University of Luxembourg

Multilingual. Personalised. Connected.

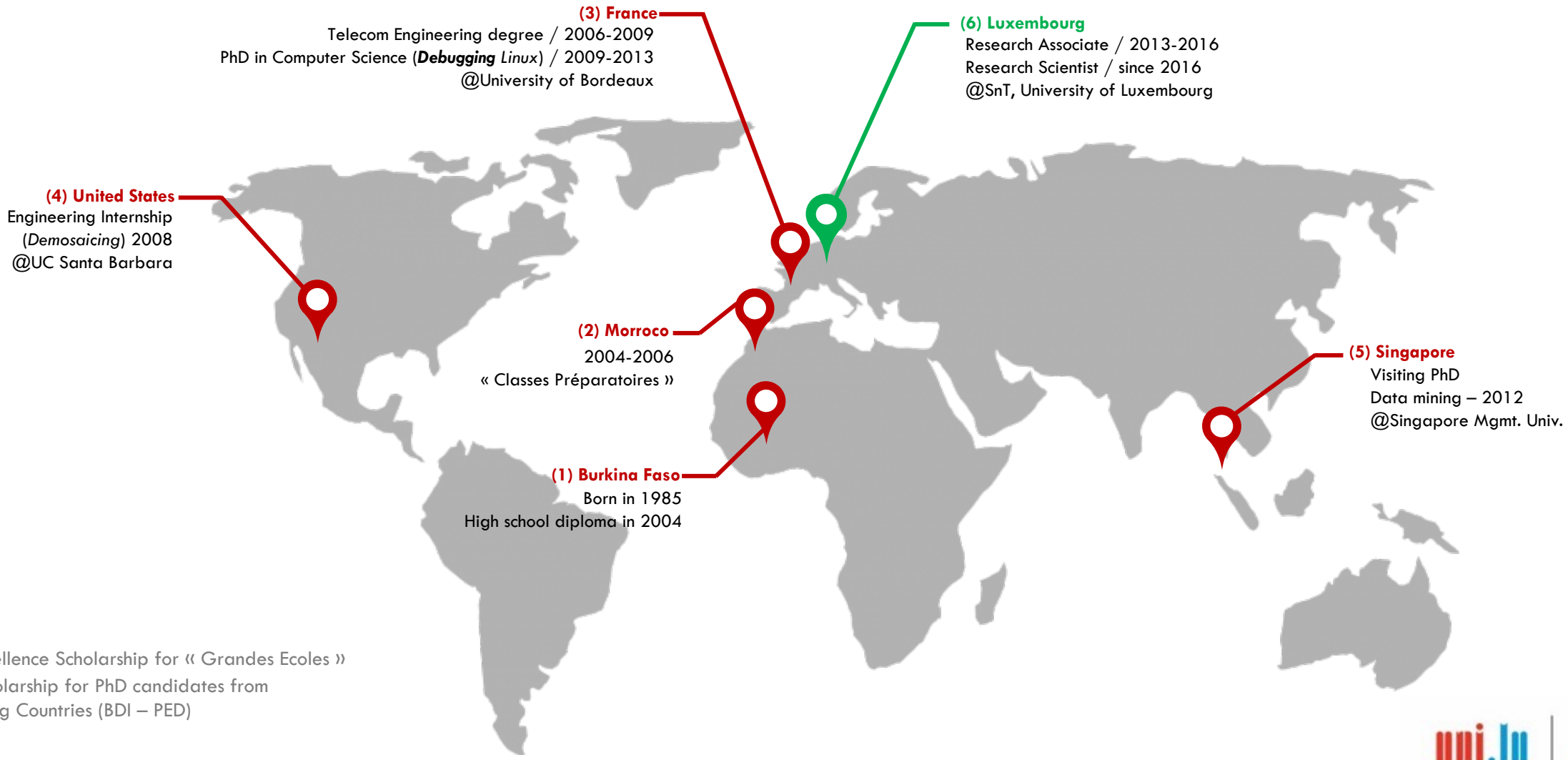
Réparation Automatique des Logiciels: le Rêve et la Fantaisie

GDR GPL, 14 Juin 2021

Prof. Dr. Tegawendé F. BISSYANDE

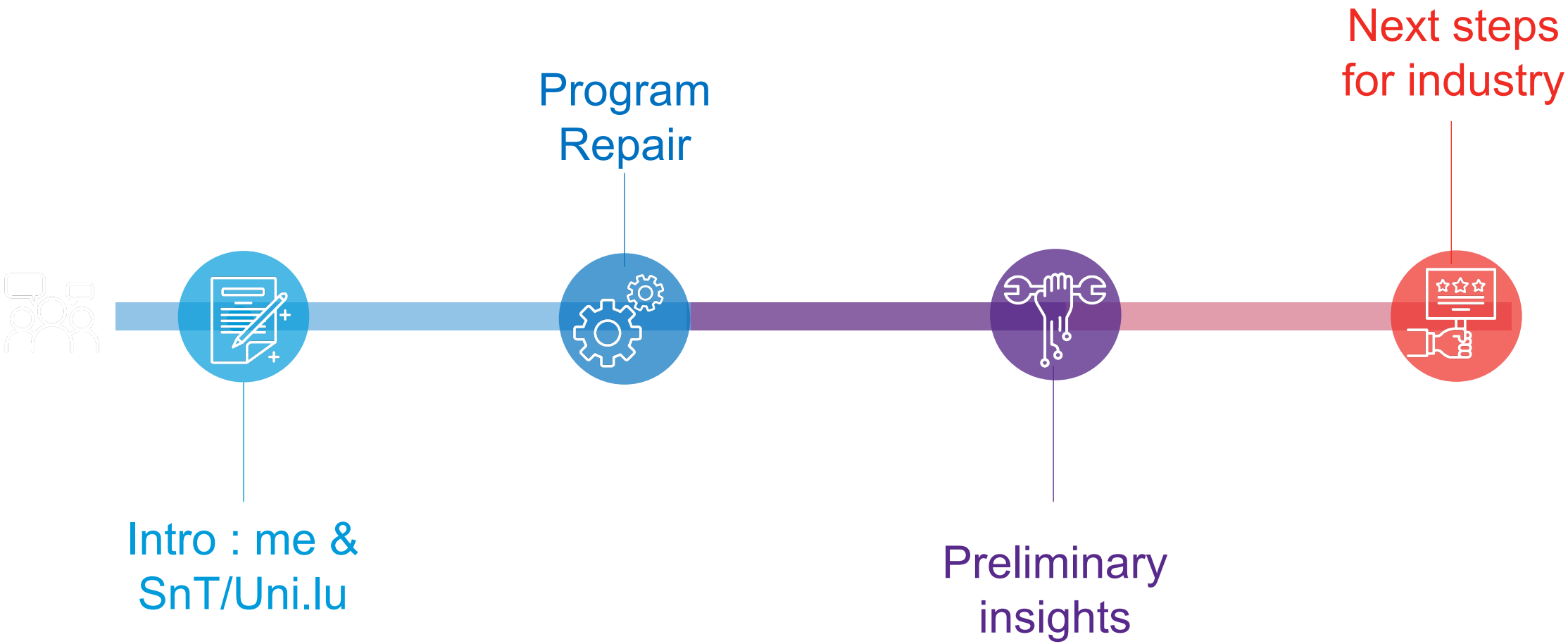
UNIVERSITÉ DU LUXEMBOURG

Background: Education, Mobility, Positions



- Eiffel Excellence Scholarship for « Grandes Ecoles »
- CNRS scholarship for PhD candidates from Developing Countries (BDI – PED)

Agenda


Program Repair
Next steps for industry
Intro : me & SnT/Uni.lu
Preliminary insights

The University of Luxembourg

The University of Luxembourg is a research university with a distinctly **international**, **multilingual** and **interdisciplinary** character.

The University's ambition is to provide the **highest quality research** and teaching in its chosen fields and to generate a positive scientific, educational, social, cultural and societal impact in Luxembourg and the Greater Region.

UNIVERSITY OF LUXEMBOURG

Ranked

12th Young University

worldwide and #1 worldwide for its "international outlook" in the Times Higher Education (THE) World University Rankings 2020

6,714 students

897 PhDs

270 faculty members

129 nationalities

56% international students

The University of Luxembourg

Research Focus Areas

- Computer Science & ICT Security
- European and International Law
- Finance and Financial Innovation
- Education
- Materials Science
- Contemporary and Digital History
- Interdisciplinary theme: Health and Systems Biomedicine
- Interdisciplinary theme: Data Modelling and Simulation

3 Faculties

Faculty of Science, Technology and Medicine

Faculty of Law, Economics and Finance

Faculty of Humanities, Education and Social Sciences

3 Interdisciplinary Centres

SnT

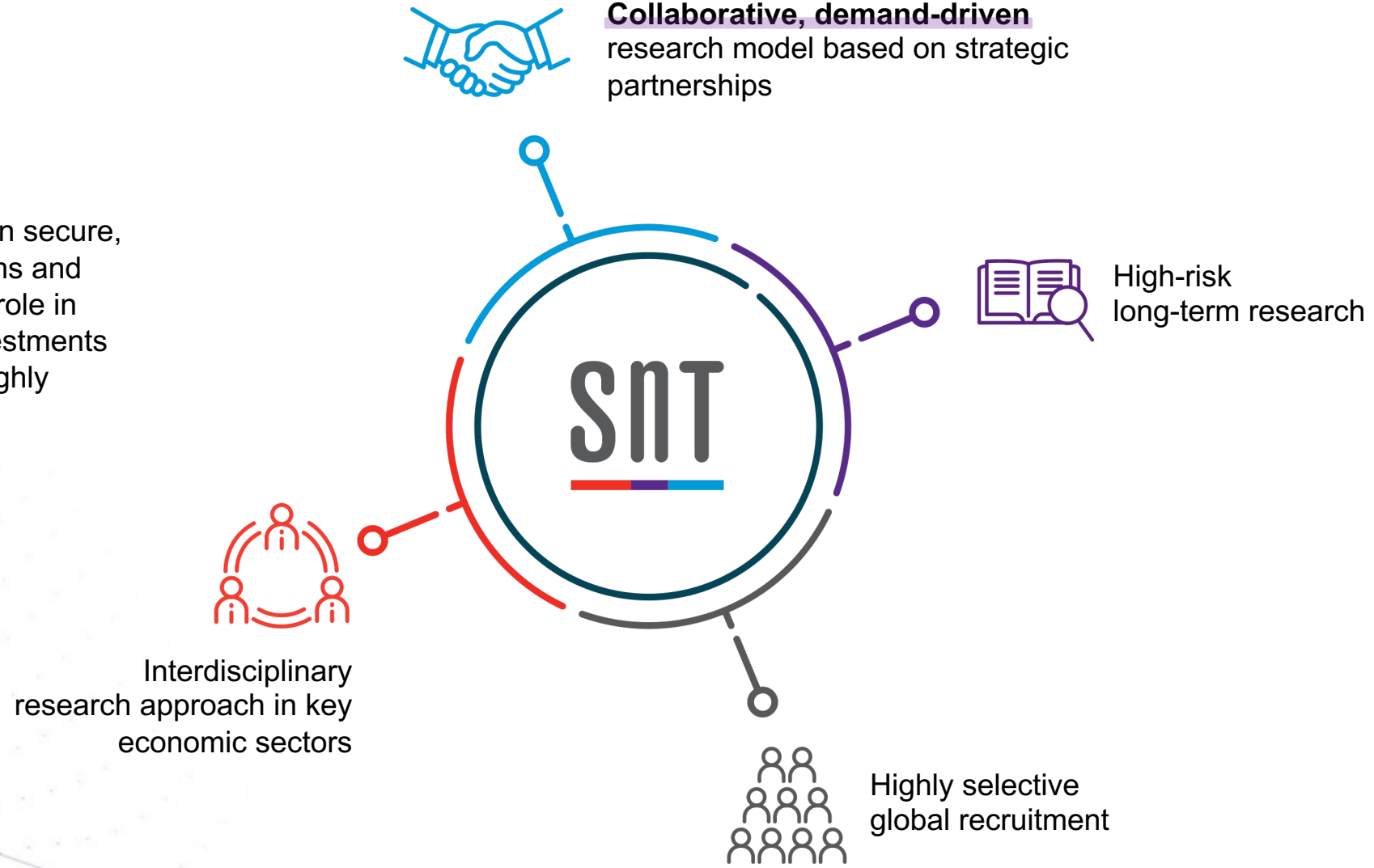
LCSB

C²DH

uni.lu | SnT

Our vision

A leading international **research and innovation centre** in secure, reliable and trustworthy ICT systems and services. We play an instrumental role in Luxembourg by boosting R&D investments leading to economic growth and highly qualified talent.



