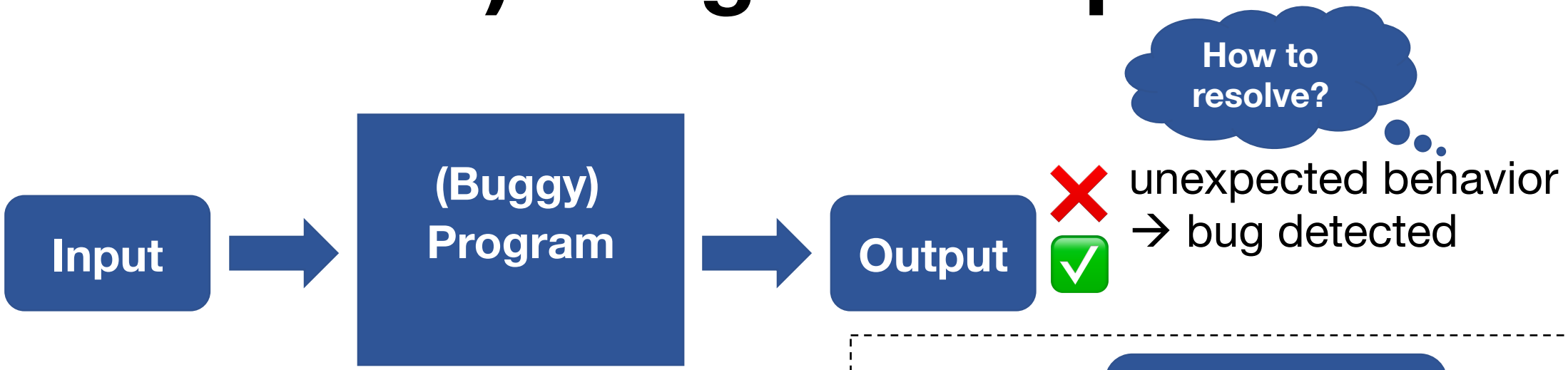
Trust Enhancement Issues in Program Repair

