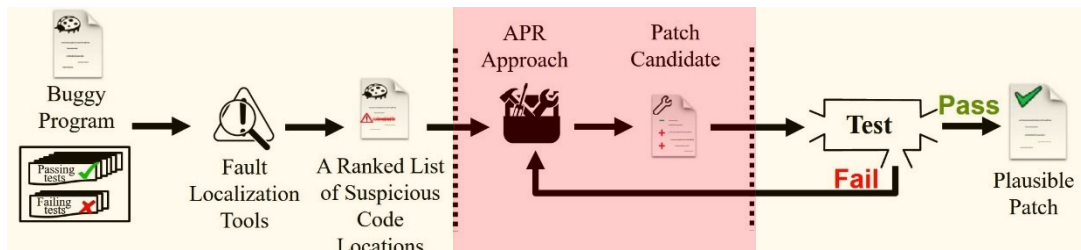
可适配的缺陷自动修复



PART 04

大模型下软件缺陷自动修复的探索

大模型对APR工具生成的补丁重排序



缺陷定位

补丁生成

补丁验证

Listing 2: TBar Lang57 Patch #1 (incorrect)

```
// Legend: token prob > 0.5 p > 0.1 p > 0.01 p <= 0.01
-220,7 +220,7
(...) boolean isAvailableLocale (Locale locale) {
- return cAvailableLocaleSet.contains(locale);
+ return (cAvailableLocaleSet.contains(locale)
+ || (locale.getVariant().length() > 0));
}
```

Listing 3: TBar Lang57 Patch #37 (correct)

```
-220,7 +220,7
(...) boolean isAvailableLocale (Locale locale) {
- return cAvailableLocaleSet.contains(locale);
+ return availableLocaleList().contains(locale);
}
```

Listing 4: SimFix Closure57 Patch #1 (incorrect)

```
-194,9 +194,16
Node target = callee.getNext();
if (target != null) {
- className = target.getString();
+ if(child!=null){
+ className=child.getString();
+ }
}
```

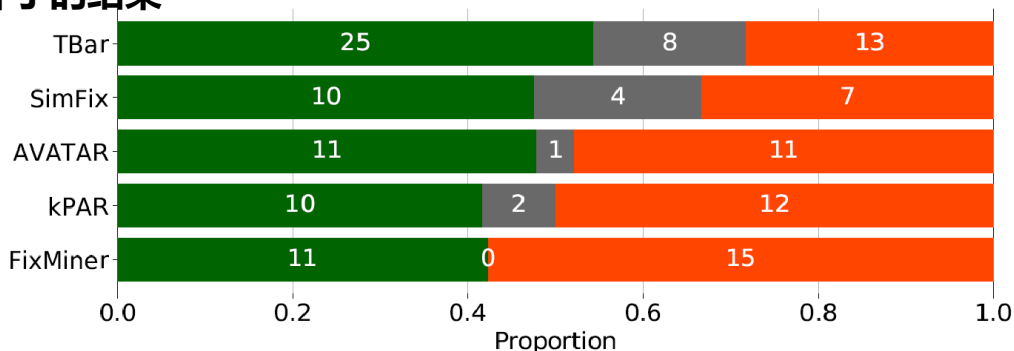
Listing 5: SimFix Closure57 Patch #27 (correct)

```
-194,9 +194,16
Node target = callee.getNext();
if (target != null) {
+ if(target!=null&&target.getType()==Token.STRING){
+ className=target.getString();
+ }
}
```