Key Figures

PEOPLE

385
workforce

66
nationalities

36%
alumni who stay in Luxembourg

PARTNERSHIPS & INNOVATION

47%
of Doctoral candidates on Industrial projects

>50
partners

5M
Partners annual contribution in Euros

5
Spin-offs

Breathing Trust into Business-Critical Software

Most (if not all) modern business-critical solutions rely on software.

e.g., E-payment, blockchain-based solutions, machine-learning based approaches, mobile apps, etc.

Critical Questions:

How to foster the development of Trustworthy software-based solutions?

With:

- Quality of service (crash, bug)
- Limited Security Risks (vulnerabilities)
- Accounting for Compliance (GDPR)

Trustworthy Software Engineering

Software Security

Software Repair

Explainable Software

Software Security

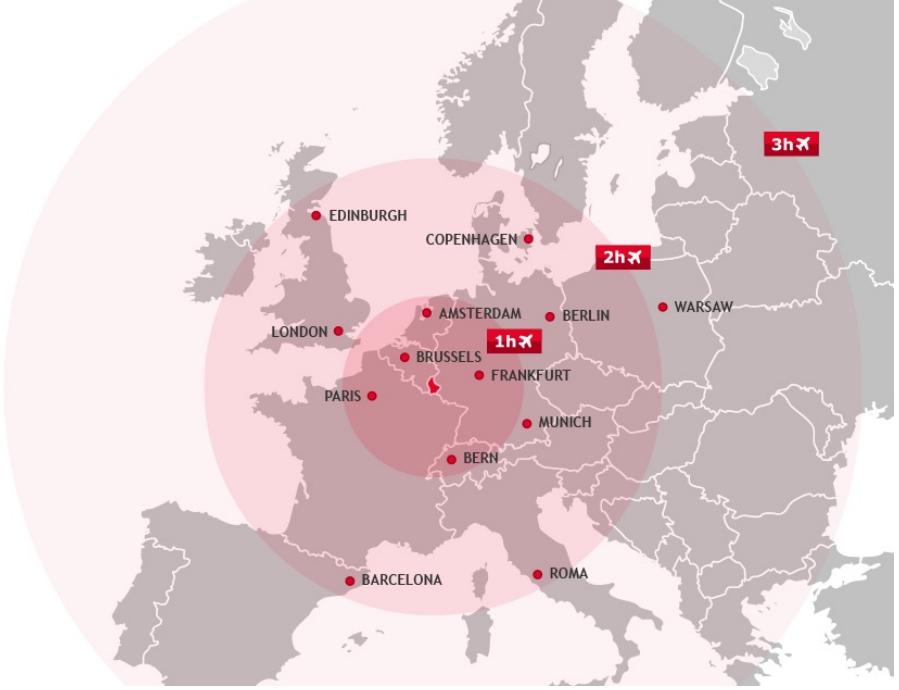
- Vulnerability detection, Data Leaks
- GDPR compliance
- Malware Detection, Piggybacking Detection

Software Repair

- Patch Recommendation
- Automated Program Repair
- Bug Detection
- Vulnerability patching

Explainable Software

- Information Retrieval
- Natural Language Processing
- Time Series Pattern Recognition
- Machine learning



Hiring PhDs and Postdocs Now →

SCAN ME

Kui Lui
(now @NUAA)

Anil Koyuncu
(now @Sabanci)

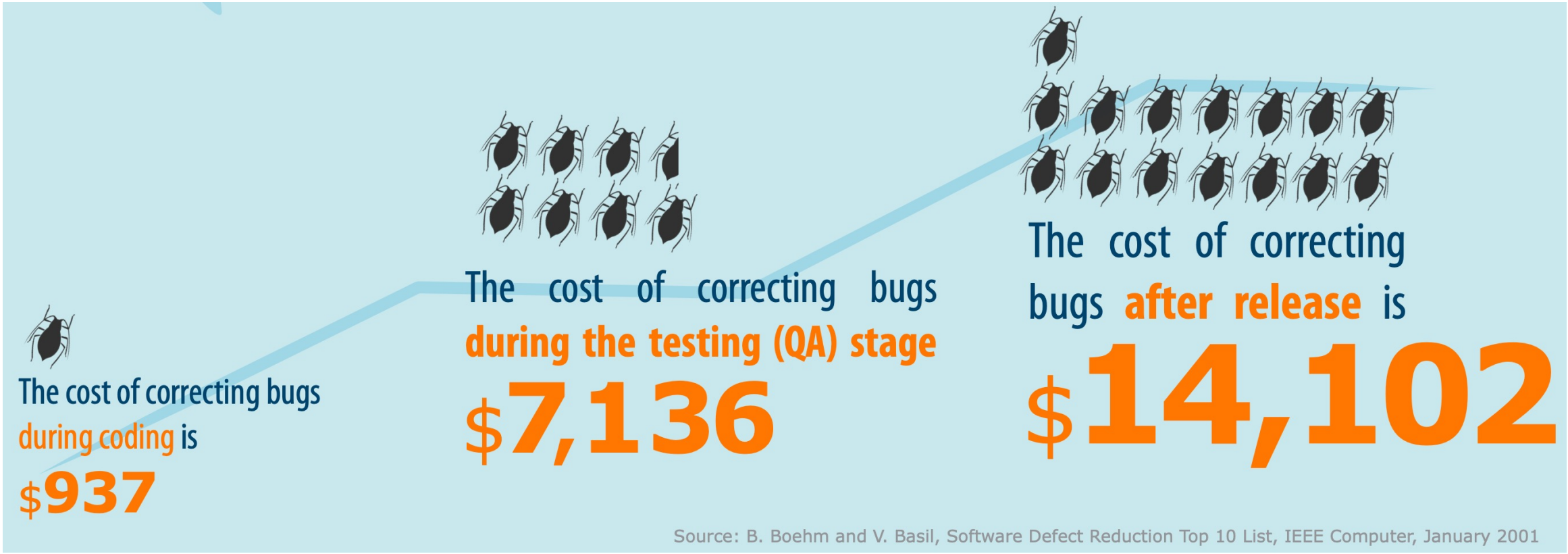
Haoye Tian
(2 years left)

Kisub Kim
(2 weeks left)

Program Repair Task Force:
*Those who **did** the work!*


SNT
Program Repair
UNIVERSITY OF LUXEMBOURG

Fixing Bugs is Expensive



3. B. Boehm and V. Basil, Software Defect Reduction Top 10 List, IEEE Computer, January 2001.

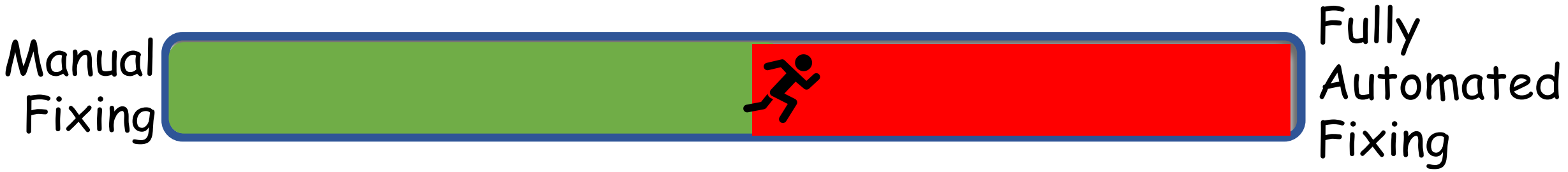
Let’s Recall Traditional Bug Fixing

Detection

Localization

Generation

From Manual to Automated Fixing



Test Automation

Automated Bug/Fault Localization

Static/Dynamic Analysis

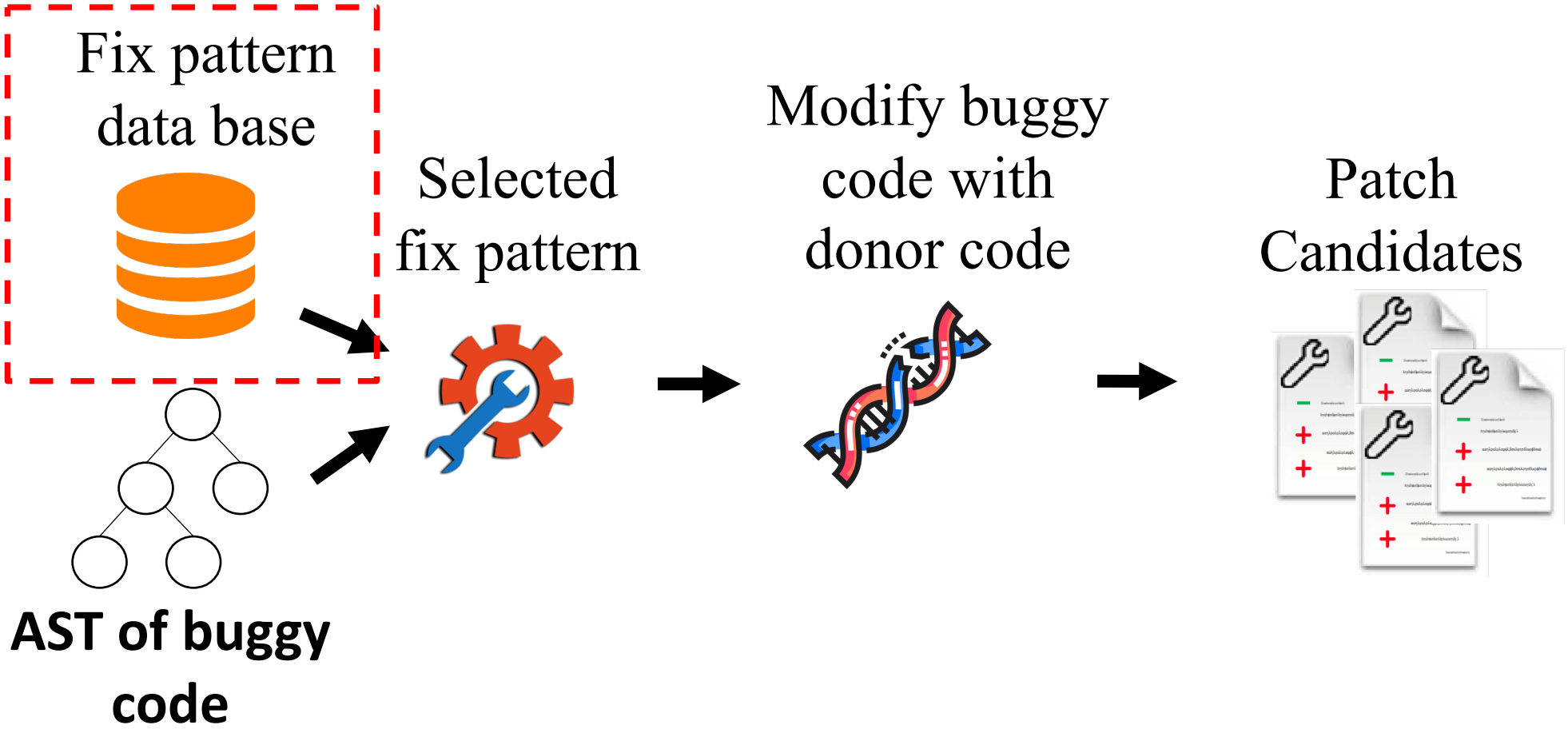
Program Repair

Automated Program Repair (APR)