Yannic Noller*
Ridwan Shariffdeen*
Xiang Gao
Abhik Roychoudhury

National University of Singapore

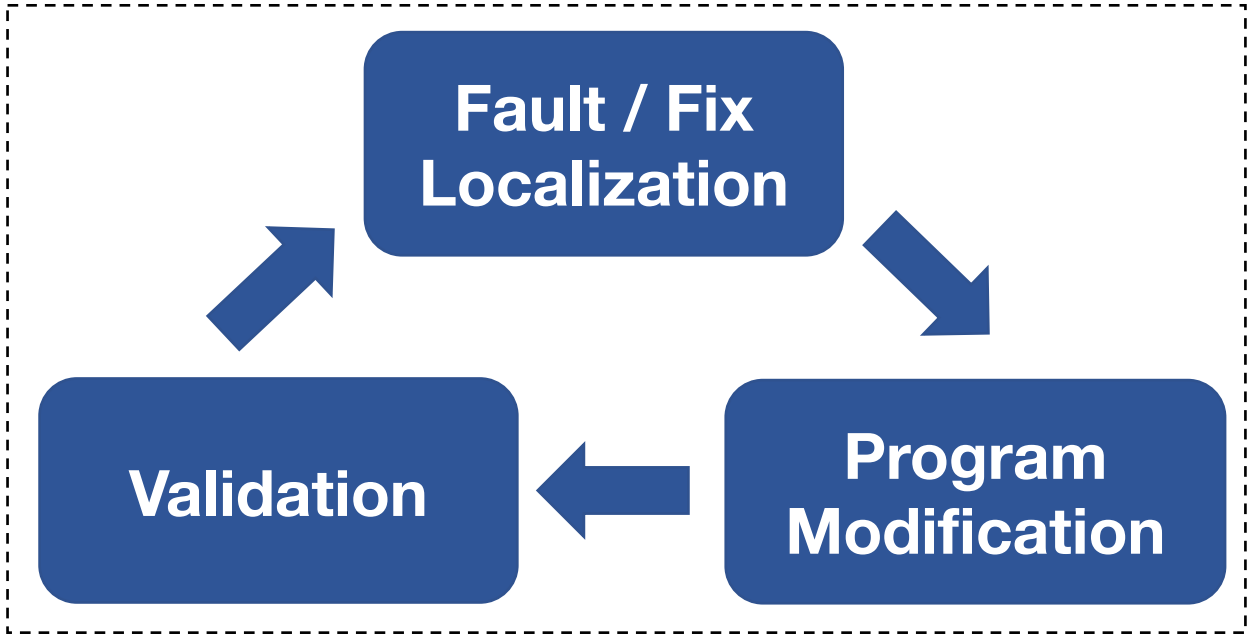
*Joint first authors

(Automated) Program Repair



can be very tedious
… and time consuming

→ **Automated** program repair

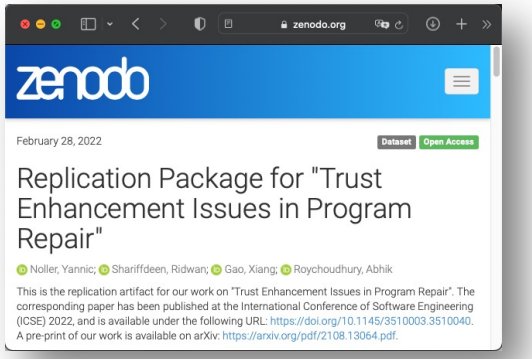


Questions for Automated Program Repair (APR) in practice

- How do developers want to **use** APR?
- Would they **trust** auto-generated patches and **accept** them?
- What kind of **additional inputs** can developers provide and how would these inputs **impact** the **trustworthiness** of the patches?
- Do **current** APR techniques fulfill requirements by developers?

1. Developer Survey

C1 **Usage** of APR
C2 **Availability** of inputs/specification
C3 **Impact** on trust
C4 **Explanations**
C5 **Usage** of APR **side-products**
C6 Background