■ better ■ same ■ worse

重排5个APR工具生成Defects4J补丁的结果

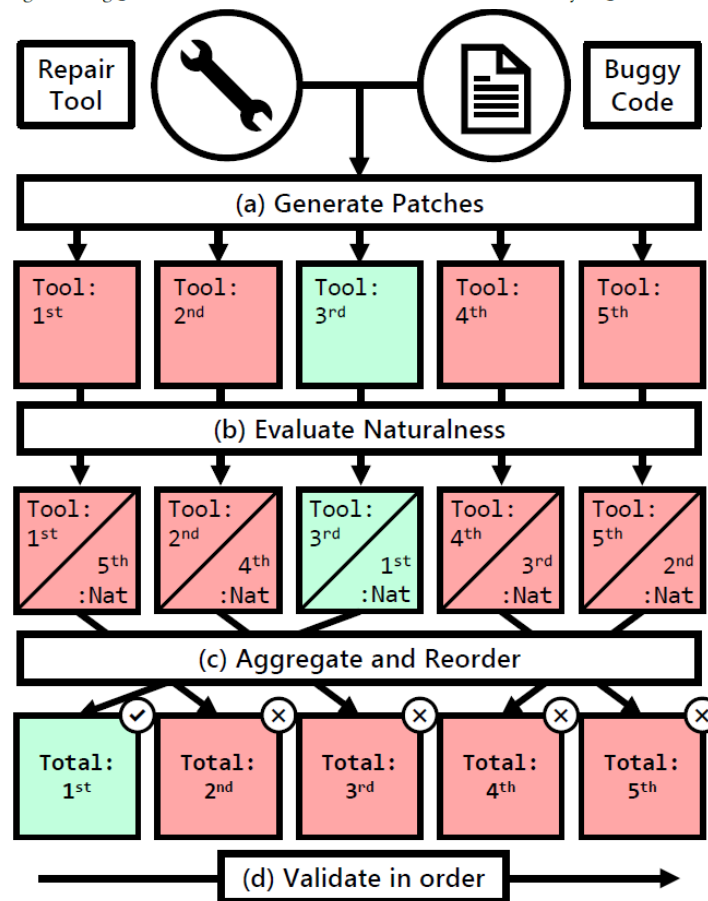
APR Tool	Default				LM			
	B	S	L	p	B	S	L	p
TBar	29	2	15	.024	40	1	5	<.01
SimFix	16	2	3	<.01	14	2	5	.031
AVATAR	12	0	11	.5	15	1	6	.026
kPAR	17	0	7	.031	16	0	8	.075
FixMiner	20	0	6	<.01	18	0	8	.037



Language Models Can Prioritize Patches for Practical Program Patching

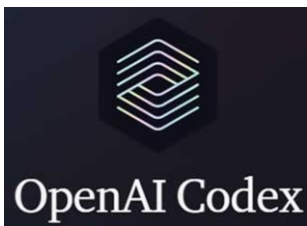
Sungmin Kang
KAIST
Daejeon, South Korea
sungmin.kang@kaist.ac.kr

Shin Yoo
KAIST
Daejeon, South Korea
shin.yoo@kaist.ac.kr





直接使用Codex修复QuixBugs的Bug



Prompt :

```
### fix the bug in the following function  
<buggy function and/or docstring here>
```

```
### fixed function
```

1. *Code only*: 有Bug的函数.
2. *Code with hint*: ### 提示Bug的行位置.
3. *Code with docstring* : 有Bug的代码和正确的功能描述

Can OpenAI's Codex Fix Bugs?

An evaluation on QuixBugs

Julian Aron Prenner
prenner@inf.unibz.it
Free University of Bozen-Bolzano
Italy

Hlib Babii
Hlib.Babii@stud-inf.unibz.it
Free University of Bozen-Bolzano
Italy

Romain Robbes
rrobbes@unibz.it
Free University of Bozen-Bolzano
Italy

QuixBugs基准数据集上的验证结果:

	Java	Python
DeepDebug [7]	-	21
CoCoNuT [19]	13	19
CURE [10]	26	-
Codex	14	23/21 ¹

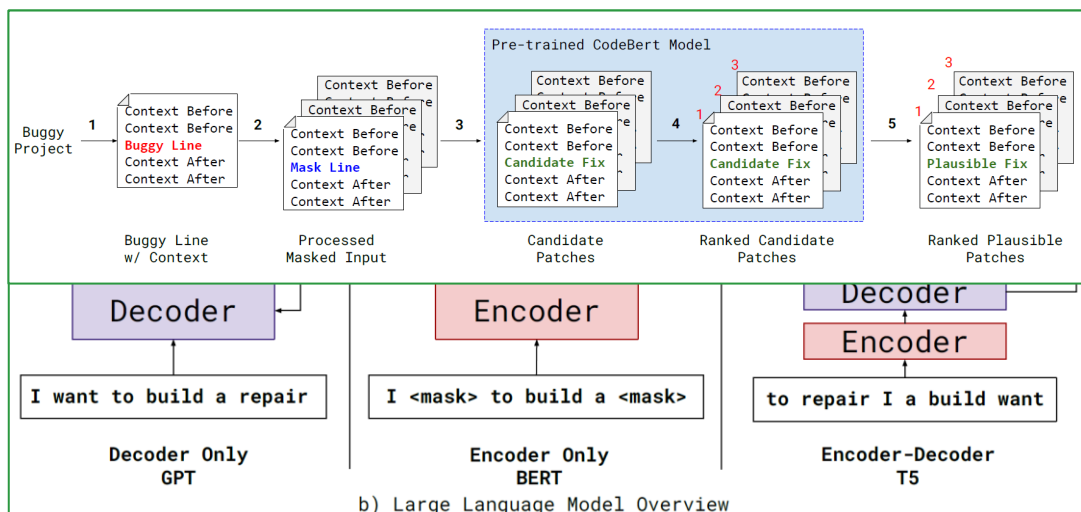
未来方向: *Fine-Tuning*.