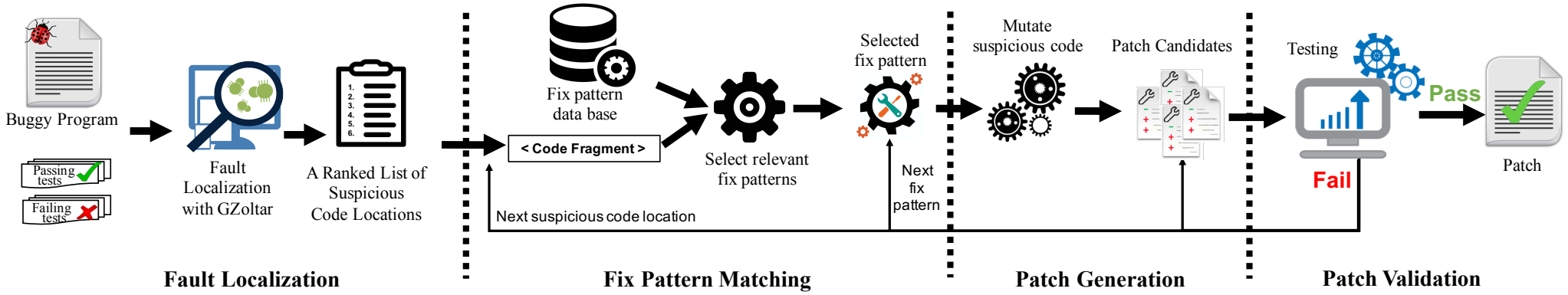
- Heuristic-based program repair,
  e.g., GenProg, SimFix, CapGen, AVATAR.
- Constraint-based program repair,
  e.g., Nopol, ACS, and Cardumen.
- Learning-aided program repair,
  e.g., Deepfix, Prophet, and Genesis.

4. Claire Le Goues, Michael Pradel, and Abhik Roychoudhury. Automated program repair. *Communications of the ACM*, 2019.

Template-based APR



Typical generate-and-validate pipeline / "Template-based"



- In practice, when can we identify the **fault location** ?
- In practice, where should we get the **patterns** ?
- In practice, is there a test suite available to **validate** the generated patch?

Research axes

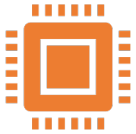
How do we localize faults in practice ?

Can we ignore the assumptions of exhaustive test suites ?

Can we be efficient in the generation of patches?

Can we predict patch correctness beyond tests?

More on this talk



[1] Tian et al. *Evaluating Representation Learning of Code Changes for Predicting Patch Correctness in Program Repair* – ***ASE 2020***

[2] Liu et al. *On the Efficiency of Test Suite based Program Repair: A Systematic Assessment of 16 Automated Repair Systems for Java Programs* – ***ICSE 2020***

[2] Liu et al. *You cannot Fix what you cannot find! An Investigation of Fault Localization Bias in Benchmarking Automated Program Repair Systems* – ***ICST 2019***

[3] Koyuncu et al. *iFixR – Bug Report driven Program Repair* – **FSE 2019**

[4] Kim et al. *FaCoY – A Code-to-Code Search Engine* – **ICSE 2018**

[5] Liu et al. *LSRepair: Live Search of Fix Ingredients for Automated Program Repair* – **APSEC 2018**

SNT

Preliminary Insights

UNIVERSITY OF LUXEMBOURG

Repair Tool Performance Assessment

TABLE I

TABLE EXCERPTED FROM [33] WITH THE CAPTION "*Correct patches generated by different techniques*".

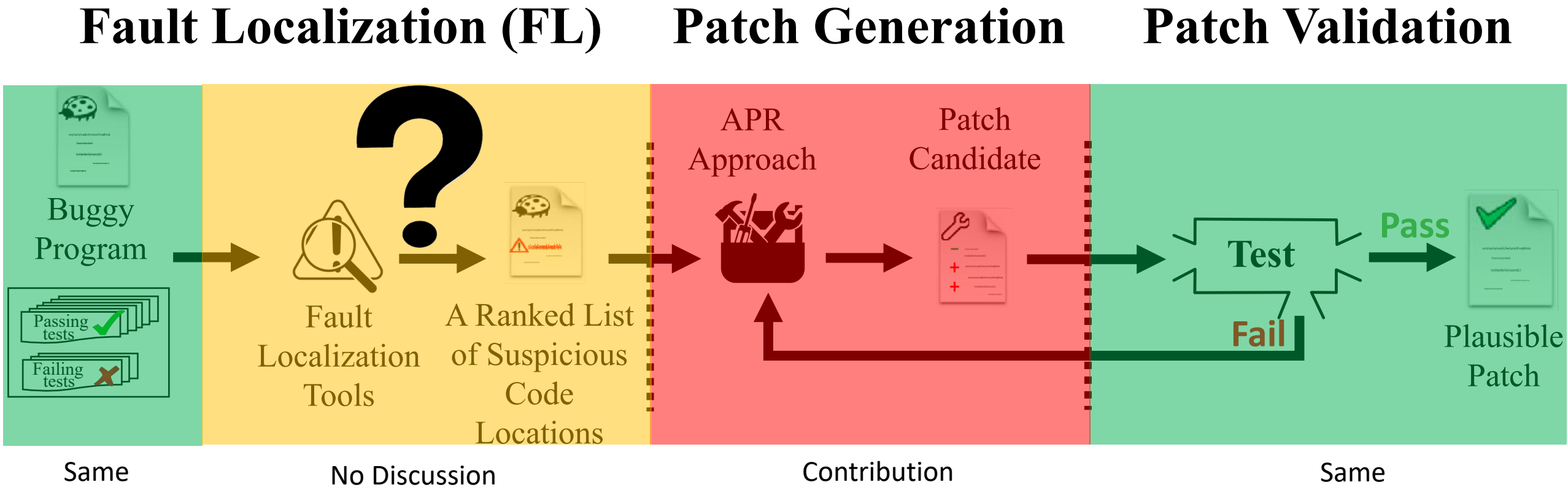
Proj.	SimFix	jGP	jKali	Nopol	ACS	HDR	ssFix	ELIXIR	JAID
Chart	4	0	0	1	2	-(2)	3	4	2(4)
Closure	6	0	0	0	0	-(7)	2	0	5(9)
Math	14	5	1	1	12	-(7)	10	12	1/(7)
Lang	9	0	0	3	3	-(6)	5	8	1/(5)
Time	1	0	0	0	1	-(1)	0	2	0/(0)
Total	34	5	1	5	18	13(23)	20	26	9/(25)

*The numbers in parenthesis(#) denote the number of bugs fixed by APR tools but ignoring the patch ranking.

How could such a comparison be unfair?

What are the assumptions of fault localization?

Basic Repair process



Limited discussion on the impact of fault localization on APR tool performance.

Variabilities in FL integration within the Tools

FAULT LOCALIZATION (FL) TECHNIQUES INTEGRATED INTO STATE-OF-THE-ART APR TOOLS.

	jGP	jKali	jMutRepair	HDRepair	Nopol	ACS	ELIXIR	JAID	ssFix	CapGen	SketchFix	FixMiner	LSRepair	SimFix
FL testing framework	GZoltar	GZoltar	GZoltar	?	GZoltar	GZoltar	?	?	GZoltar	GZoltar	?	GZoltar	GZoltar	GZoltar
Framework version	0.1.1	0.1.1	0.1.1	?	0.0.10	0.1.1	?	?	0.1.1	0.1.1	?	0.1.1	0.1.1	1.6.0
FL ranking metric	Ochiai	Ochiai	Ochiai	?	Ochiai	Ochiai	Ochiai	?	?	Ochiai	Ochiai	Ochiai	Ochiai	Ochiai
Granularity of fault locality	line	line	line	line	line	line	line	line	line	line	line	line	method	line
Supplementary information	∅	∅	∅	Faulty method is known	∅	Predicate switching [53]	?	?	Statements in crashed stack trace	?	?	∅	∅	Test Case Purification [54]

* The unspecified/unconfirmed information of an APR tools is marked with '?'. If an APR tool does not use any supplementary information for FL, the corresponding table cell is marked with '∅'.

1. APR tools may add some adaptations to the classical FL
2. Unknown to what extent performance is »just « due to better FL
3. Missing FL details for replication/reproduction

Repair Tool Performance Assessment

TABLE I

TABLE EXCERPTED FROM [33] WITH THE CAPTION "*Correct patches generated by different techniques*".

Proj.	SimFix	jGP	jKali	Nopol	ACS	HDR	ssFix	ELIXIR	JAID
Chart	4	0	0	1	2	-(2)	3	4	2(4)
Closure	6	0	0			(7)	2	0	5(9)
Math	14	5	1	1	12	(7)		12	1/(7)
Lang	9	0	0		3	-(6)		8	1/(5)
Time	1	0	0	0	1	-(1)	0	2	0/(0)
Total	34	5	1	5	18	13(23)	20	26	9/(25)

BIASED

*The numbers in parenthesis(#) denote the number of bugs fixed by APR tools but ignoring the patch ranking.

1. **If the testing frameworks are different**
2. **If the localization assumptions are different**

> Localizability of benchmark bugs

Localizability:

File Level

Method Level

Line Level

```
--- a/src/org/jfree/data/time/Week.java
+++ b/src/org/jfree/data/time/Week.java

@@ -173, 1 +173, 1 @@
    public Week(Date time, TimeZone zone) {
            // defer argument checking...
-           this(time, RegularTimePeriod.DEFAULT_TIME_ZONE,
                    Locale.getDefault());
+           this(time, zone, Locale.getDefault);
    }
```

> Localizability of Defects4J bugs

NUMBER OF BUGS LOCALIZED* WITH OCHIAI/GZOLTAR.

Project	# Bugs	File		Method		Line	
		GZ_1	GZ_2	GZ_1	GZ_2	GZ_1	GZ_2
Chart	26	25	25	22	24	22	24
Closure	133	113	128	78	96	78	95
Lang	65	54	64	32	59	29	57
Math	106	101	105	92	100	91	100
Mockito	38	25	26	22	24	21	23
Time	27	26	26	22	22	22	22
Total	395	344	374	268	325	263	321

*A bug is counted as localized as long any of the faulty locations appear in the ranked list of suspicious locations reported by the FL tool. GZ_1 and GZ_2 indicate GZoltar 0.1.1 and 1.6.0, respectively. The same abbreviations are used for GZoltar versions in the following tables. The column GZ_1 of "Line" is highlighted since it is the most common configuration in APR systems.

One third of bugs in the Defects4J dataset cannot be localized at line level by the commonly used automated fault localization tool.