https://doi.org/10.5281/zenodo.5376903

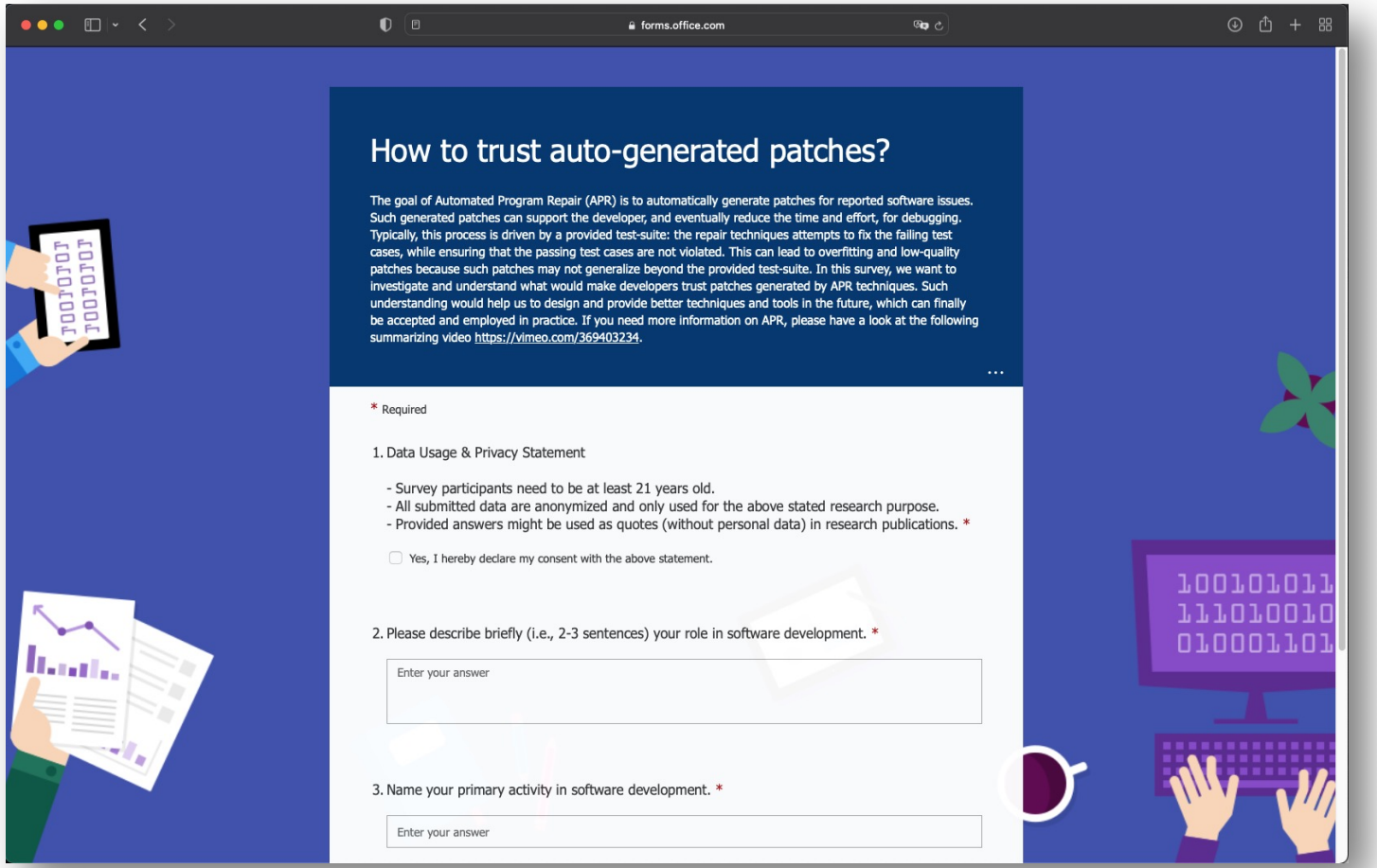
2. Experimental Evaluation

(1) Constraints by developers
(2) Impact of additional inputs

GenProg
Angelix
Prophet
Fix2Fit
CPR

wide-spectrum of APR techniques

Survey Distribution



- ❑ **approval** from the **Institutional Review Board** (IRB)
- ❑ **two** channels:
  (1) Amazon MTurk, and
  (2) Personalized email invitations to contacts from global-wide companies
- ❑ **incentives**: 10 USD for participants on MTurk; otherwise we donated 2 USD to a COVID-19 charity fund
- ❑ **35 questions** (5-point Likert scale, multiple choice, open-ended and close-ended questions)

Research Questions (1/2)

1. Developer Survey

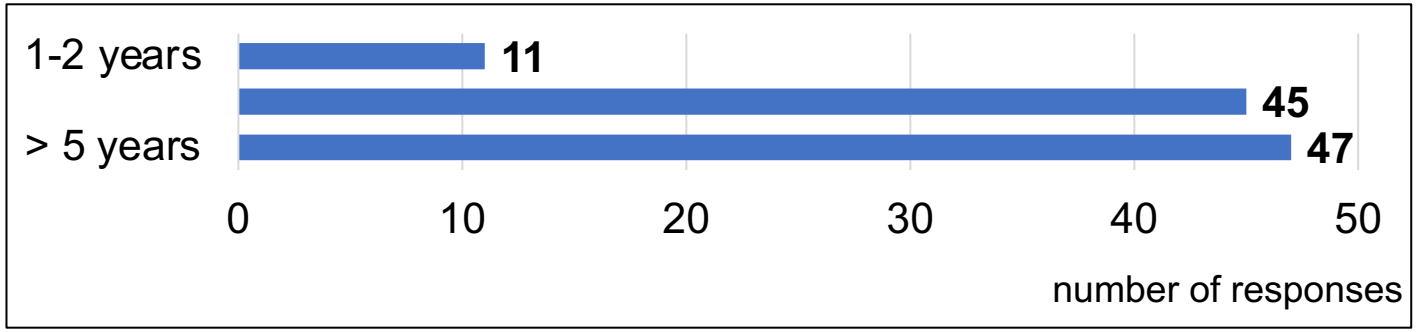
RQ1 To what extent are the developers **ready to accept** and **apply** automated program repair (APR)?

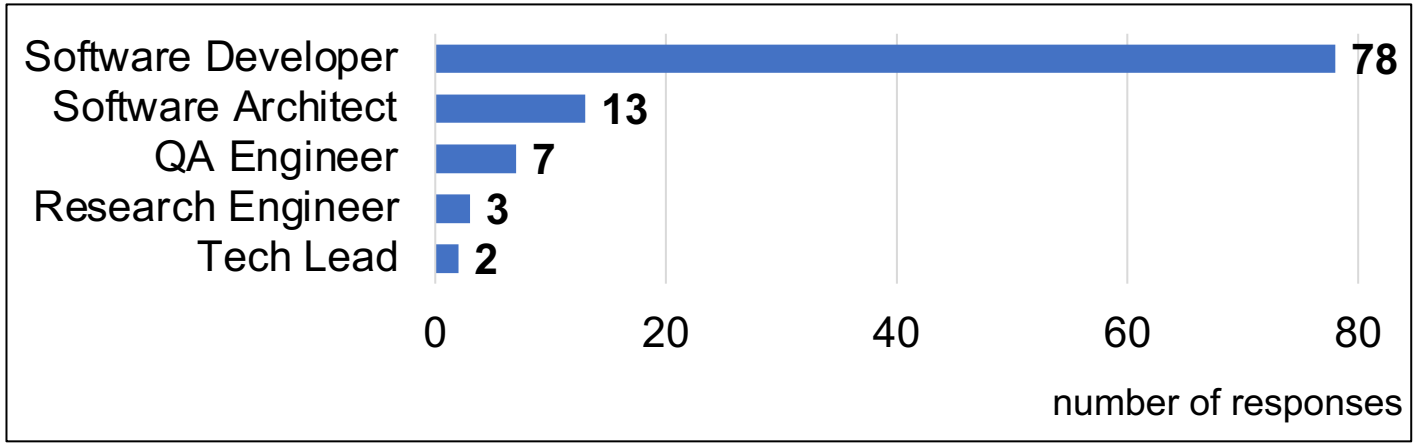
RQ2 Can software developers provide **additional inputs** that would cause higher **trust** in generated patches? If yes, **what** kind of inputs can they provide?