零样本学习的缺陷修复

Less Training, More Repairing Please:
Revisiting Automated Program Repair via Zero-Shot Learning

Chunqiu Steven Xia
University of Illinois Urbana-Champaign
chunqiu2@illinois.edu

Lingming Zhang
University of Illinois Urbana-Champaign
lingming@illinois.edu



b) Large Language Model Overview

Defects4J基准集验证结果

Project	AlphaRepair	Recoder	TBar	CURE	CoCoNuT	PraPR	DLFix	SequenceR
Chart	9	10	11	10	7	7	5	3
Closure	23	21	16	14	9	12	11	3
Lang	13	11	13	9	7	6	8	2
Math	21	18	22	19	16	10	13	6
Mockito	5	2	3	4	4	3	1	0
Time	3	3	3	1	1	3	2	0
Total Correct / Plausible	74 / 109	65 / 112	68 / 95	57 / 104	44 / 85	41 / 146	40 / 68	14 / 19

QuixBugs基准集验证结果

Tool	AlphaRepair	CURE	DeepDebug	Recoder	CoCoNuT
Java	28 / 30	26 / 35	- / -	17 / 17	13 / 20
Python	27 / 32	- / -	21 / 22	- / -	19 / 21

Context before

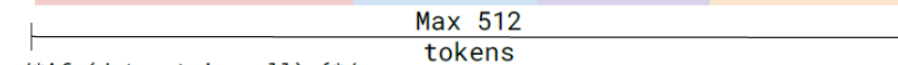
Buggy line

Context after

3

Tokenization Structure

```
public LegendItemCollection getLegendItems() {
    LegendItemCollection result = new LegendItemCollection();
    if (this.plot == null) {
        return result;
    }
    int index = this.plot.getIndexOf(this);
    CategoryDataset dataset = this.plot.getDataset(index);
    if (dataset != null) {
        return result;
    }
    int seriesCount = dataset.getRowCount();
    if (plot.getRowRenderingOrder().equals(SortOrder.ASCENDING)) {
        for (int i = 0; i < seriesCount; i++) {
            if (isSeriesVisibleInLegend(i)) {
                LegendItem item = getLegendItem(index, i);
            }
        }
    }
}
```



Candidate Patches

Ranked Candidate Patches

Ranked Plausible Patches

基于大模型驱动科学解释的缺陷修复

Liked Run - BIP006 ✓

Pipeline

Disliked Run - BIP002 ✗

```
def test_chunked_request_uppercase(self): ...
    Transfer-Encoding: Chunked ...
```

A Prompt (Failing Test)

```
def test_bp_middleware_order(app): ...
    assert order == [1, 2, 3, 4, 5, 6]
```

Step 1.

Hypothesis: The function is not handling the uppercase 'Chunked' header correctly (✓)

Prediction: If I print `headers.get("Transfer-Encoding")`, ... it will be 'Chunked'.

Experiment: `b tornado/http1connection.py:568 ;; c ;; p headers.get("Transfer-Encoding")`

Observation: 'Chunked'

Conclusion: The hypothesis is supported.

B1 Hypothesize

C1 Observe

D1 Conclude

Step 1.

Hypothesis: ... the program is appending the middleware in the wrong order. (⚠)

Prediction: If I print ... at line 656, the order of the middleware will be incorrect.

Experiment: `b sanic/app.py:656 ;; c ;; p ...`

Observation: [The breakpoint ...]

Conclusion: The hypothesis is ...

无法识别出正确的断点。

B2 Hypothesize

C2 Observe

D2 Conclude

Step 2.

Hypothesis: Changing ... to `headers.get("Transfer-Encoding").lower()` ..., the failing test will pass (✓)

Prediction: If I change ... to `headers.get("Transfer-Encoding").lower() == "chunked"` ..., the failing test will pass.

Experiment: `REPLACE(568, ..., "headers.get(\"Transfer-Encoding\").lower() == \"chunked\") AND RUN`

Observation: [The failing test now passes.]

Conclusion: The hypothesis is supported. <DEBUGGING DONE>

Hypothesis: The test is incorrect (⚠)

Prediction: If I print `order` at line 66, the order of the middleware will be correct.

Experiment: `b sanic/app.py:66 ;; c ;; p order`

Observation: [The breakpoint was not covered.]