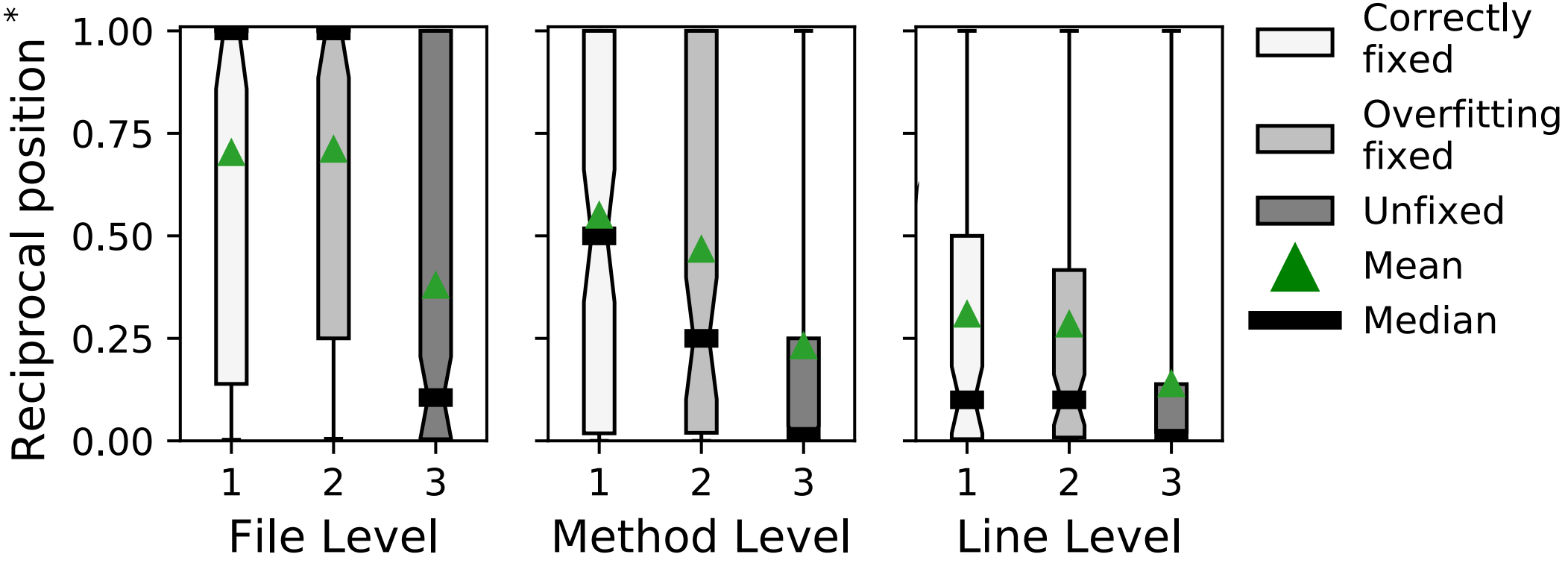
> Localizability of Defects4J bugs

NUMBER OF BUGS LOCALIZED AT TOP-1 AND TOP-10.

Ranking Metric	GZ[1]			GZ[2]		
	File	Method	Line	File	Method	Line
Top-1 Position						
Tarantula	171	101	45	169	106	35
Ochiai	173	102	45	172	111	38
DStar2	173	102	45	175	114	40
Barinel	171	101	45	169	107	36
Opt2	175	97	39	179	115	39
Muse	170	98	40	178	118	41
Jaccard	173	102	45	171	112	39
Top-10 Position						
Tarantula	240	180	135	242	189	144
Ochiai	244	184	140	242	191	145
DStar2	245	184	139	242	190	142
Barinel	240	180	135	242	190	145
Opt2	237	168	128	239	184	135
Muse	234	169	129	239	186	140
Jaccard	245	184	139	241	188	142

Only a fraction of bugs can be localized with high positions in the ranking list of suspicious positions.

> Impact of Effective Localization Ranking



APR tools are prone to correctly fix the subset of Defects4J bugs that can be accurately localized.

> kPAR: A baseline for the research community

kPAR: Java implementation of PAR (Kim et al. ICSE 2013)
+ Gzoltar-0.1.1 + Ochiai.

Normal FL: It relies on the ranked list of suspicious code locations reported by a given FL tool.

File Assumption: It assumes that the faulty code files are known.

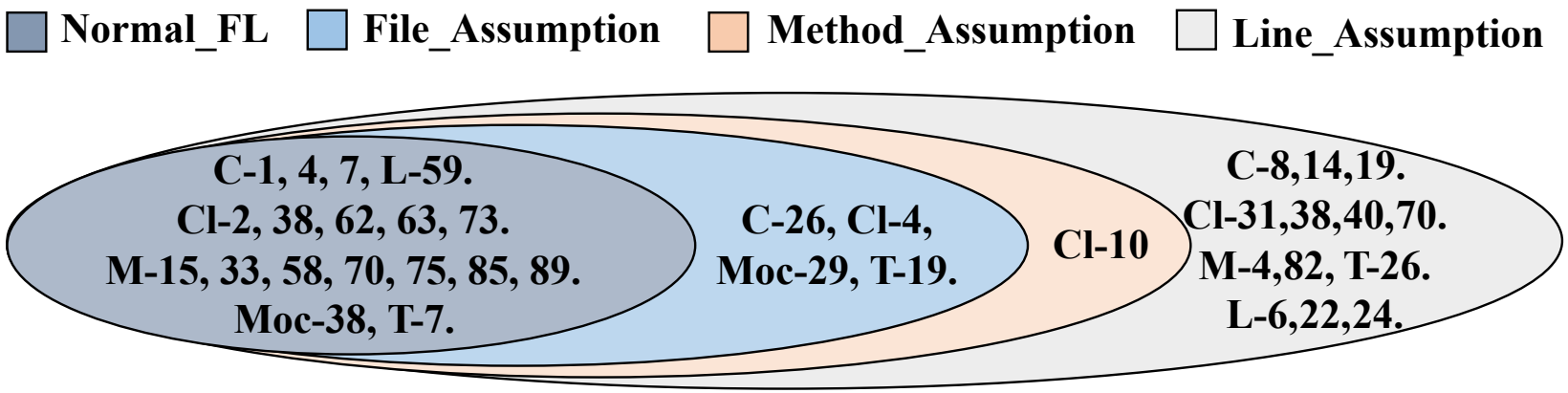
Method Assumption: It assumes that the faulty methods are known.

Line Assumption: It assumes that the faulty code lines are known. No fault localization is then used.

> kPAR: comparison

Number of Defects4J bugs fixed by kPAR with four FL configurations.

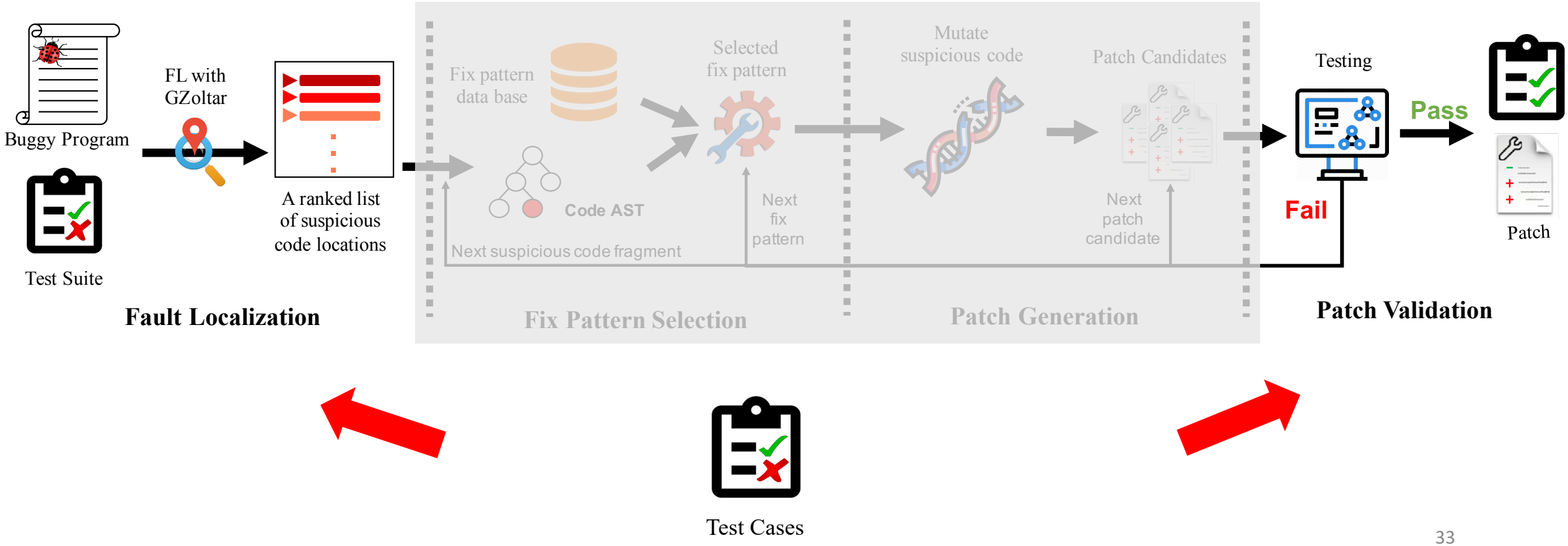
FL Conf.	Chart (C)	Closure (Cl)	Lang (L)	Math (M)	Mockito (Moc)	Time (T)	Total
Normal FL	3/10	5/9	1/8	7/18	1/2	1/2	**18/49**
File Assumption	4/7	6/13	1/8	7/15	2/2	2/3	**22/48**
Method Assumption	4/6	7/16	1/7	7/15	2/2	2/3	**23/49**
Line Assumption	7/8	11/16	4/9	9/16	2/2	3/4	**36/55**



With better fault localization results, kPAR can correctly fix more bugs.

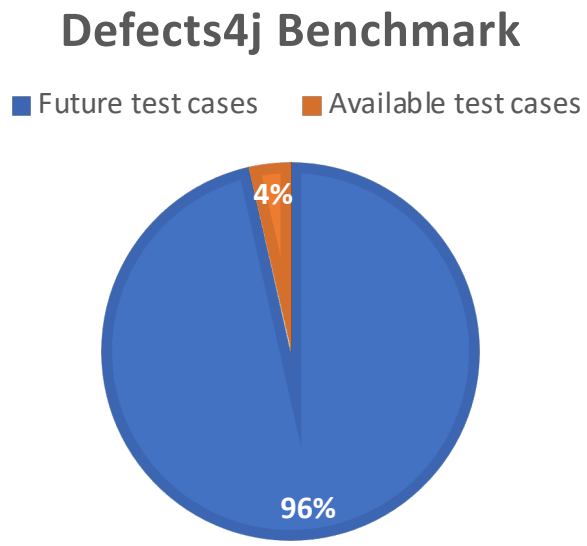
What about Test Suites?