RQ3 What **evidence** from APR will **increase developer trust** in the patches produced?

Demographics

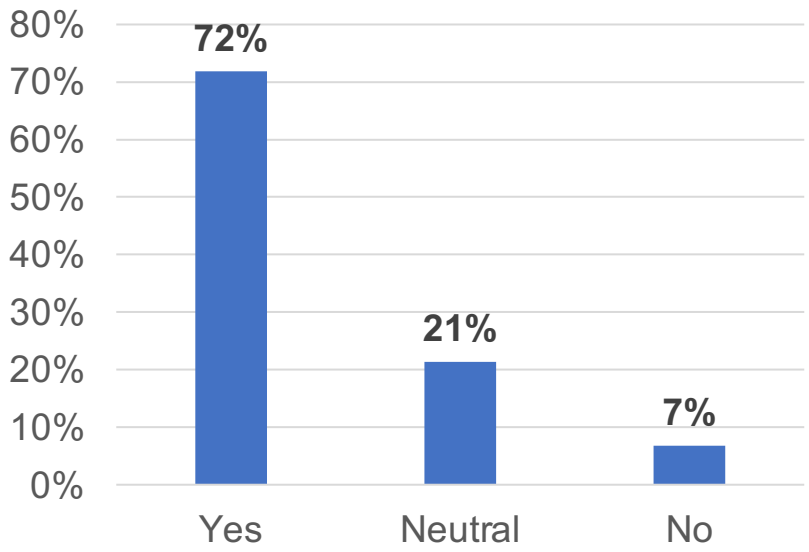
- ❑ **103** software practitioners
- ❑ **89%** with **2+ years experience**
- ❑ 75% **Software Developers**





RQ1: Acceptability of APR

Q1.1 Are you willing to review patches that are submitted by APR techniques?



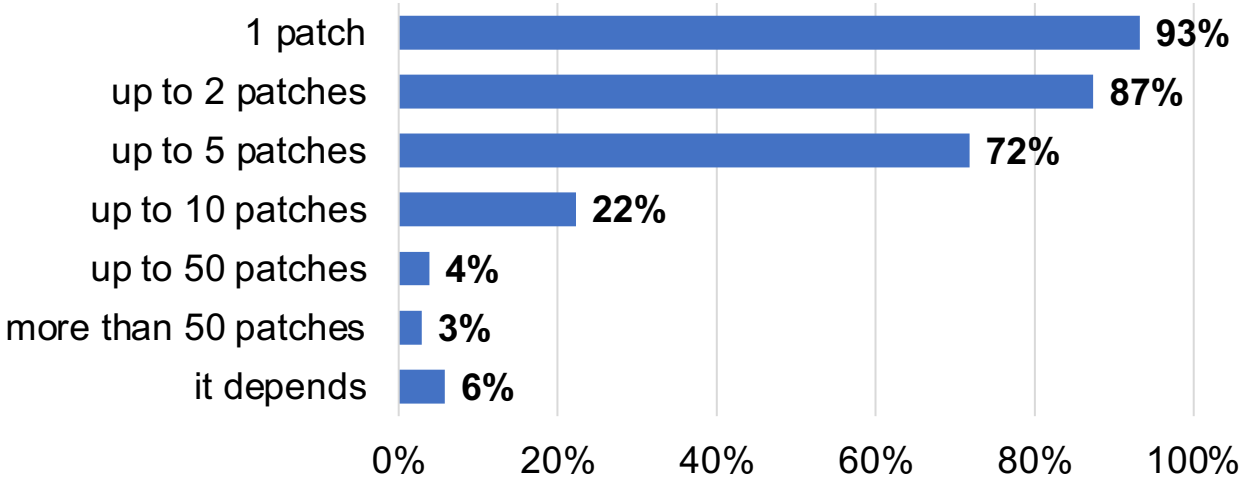
Full developer trust requires **manual** patch review.