Conclusion: The hypothesis is undecided due to experiment error.

Generated Fix

Adds `.lower()` to header checking expression.

E Suggest

Generated Fix

Changes all `append` method calls to `appendleft` method calls.

人工编写

对话形式的缺陷自动修复

The following code is buggy.

```
def sieve(max):
    primes = []
    for n in range(2, max):
        if any(n%p for p in primes):
            primes.append(n)
    return primes
```

```
def sieve(max):
    primes = []
    for n in range(2, max):
        if any(n%p for p in primes):
            primes.append(n)
    return primes
original buggy function
```

Please provide a fixed version.

Initial Prompt

I

```
def sieve(max):
    primes = []
    for n in range(2, max+1):
        if not any(n%p for p in primes):
            primes.append(n)
    return primes
```

The fixed version is still not correct.

sample output
S₁

sieve(4) returns [2, 4] but it should return [2, 3]

Please provide a fixed version.

validation feedback

F₁

```
def sieve(max):
    primes = []
    for n in range(2, max):
        if all(n%p for p in primes):
            primes.append(n)
    return primes
```

The fixed version is still not correct.

sample output
S₂

sieve(2) returns [] but it should return [2]

Please provide a fixed version.

validation feedback

F₂

```
def sieve(max):
    primes = []
    for n in range(2, max+1):
        if all(n%p for p in primes):
            primes.append(n)
    return primes
```

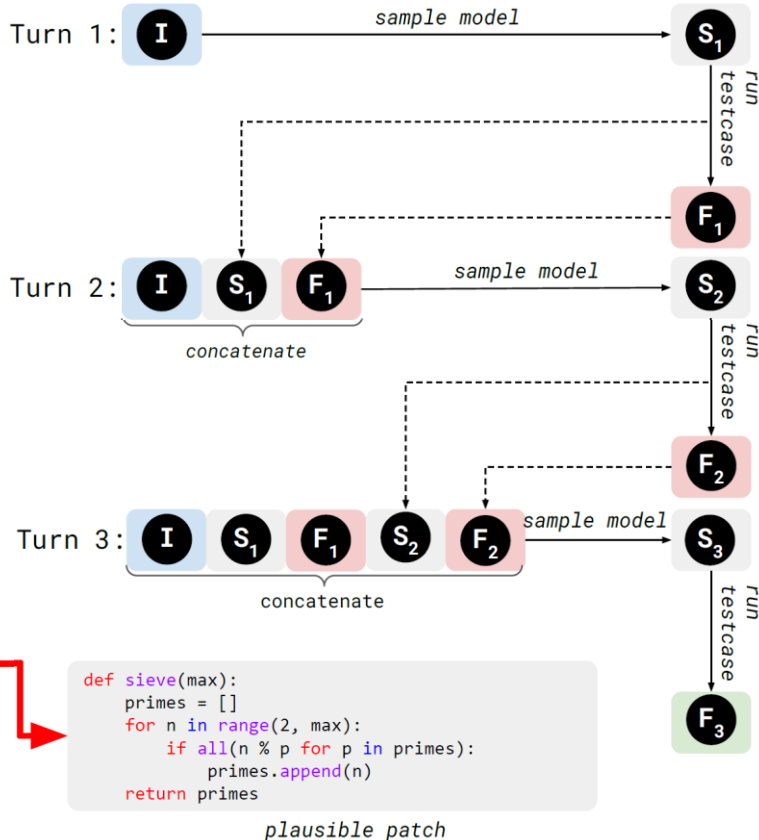
The fixed version is correct!

validation feedback

F₃

sample output
S₃

```
def sieve(max):
    primes = []
    for n in range(2, max):
        if all(n%p for p in primes):
            primes.append(n)
    return primes
plausible patch
```



CONVERSATIONAL AUTOMATED PROGRAM REPAIR

Chunqiu Steven Xia, Lingming Zhang
University of Illinois at Urbana-Champaign
{chunqiu2, lingming}@illinois.edu

QuixBugs基准数据集上的验证结果:

Models	QuixBugs-Java			
	Sampling		Conversational	
	#c/#p	#tries	#c/#p	#tries
CODEGEN-350M	4 / 4	24.2	5 / 5	23.5
CODEGEN-2B	12 / 14	18.8	15 / 16	16.4
CODEGEN-6B	18 / 20	19.8	22 / 22	13.5
CODEGEN-16B	24 / 25	14.5	28 / 29	13.2
Codex	28 / 30	7.2	29 / 30	5.7