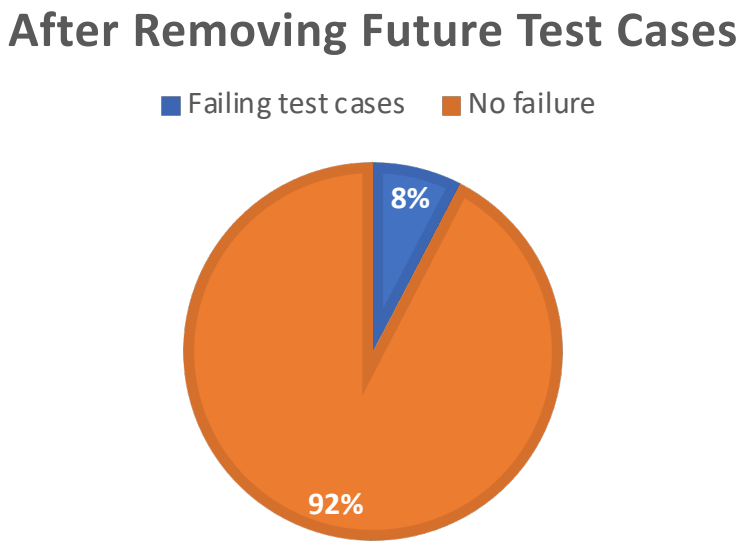
Assumption of Complete/Reliable Test suite



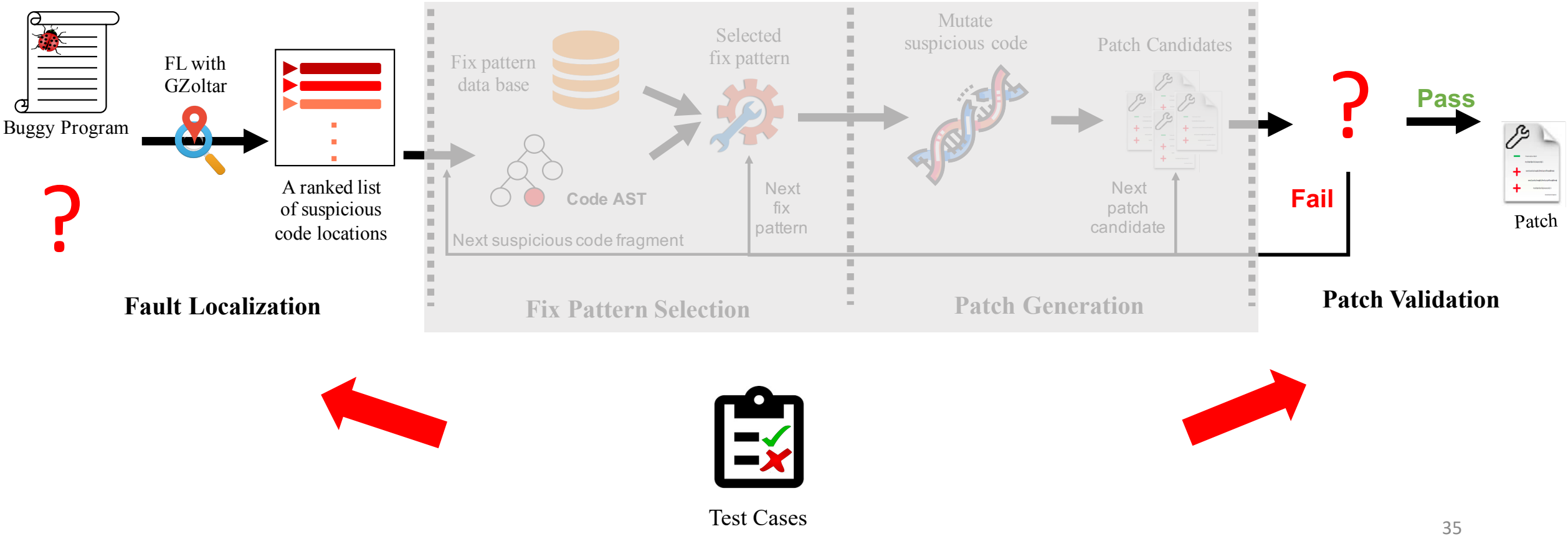
Test suite

A relevant test case reproducing the bug may not be readily available, when a bug report is submitted to the issue tracking system.

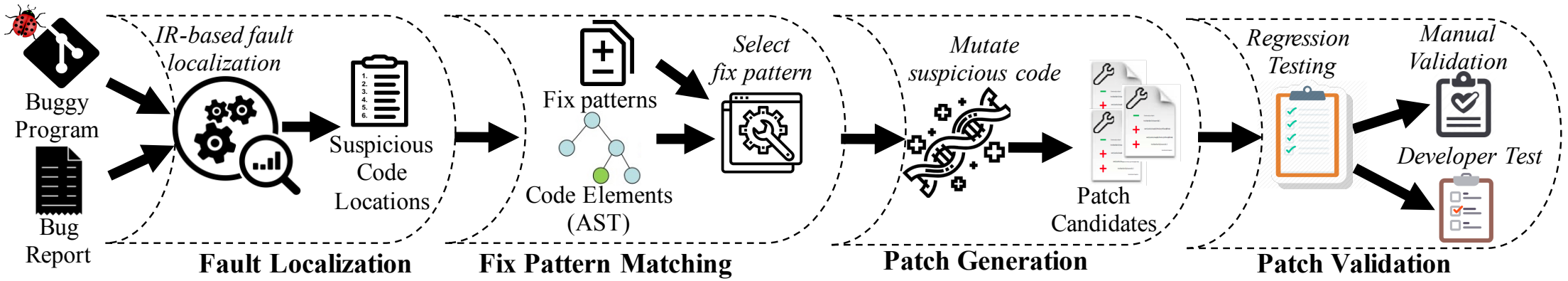




How to repair without future information?

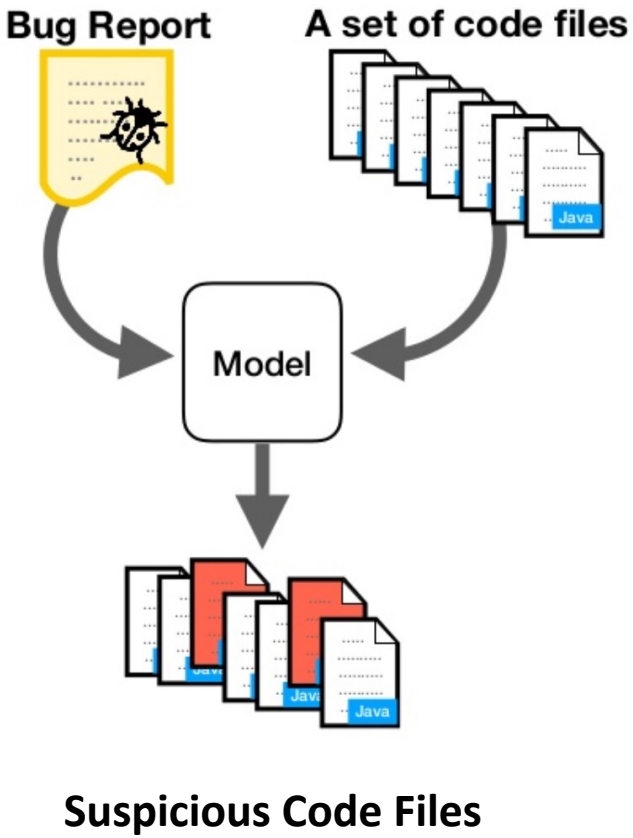


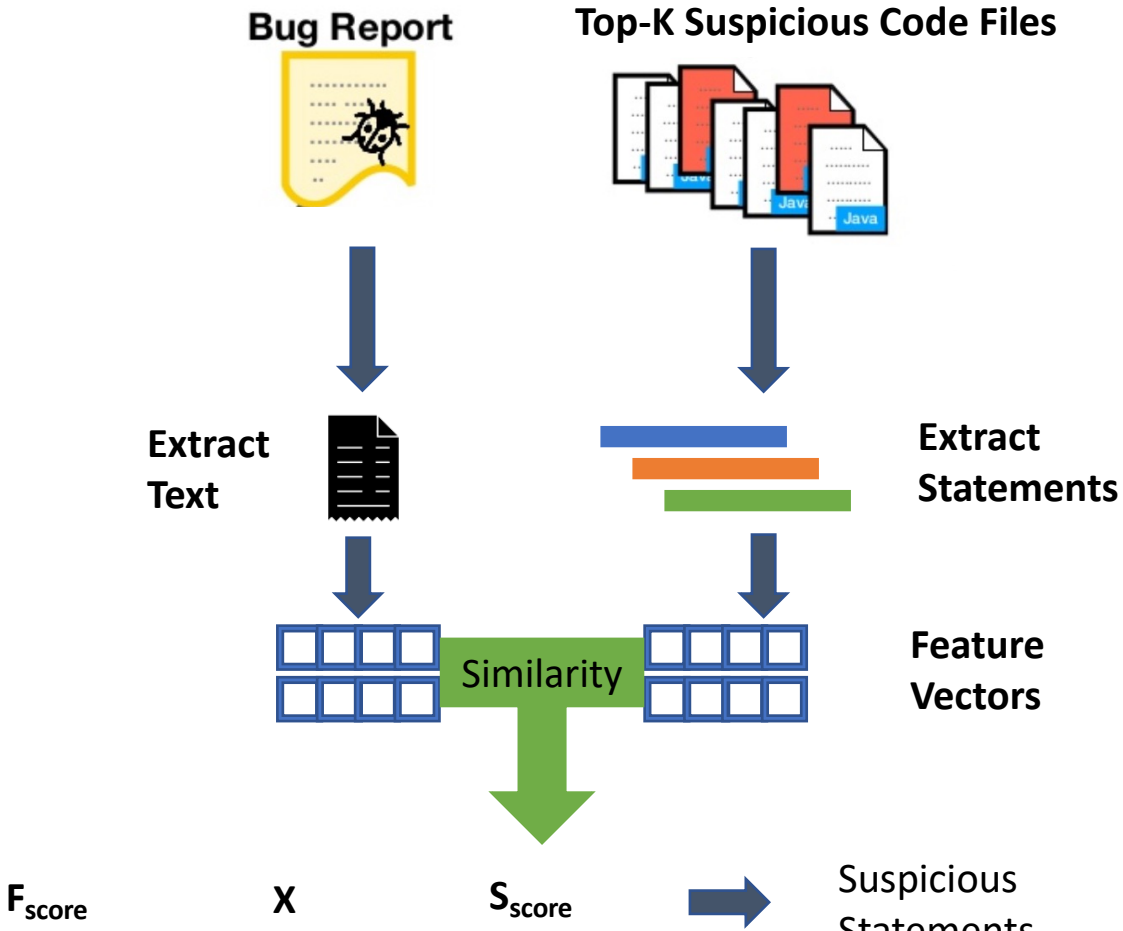
iFixR: Bug Report driven Program Repair



iFixR - Fault Localization

Statement level Information Retrieval Fault Localization**(IRFL)**





iFixR - Fix Pattern-based Patch Generation

All fix patterns in the APR community

Pattern Description	Used by	Pattern Description	Used by
Insert Cast Checker	Genesis	Mutate Literal Expression	SimFix
Insert Null Pointer Checker	NPEFix	Mutate Method Invocation	ELIXIR
Insert Range Checker	SOFix	Mutate Operator	jMutRepair
Insert Missed Statement	HDRepair	Mutate Return Statement	SketchFix
Mutate Conditional Expression	ssFix	Mutate Variable	CapGen
Mutate Data Type	AVATAR	Move Statement(s)	PAR
Remove Statement(s)	FixMiner		

```
+  if (exp instanceof T) {
       ...(T) exp...; ......
+  }
```

"Insert Cast Checker" fix pattern

iFixR - Patch Validation

A **patch ordering strategy** to recommend patches with priority

Heuristics to **re-prioritize** the patch candidates

1. Minimal changes
2. Fault localization suspiciousness
3. Affected code elements

> Research Questions

securityandtrust.lu

RQ1: [Fault localization] : To what extent does IR-based fault localization provide reliable results for an APR scenario?

RQ2: [Overfitting] : To what extent does IR-based fault localization point to locations that are less subject to overfitting?

RQ3: [Patch ordering] : What is the effectiveness of MIMIC's patch ordering strategy?

> IR-based FL vs Spectrum-based FL

Table 5: Fault localization results: IRFL (IR-based) vs. SFL (Spectrum-based) on Defects4J (Math and Lang) bugs.

(171 bugs)		Top-1	Top-10	Top-50	Top-100	Top-200	All
IRFL		25	72	102	117	121	139
SFL	GZ_{v1}	26	75	106	110	114	120
	GZ_{v2}	23	79	119	135	150	156

† GZ_{v1} and GZ_{v2} refer to GZoltar 0.1.1 and 1.6.0 respectively, which are widely used in APR systems for Java programs.