Primary Goal: **Save time** for developers.

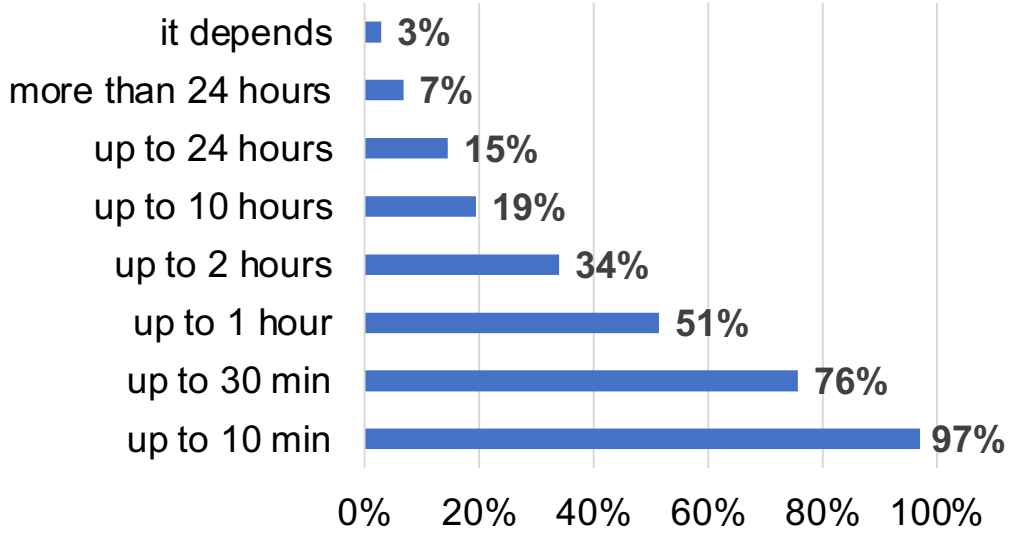
Integration into existing DevOps pipelines.

RQ1: Interaction with APR

Q1.2 How many patches?



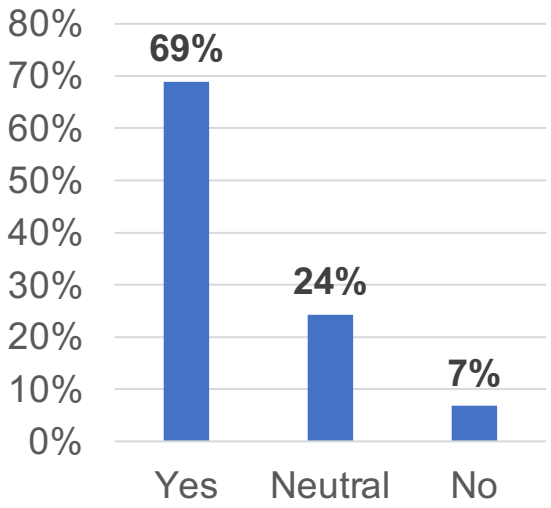
Q1.3 What is an acceptable timeout for APR?



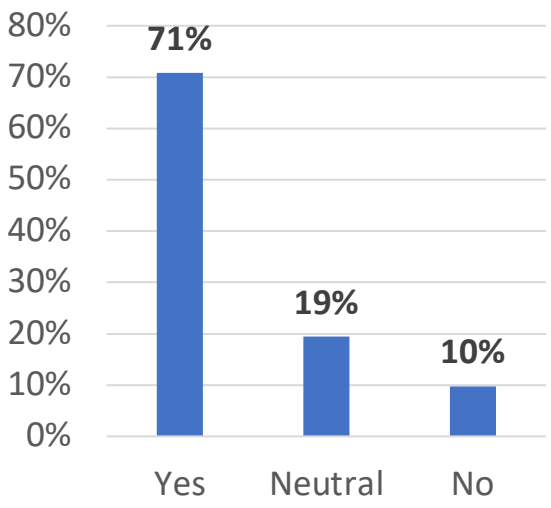
→ Timeout **1 hour** and **Top-5** Patches