Models	QuixBugs-Python			
	Sampling		Conversational	
	#c/#p	#tries	#c/#p	#tries
CODEGEN-350M	7 / 10	20.5	8 / 11	18.4
CODEGEN-2B	22 / 23	16.6	25 / 26	14.3
CODEGEN-6B	22 / 24	14.0	27 / 28	12.1
CODEGEN-16B	29 / 29	5.6	30 / 30	4.8
Codex	29 / 30	4.6	30 / 30	3.8

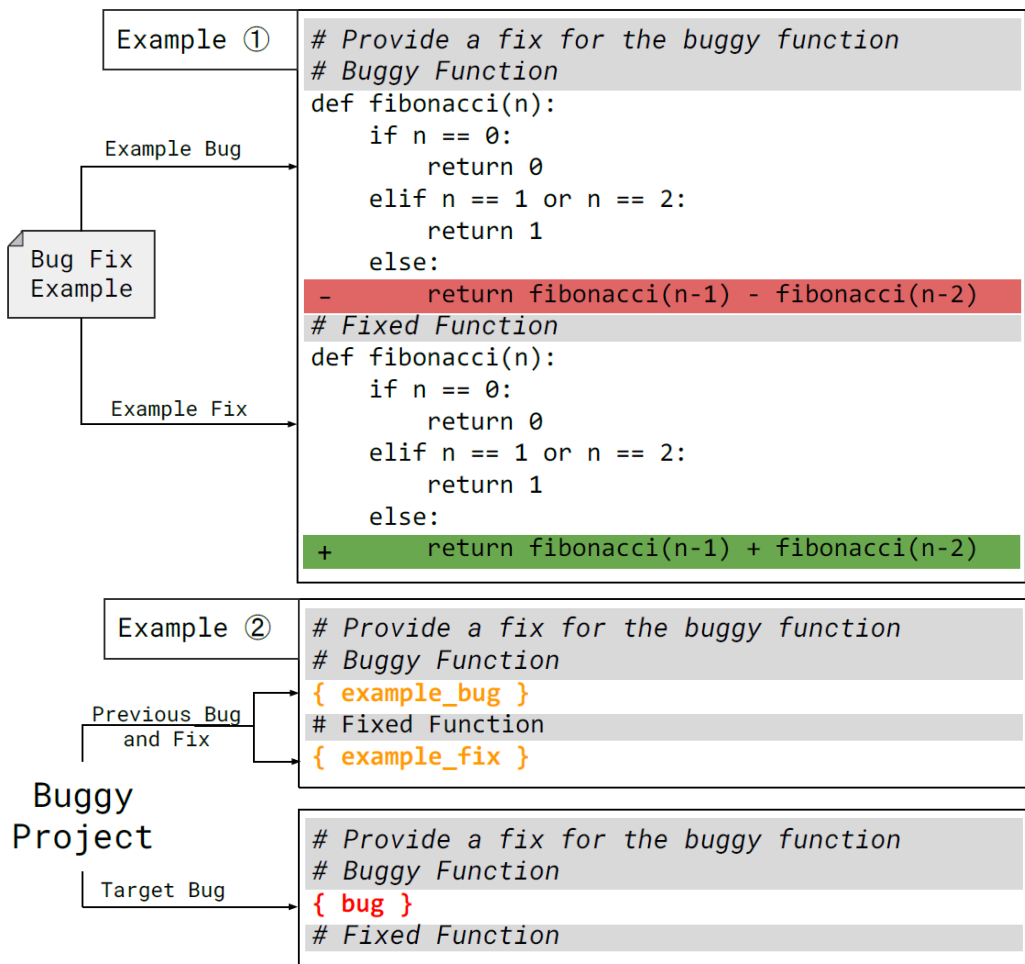
PART 05

总结与展望

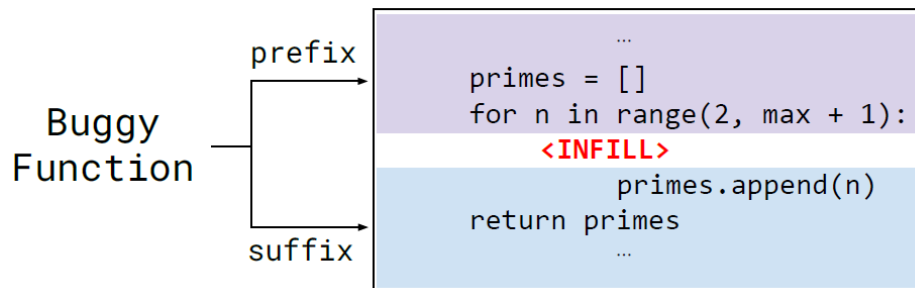


基于大模型的缺陷修复：修复形式

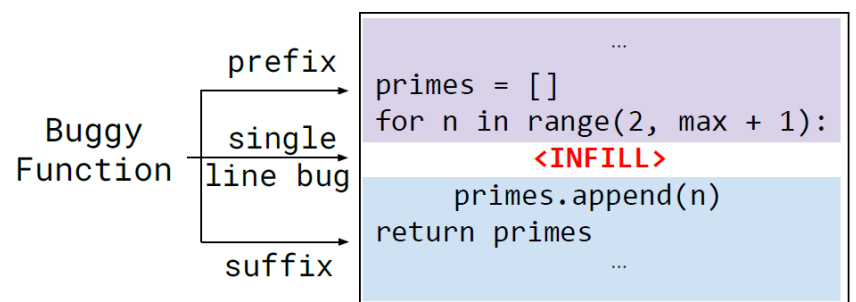
(1) 完整正确功能函数的生成



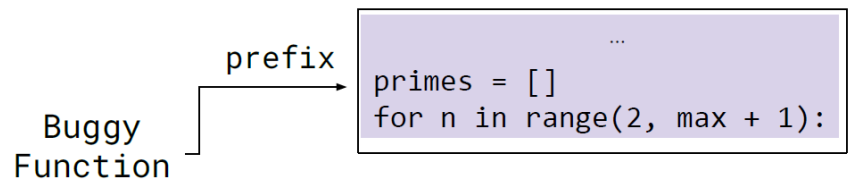
(2) 正确代码的填空



(3) (多个)单行代码的生成



a)