Fine-grained IR-based Fault Localization (IRFL) can be as accurate as Spectrum-based fault localization
+ it does not require test cases

> Overfitting

IRFL vs. SFL impacts on the number of generated genuine/plausible patches for Defects4J bugs.

	Lang	Math	Total
IRFL Top-1	1/4	3/4	4/8
SFL Top-1	1/4	**6/8**	7/12
IRFL Top-5	**3**/6	7/14	10/20
SFL Top-5	2/**7**	**11/17**	13/24
IRFL Top-10	**4/9**	9/17	13/26
SFL Top-10	4/11	**16/27**	20/38
IRFL Top-20	**7/12**	9/18	16/30
SFL Top-20	4/11	**18/30**	22/41
IRFL Top-50	**7/15**	10/22	17/37
SFL Top-50	4/13	**19/34**	23/47
IRFL Top-100	**8/18**	10/23	18/41
SFL Top-100	5/14	**19/36**	24/50
IRFL All	**11/19**	10/25	21/44
SFL All	5/14	**19/36**	24/50

* We indicate x/y numbers of patches: x is the number of bugs for which a *genuine* patch is generated; y is the number of bugs for which a *plausible* patch is generated.

Dissection of reasons why patches are plausible* but not genuine.

	Localization Error	Pattern Prioritization	Lack of Fix ingredients
w/ IRFL	6	1	16
w/ SFL	15	1	10

*A plausible patch passes all test cases, but may not be semantically equivalent to developer patch (i.e., genuine). We consider a plausible patch to be overfitted to the test suite

IR-based fault localization lead less to overfitted patches than the code locations suggested by **Spectrum-based** fault localization

> Patch Ordering

Table 9: Overall performance of iFixR for patch recommendation on the Defects4J benchmark.

Recommendation rank	Top-1	Top-5	Top-10	Top-20	All
without patch re-prioritization	3/3	4/5	6/10	6/10	13/27
with patch re-prioritization	3/4	8/13	9/14	10/15	13/27

* x/y: x is the number of bugs for which a *correct* patch is generated; y is the number of bugs for which a *plausible* patch is generated.

Ordering works!

> iFixR vs the State-of-the-Art

securityandtrust.lu

Table 10: iFixR vs state-of-the-art APR tools.

APR tool	Lang*	Math*	Total*
jGenProg [58]	0/0	5/18	5/18
jKali [58]	0/0	1/14	1/14
jMutRepair [58]	0/1	2/11	2/12
HDRepair [35]	2/6	4/7	6/13
Nopol [92]	3/7	1/21	4/28
ACS [91]	3/4	12/16	15/20
ELIXIR [72]	8/12	12/19	20/31
JAID [12]	1/8	1/8	2/16
ssFix [89]	5/12	10/26	15/38
CapGen [83]	5/5	12/16	17/21
SketchFix [18]	3/4	7/8	10/12
FixMiner [30]	2/3	12/14	14/17
LSRepair [43]	8/14	7/14	15/28
SimFix [19]	9/13	**14/26**	**23**/39
kPAR [47]	1/8	7/18	8/26
AVATAR [48]	5/11	6/13	11/24
iFixR$_{opt}$	**11/19**	10/25	21/**44**
iFixR$_{all}$	6/11	7/16	13/27
iFixR$_{top5}$	3/7	5/6	8/13

* x/y: x is the number of bugs for which a *correct* patch is generated; y is the number of bugs for which a *plausible* patch is generated.
iFixR$_{opt}$: the version of iFixR where available test cases are relevant to the bugs.
iFixR$_{all}$: all recommended patches are considered.
iFixR$_{top5}$: only top 5 recommended patches are considered.

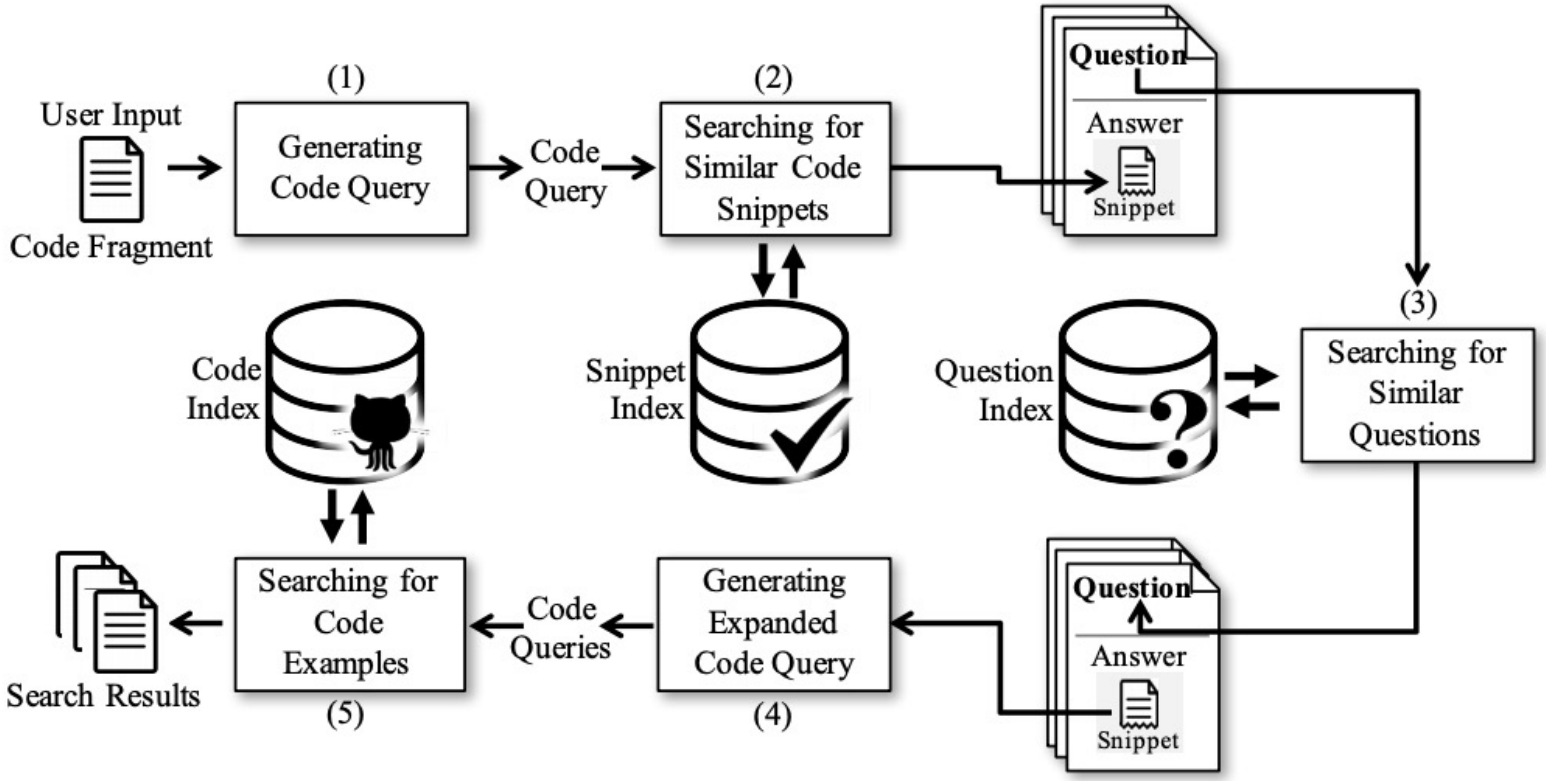
- reasonable performance in patch recommendation @Top5 (we assume not having relevant test cases to validate the patch candidates).

- Comparable performance to many state-of-the-art test-based APR tools in the literature.

> One Last Thing...

securityandtrust.lu

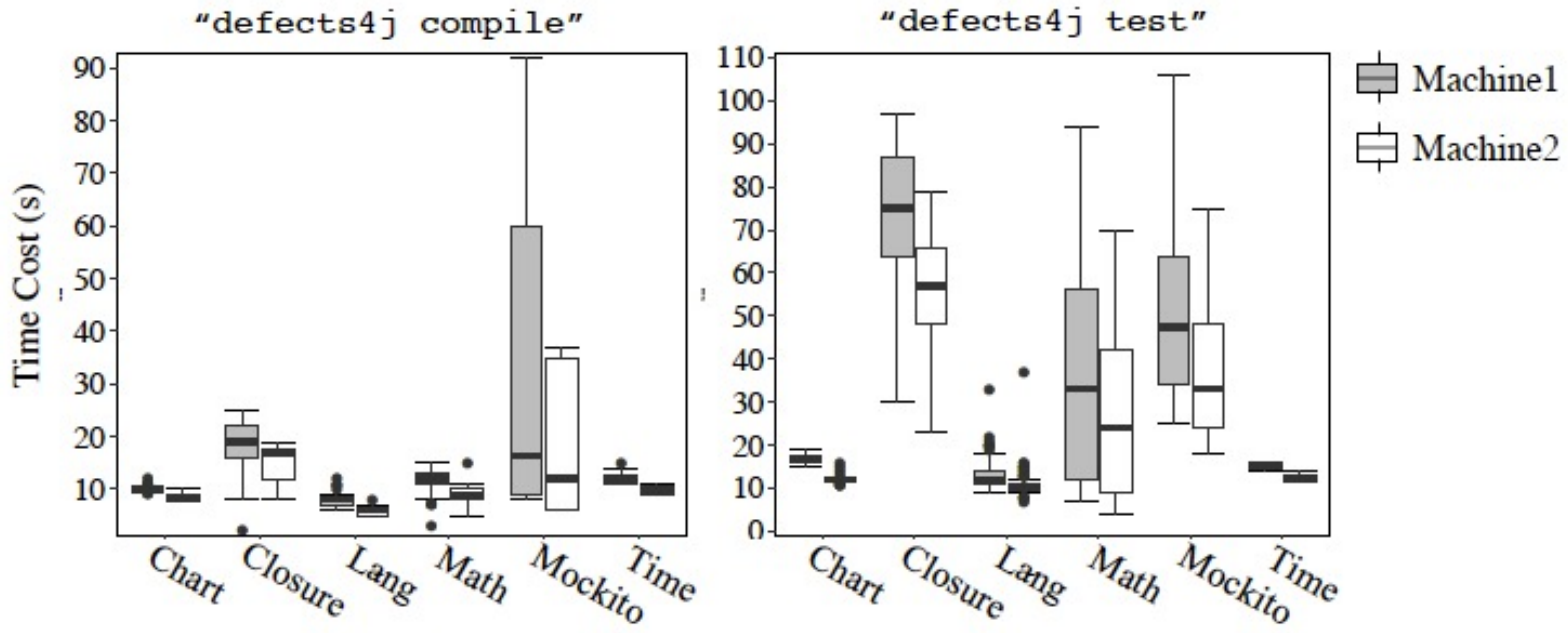
Buggy code can be fixed by simply **replacing it** with « semantically » similar code...



→ **Effective for 21 Defects4J Bugs**

Is patch generation **efficient**?