RQ2: Artifact Availability

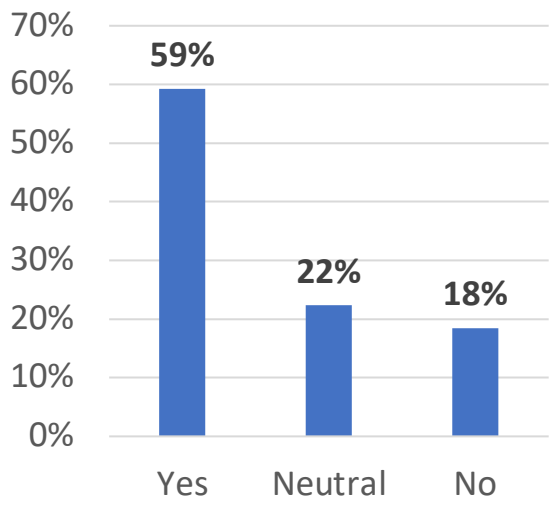
Q2.1 Additional Test Cases



Q2.2 Additional Program Assertions



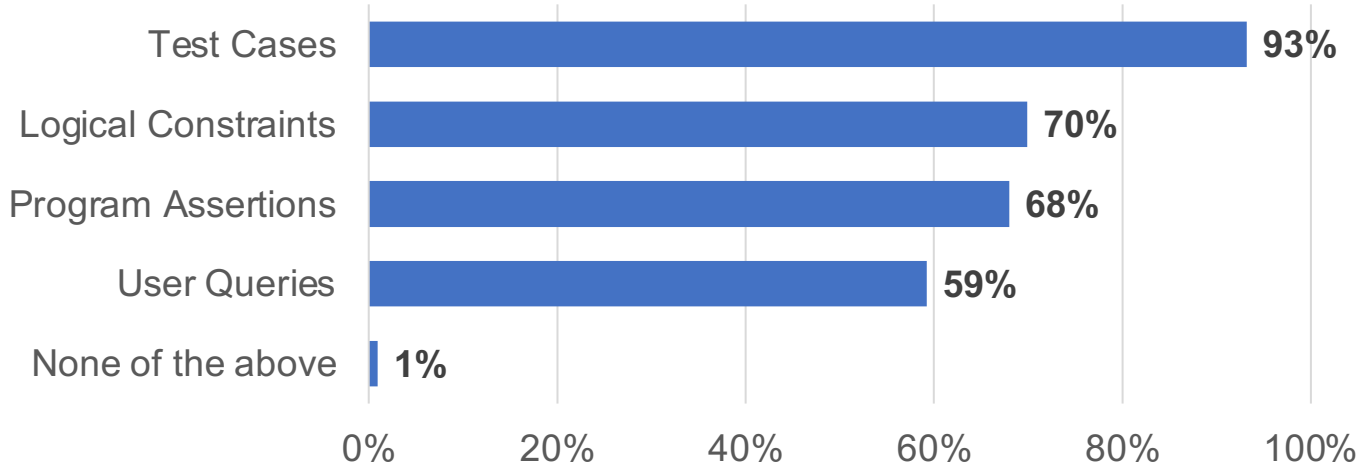
Q2.3 Additional Logical Constraints



Other artifacts include: **execution logs** and relevant **source code locations**.

RQ2: Impact on Trust

Q3.2 Which of the following additional artifacts will increase your trust?



Additional **test cases** would have a great impact on the trustworthiness of APR.

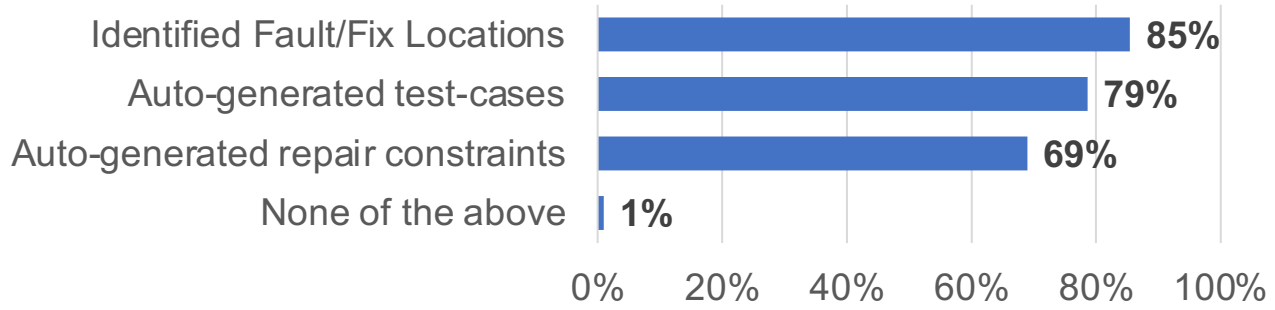
RQ3: Patch Explanations

Evidence is needed to **efficiently** select patch candidates.

