Program Analysis for Safe and Secure Software Evolution

Cristian Cadar

SOFTWARE RELIABILITY GROUP

Imperial College London

Funded by UKRI Engineering and Physical Sciences Research Council

erc European Research Council
Established by the European Commission

University of Stuttgart
Stuttgart, Germany
15 April 2025

By Fuzzypiggy, Wikipedia

@ Imperial College London

Imperial College
London

SOFTWARE RELIABILITY
GROUP

http://srg.doc.ic.ac.uk

Current and recent members

Cristian Cadar

Anastasios Andronidis

Frank Busse

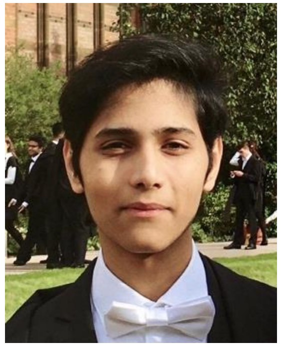
Manuel Carrasco

Karine Even-Mendoza

Martin Nowack

Jordy Ruiz

Daniel Schemmel



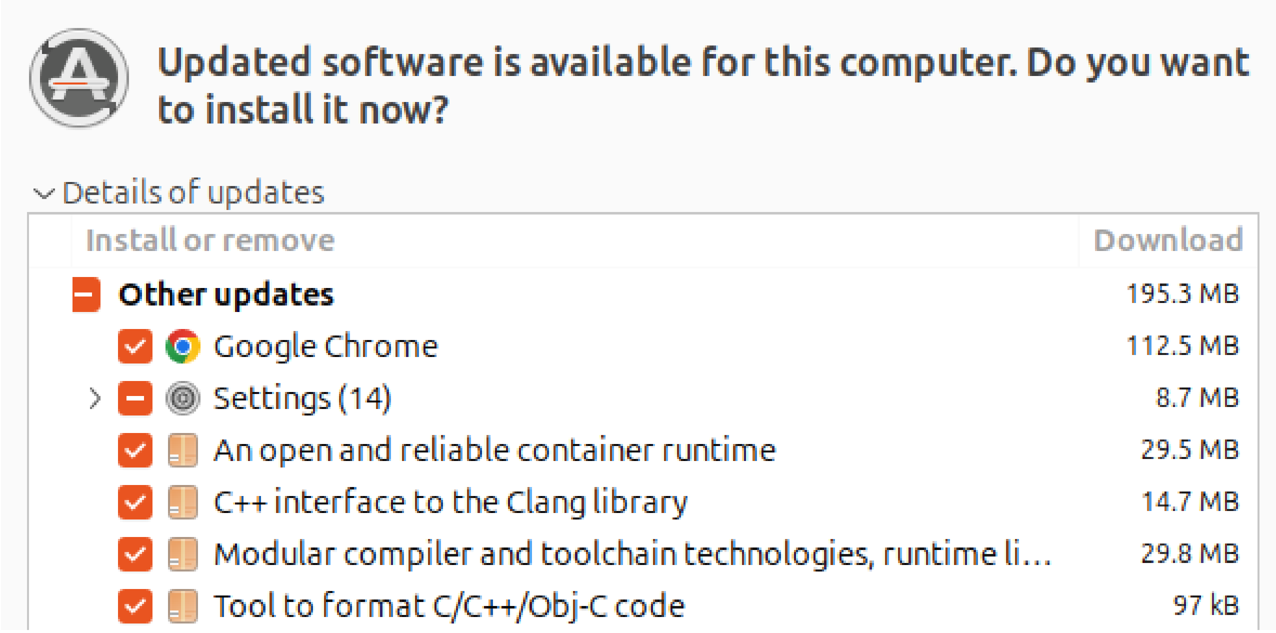
Arindam Sharma

Bachir Bendrissou

Ahmed Zaki

By Julian Herzog, CC BY 4.0

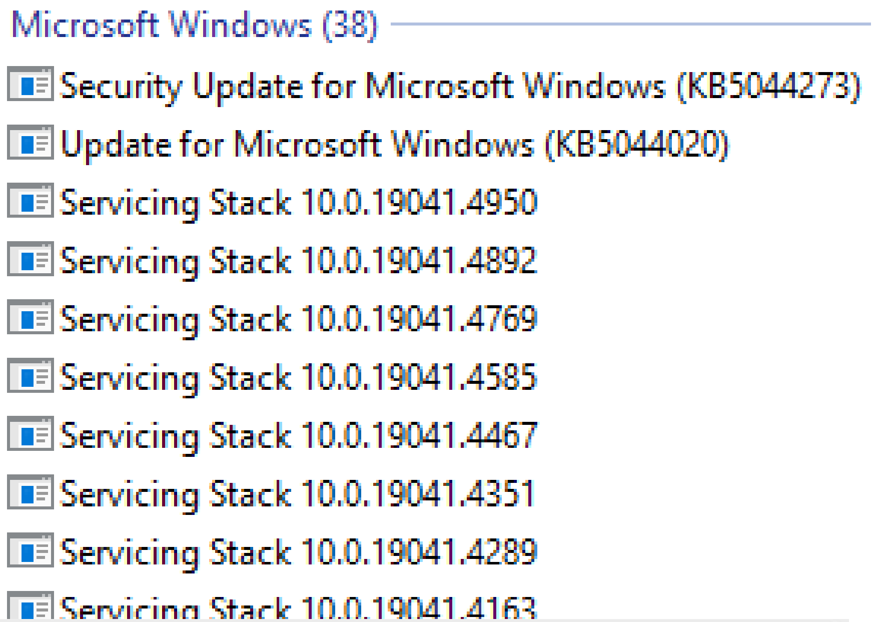
@ University of Stuttgart / Sven Cichowicz