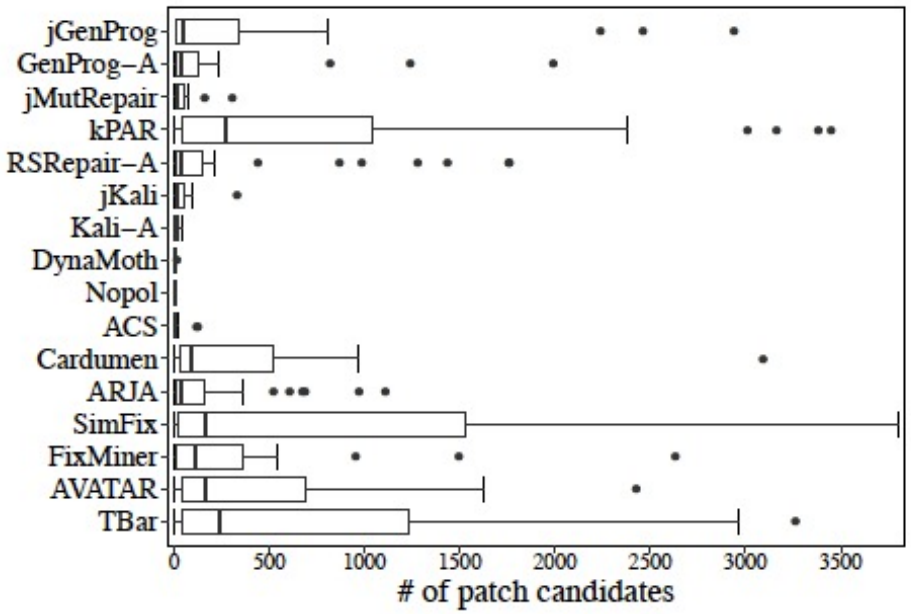
“Time” is not a good metric for efficiency of APR



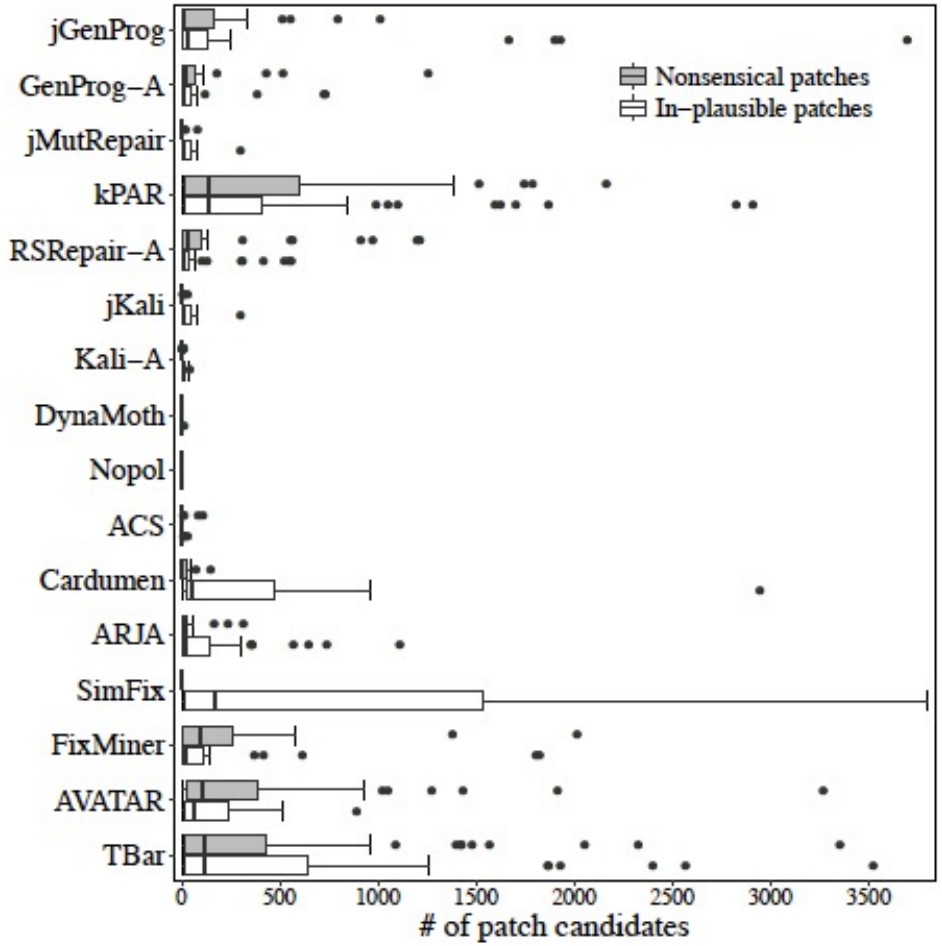
Distribution CPU times for compiling and testing Defects4J programs

- *Machine 1 runs OS X El Capitan 10.11.6 with 2.5 GHz Intel Core i7, 16GB 1600MHz DDR3 RAM.*
- *Machine 2 runs macOS Mojave 10.14.1 with 2.9 GHz Intel Core i9, 32 GB 2400MHz DDR4 RAM.*

“NPC”: Number of Patch Candidates



Efficiency is not yet a widely-valued performance target

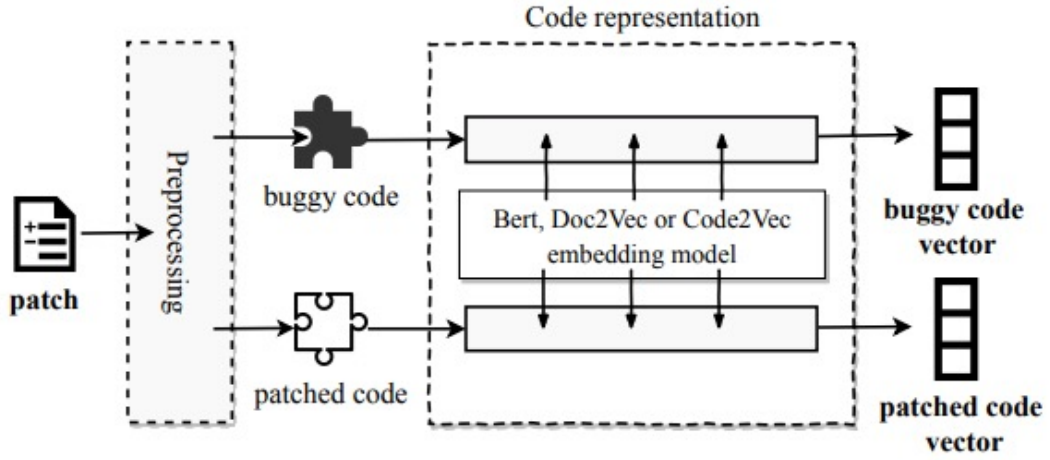


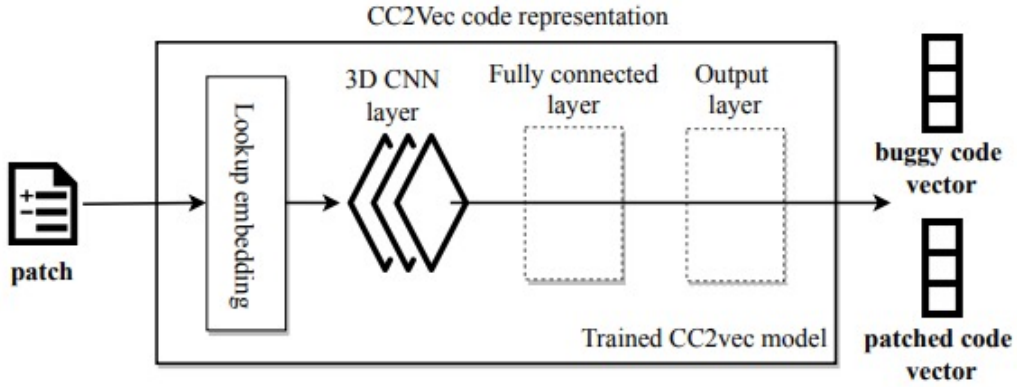
The more templates an APR system considers, the more nonsensical and in-plausible patches it will generate

Can we predict patch correctness?

Representation learning of code changes

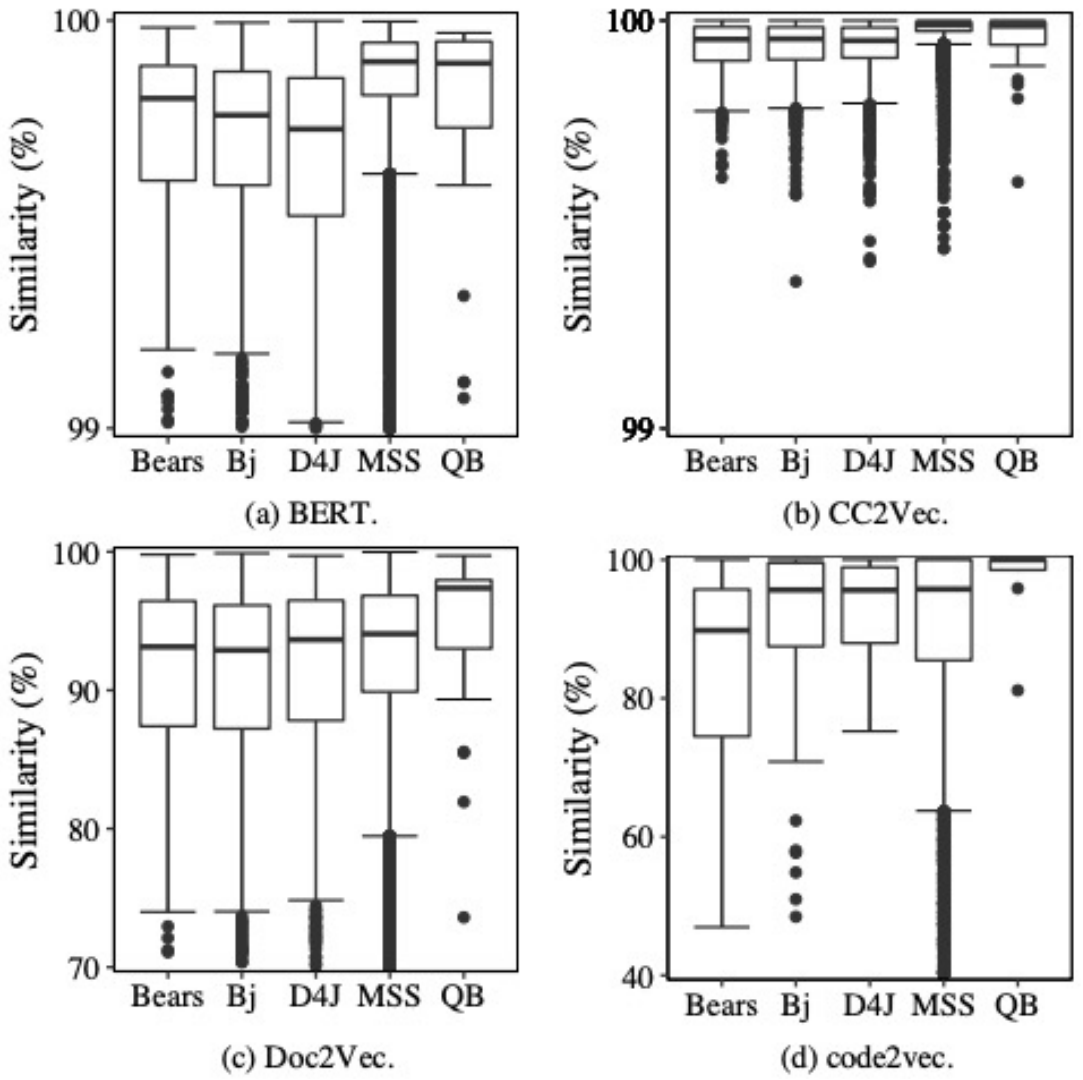
- Static feature learning from patches with BERT, Doc2Vec, Code2Vec and CC2Vec





Representation learning of code changes

- Fixed code is ”similar” to buggy code!