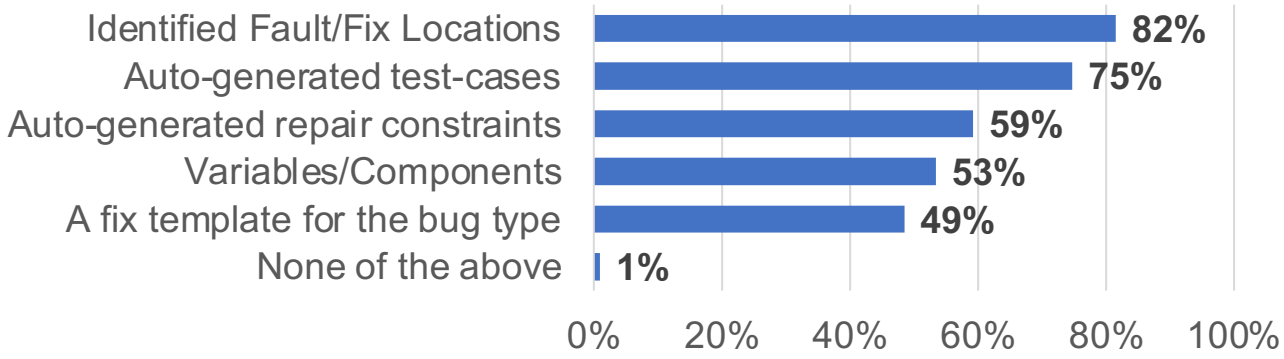
For example: **code coverage** and the **ratio of the covered input space**.

APR **side-products** can assist manual patch validation.

Q5.1 APR side-products helpful to validate the patch?



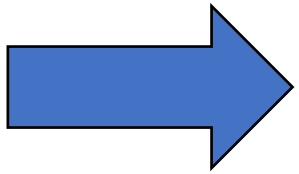
Q5.2 APR side-products helpful to create the patch yourself?



Research Questions (2/2)

RQ4 Can **existing APR** techniques pinpoint **high-quality** patches in the **top-ranking** (e.g., among top-10) patches within a **tolerable time limit** (e.g., 1 hour)?

RQ5 What is the **impact** of **additional inputs** (say, fix locations and additional passing test cases) on the **efficacy of APR**?



Tools/ Techniques

GenProg
Angelix
Prophet
Fix2Fit
CPR

- ❑ investigate specific aspects concerning the adoption of program repair
- ❑ default parameter settings instead of fine-tuning or extending the tools
- ❑ use strict timeouts and computation power restrictions

Experimental Setup

Experiment Configurations

ID	Fix Locations	Passing Tests	Timeout
EC1	Tool fault localization	100%	1hr
EC2	Developer fix location	100%	1hr
EC3	Developer fix location	0%	1hr
EC4	Developer fix location	50%	1hr

ManyBugs Benchmark

Program	Description	LOC	Defects	Test
LibTIFF	Image processing library	77k	7	78
lighttpd	Web server	62k	2	295
PHP	Interpreter	1046k	43	8671
GMP	Math Library	145k	1	146
Gzip	Data compression program	491k	3	12
Python	Interpreter	407k	4	355

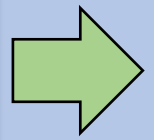
RQ4: Repair Sucess

Subject	Def.	ANGELIX				PROPHET				GENPROG				FIX2FIT				CPR		
		EC1	EC2	EC3	EC4	EC1	EC2	EC3	EC4	EC1	EC2	EC3	EC4	EC1	EC2	EC3	EC4	EC2	EC3	EC4
LibTIFF	7	3/1	3/1	3/1	3/1	1/0	1/0	1/0	1/0	5/0	5/0	5/0	5/0	5/1	4/1	4/1	4/1	4/2	4/2	4/2
lighttpd	2	-	-	-	-	1/0	0/0	0/0	0/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	1/0	-	-	-
PHP	43	0/0	0/0	0/0	0/0	0/0	0/0	2/1	3/1	0/0	0/0	10/1	0/0	8/1	4/2	7/2	5/1	5/4	5/4	5/4
GMP	1	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0	1/1	1/1	1/1
Gzip	3	0/0	1/0	1/0	1/0	0/0	1/1	1/1	1/1	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0	3/1	3/1	3/1
Python	4	-	-	-	-	0/0	1/1	1/1	1/1	0/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0	-	-	-
Overall	60	3/1	4/1	4/1	4/1	2/0	3/2	5/3	6/3	6/0	6/0	16/1	6/0	14/2	9/3	12/3	10/2	13/8	13/8	13/8