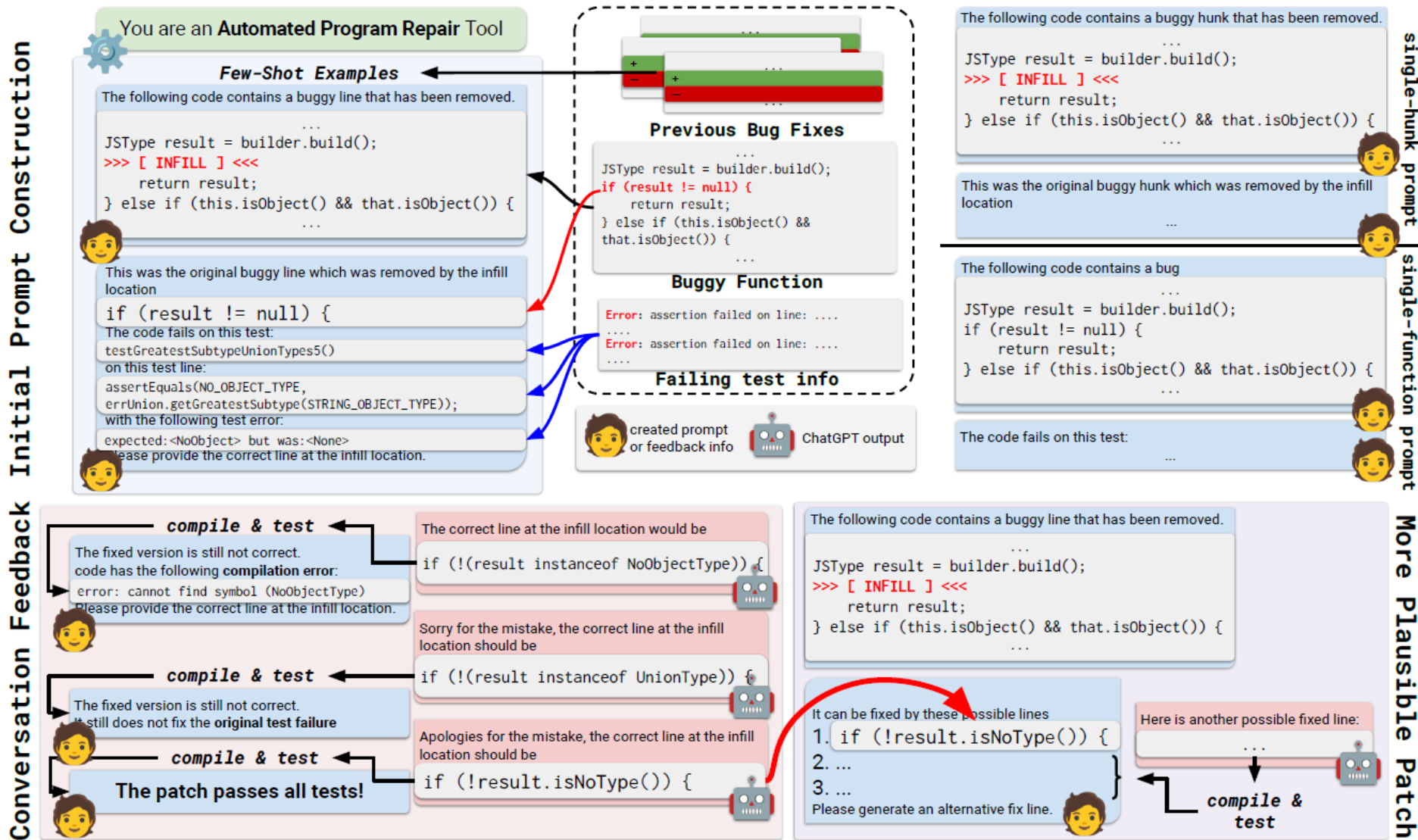


基于大模型的缺陷修复：对话Prompt形式

Keep the Conversation Going:
Fixing 162 out of 337 bugs for \$0.42 each using ChatGPT

Chunqiu Steven Xia
University of Illinois Urbana-Champaign
chunqiu2@illinois.edu

Lingming Zhang
University of Illinois Urbana-Champaign
lingming@illinois.edu



基于大模型的缺陷修复：专业数据的微调

Revisiting the Plastic Surgery Hypothesis via Large Language Models

Chunqiu Steven Xia
University of Illinois
Urbana-Champaign

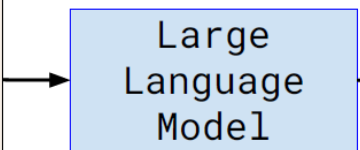
Yifeng Ding
University of Illinois