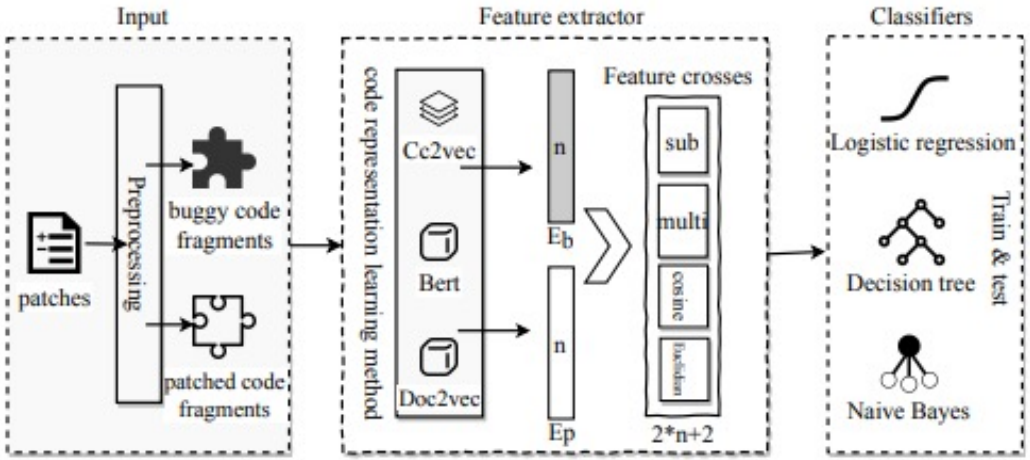
Cosine similarity as a filter

Dataset	# CP	# IP	Threshold	BERT				CC2Vec				Doc2Vec			
				# +CP	# -IP	+Recall	-Recall	# +CP	# -IP	+Recall	-Recall	# +CP	# -IP	+Recall	-Recall
Bears, Bugs.jar and Defects4J	893	61,932	1st Qu.	57	48,846	6.4%	78.9%	797	19,499	89.2%	31.5%	794	25,192	88.9%	40.7%
			Mean	49	51,783	5.5%	83.6%	789	23,738	88.4%	38.3%	771	33,218	86.3%	53.6%
QuixBugs	7	1,461	1st Qu.	4	1,387	57.1%	94.9%	4	1,198	57.1%	82.0%	7	1,226	100%	83.9%
			Mean	4	1,378	57.1%	94.3%	4	1,255	57.1%	85.9%	7	1270	100%	86.9%

*"**# CP**" and "**# IP**" stand for the number of correct and incorrect patches, respectively. "**# +CP**" means the number of correct patches that can be ranked upon the threshold, while "**# -IP**" means the number of incorrect patches that can be filtered out by the threshold. "**+Recall**" and "**-Recall**" represent the recall of identifying correct patches and filtering out incorrect patches, respectively.

Similarity thresholds can be used to filter out <u>some</u> incorrect patches!

Learning to classify patches



Classifier	Embedding	Acc.	Prec.	Recall.	F1	AUC
DecisionTree	BERT	63.6	62.0	57.3	59.6	0.632
	CC2Vec	69.0	66.9	68.0	67.2	0.690
	Doc2Vec	60.2	57.4	57.7	57.5	0.600
Logistic regression	BERT	74.4	73.8	70.3	72.0	0.808
	CC2Vec	73.9	72.5	72.0	72.0	0.788
	Doc2Vec	66.3	65.3	59.9	62.3	0.707
Naive bayes	BERT	60.3	55.6	77.0	64.5	0.642
	CC2Vec	58.0	65.4	22.7	28.5	0.722
	Doc2Vec	66.3	69.4	49.8	57.9	0.714

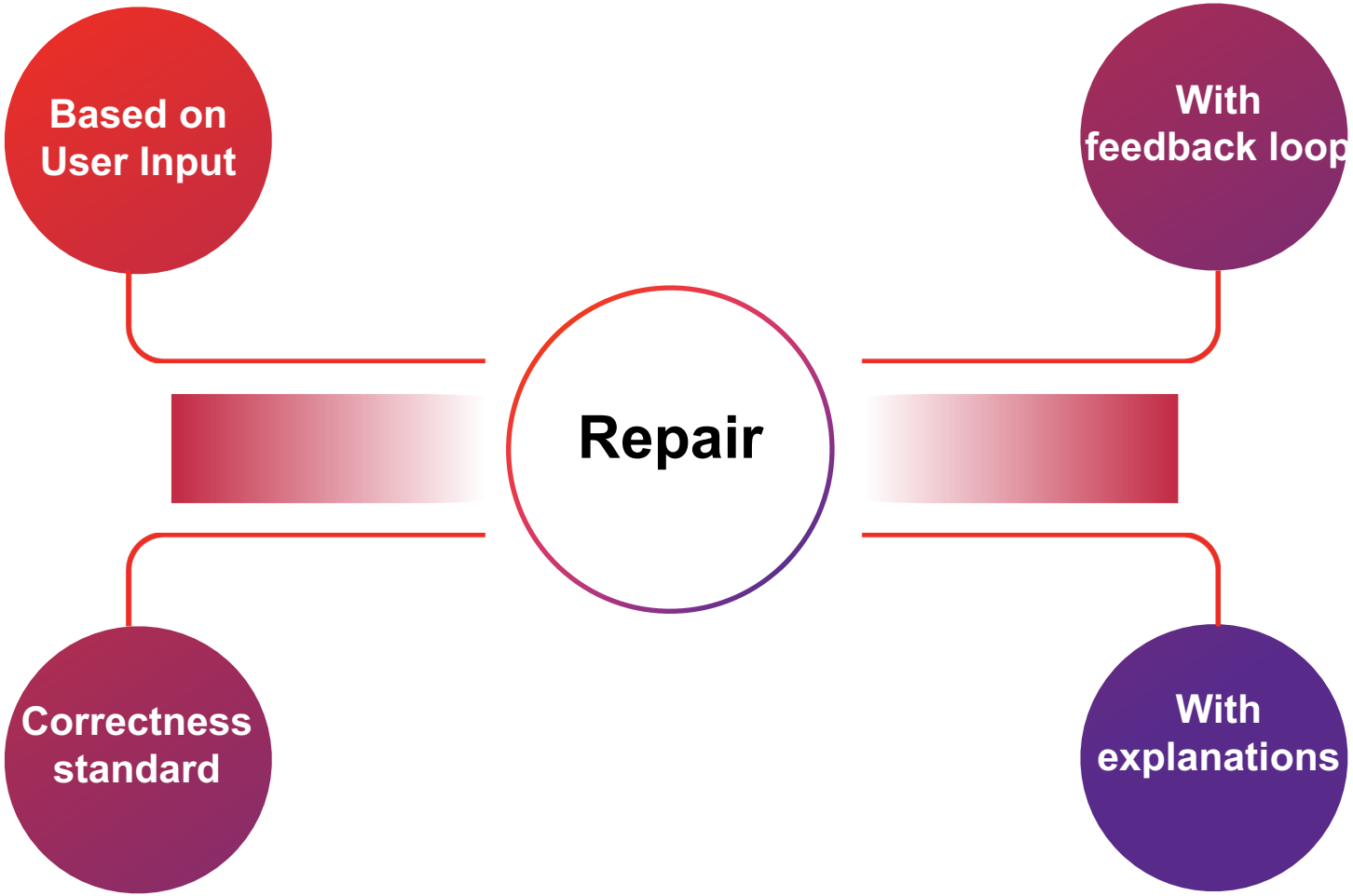
Embeddings offer reasonable performance for statically predicting patch correctness!

SnT

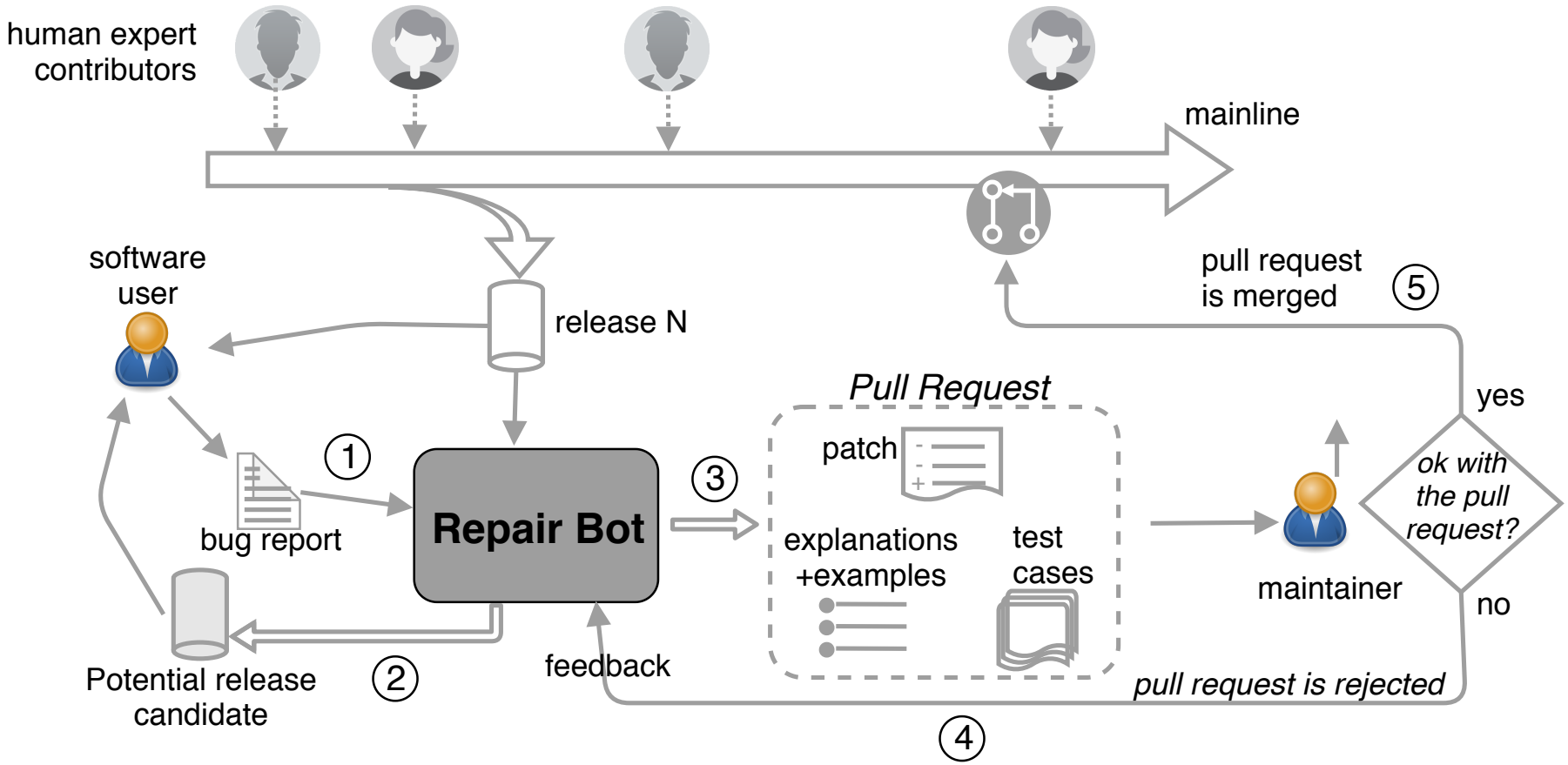
Next steps

UNIVERSITY OF LUXEMBOURG

New contexts/criteria



ERC Starting grant – « NATURAL Program Repair »



University of Luxembourg

Multilingual. Personalised. Connected.

Réparation Automatique des Logiciels: le Rêve et la Fantaisie

GDR GPL, 14 Juin 2021

Prof. Dr. Tegawendé F. BISSYANDE

Contact: tegawende.bissyande@uni.lu