Under our tight constraints (i.e., a strict **1-hour timeout** and the **top-10** ranking restriction), the state-of-the-art repair techniques cannot identify many plausible patches.

What do we learn from this?

Automated program repair tools are only beginning to gain adoption, and are still an emerging technology.



Can we **identify** what it would take to **increase** the adoption of program repair?

Let's inspect the results closer on the next slides.

RQ4: Plausible Patches

Program	#Vul	Angelix	Prophet	GenProg	Fix2Fit
LibTIFF	7	3	1	5	5
Lighttpd	2	0	1	1	1
PHP	43	0	0	0	8
GMP	1	0	0	0	0
GZip	3	0	0	0	0
Python	4	0	0	0	0
Total	60	3	2	6	14

Plausible Patches generated by APR for ManyBugs benchmark in 1h timeout using the tool's own fault localization (EC1).

1-hour timeout is a difficult constraint for current techniques

prior experiments evaluated the **capability** to generate a patch

Note: scenario-specific *parameter fine-tuning* can affect the results greatly

RQ4: Patch Space Exploration

Program	#Vul	Angelix	Prophet	GenProg	Fix2Fit
LibTIFF	7	86	25	1	100
Lighttpd	2	0	20	1	100
PHP	43	96	23	1	63
GMP	1	100	41	5	0
GZip	3	100	6	18	100
Python	4	0	15	2	0
Total	60	95	22	5	91

Exploration Ratio by APR for ManyBugs benchmark in 1h timeout using the tool's own fault localization (EC1).

a large/rich search space requires an **efficient exploration** strategy

Patch Space Abstractions can support this

RQ4: Patch Ranking

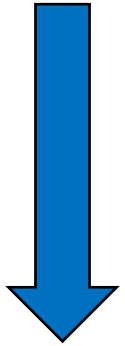
Program	#Vul	Angelix	Prophet	GenProg	Fix2Fit
LibTIFF	7	1	0	0	1
Lighttpd	2	0	0	0	0
PHP	43	0	0	0	1
GMP	1	0	0	0	0
GZip	3	0	0	0	0
Python	4	0	0	0	0
Total	60	1	0	0	2

Correct Patches generated by APR for ManyBugs benchmark in 1h timeout using the tool's own fault localization (EC1).

an **effective patch ranking** is necessary for the developer

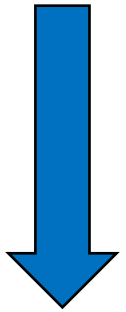
RQ5: Impact of additional inputs

Impact of **fix location**



better fix location ⇏ better repair
(techniques are limited by their search space construction and exploration)

Impact of **available number of test cases**



Variation of test cases causes different effects.
(intelligent test selection needed)

Conclusions

- **Developer Survey** with > 100 software practitioners
  - *high-quality* patches in a short time period (1-hour timeout, top-10 patches)
  - *low* interaction with tool
  - exchange of ***artifacts*** (e.g., test cases, patch explanations)
- **Experimental Evaluation** of state-of-the-art APR techniques
  - developer‘s constraints are *tough*
  - *rich* search space needed: can be supported by ***user inputs***
  - *efficient* search space exploration: can be supported by ***abstractions***
  - *patch ranking* should not be ignored

How to get closer to *trust*?

Developers need support for *efficient* patch review:

(1) insights why the patch is targeting the right issue
e.g., root cause analysis, the results of our fault/fix localization, inferred repair constraints

(2) evidence on the **correctness** of the patch
e.g., additional test cases, test suite coverage information or input coverage information

(3) easy accessibility of the patches
e.g., better ranking and navigation of patch candidates in the programming environment

NUS National University of Singapore

→ **APR side-products** can support some of these steps (e.g., identified fault locations and inferred repair constraints).

→ We definitely need *more* research on patch **explanations**, patch **ranking**, and **efficient traversal** of an *abstract* patch space.