Urbana-Champaign
linois.edu

Lingming Zhang
University of Illinois
Urbana-Champaign
lingming@illinois.edu

```

    ..masked span token
    CategoryDataset dataset =
    this.plot.getDataset(index);
    if (<SPAN> {
    return result;
    }
    dataset != null
    original buggy code
    ...
    
```

Cloze Input

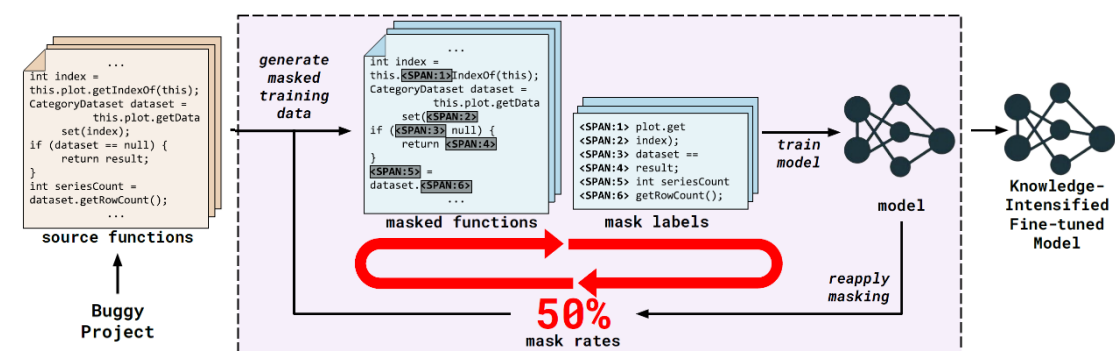


```

    ... generated code
    CategoryDataset dataset =
    this.plot.getDataset(index);
    if (dataset == null) {
    return result;
    }
    ...
    
```

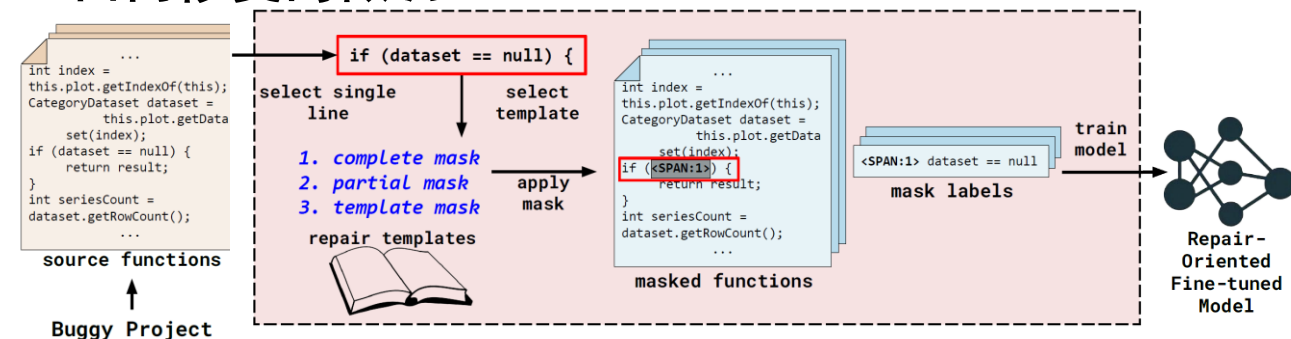
Generated Patch

1. 知识强化的微调



使用知识强化微调来生成更多使用项目特定变量、方法调用和结构的代码。

2. 面向修复的微调



优化下游任务：缺陷修复。

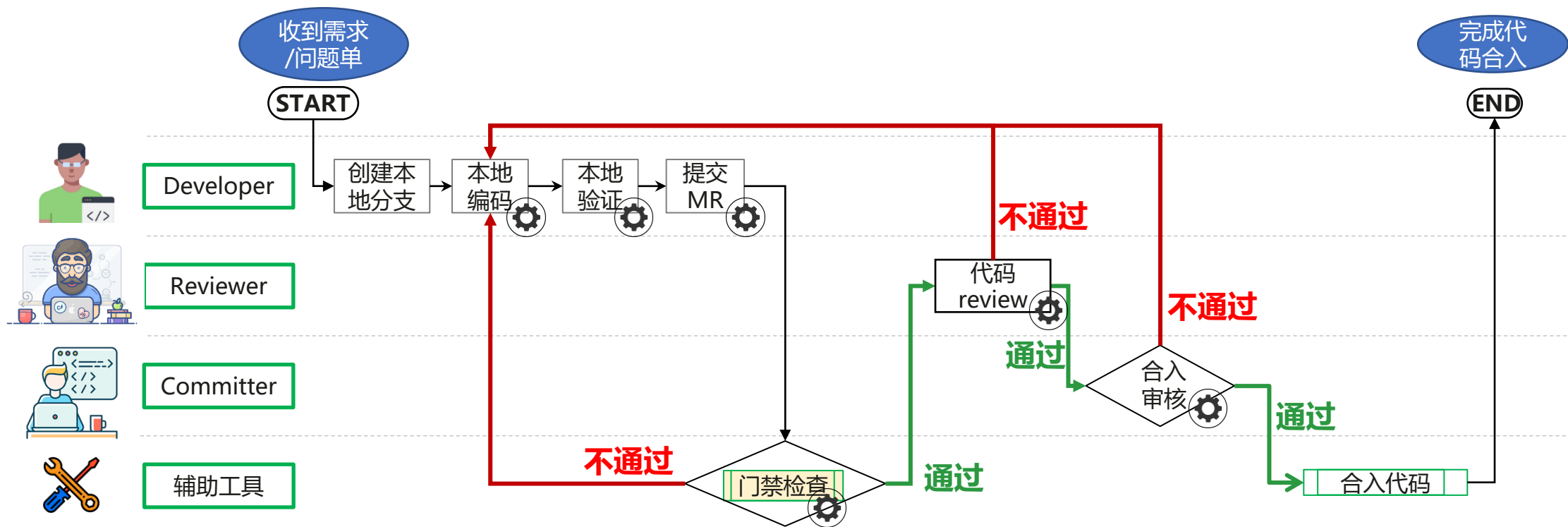
3. 标识符相关的Prompting

利用整形假说原理，补充上下文，去个性化。

基于大模型的缺陷修复：现状与挑战

1. 即使没有微调，大模型呈现出了缺陷修复能力，但**性能有限**。
2. 虽然有了缺陷代码行的提示，大模型能修复一些缺陷，但当前的大模型仍**未能很好地利用缺陷代码行信息**。当给出缺陷代码行时，大模型生成的可编译补丁较少，总体上修复的缺陷也较少。
3. 使用已有**缺陷自动修复工具的数据**进行微调能提高大模型修复缺陷的能力。
4. 需要解决**对缺陷代码的过度依赖问题**：使用缺陷代码数据微调大模型，会过度依赖缺陷代码行。可能导致模型出现偏差，倾向于少的代码变更来修复缺陷，因为微调数据集中大多数缺陷的修复都是少的代码变更。
5. 更**大参数的大模型**在修复缺点问题上展现更好的时间效率和内存效率。

开发人员实际修复缺陷场景



⚙️ 可引入LLM的辅助，LLM对于缺陷修复还有很大的发挥空间。

感谢聆听