Updated software is available for this computer. Do you want to install it now?

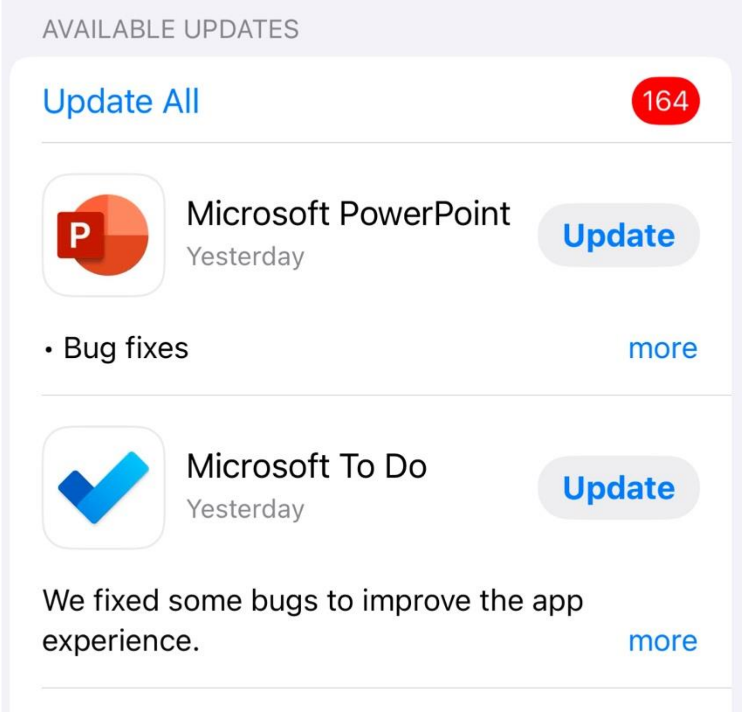
Details of updates

Install or remove	Download
⊟ **Other updates**	195.3 MB
☑ Google Chrome	112.5 MB
⊟ Settings (14)	8.7 MB
☑ An open and reliable container runtime	29.5 MB
☑ C++ interface to the Clang library	14.7 MB
☑ Modular compiler and toolchain technologies, runtime li…	29.8 MB
☑ Tool to format C/C++/Obj-C code	97 kB



Microsoft Windows (38)

- Security Update for Microsoft Windows (KB5044273)
- Update for Microsoft Windows (KB5044020)
- Servicing Stack 10.0.19041.4950
- Servicing Stack 10.0.19041.4892
- Servicing Stack 10.0.19041.4769
- Servicing Stack 10.0.19041.4585
- Servicing Stack 10.0.19041.4467
- Servicing Stack 10.0.19041.4351
- Servicing Stack 10.0.19041.4289
- Servicing Stack 10.0.19041.4163



AVAILABLE UPDATES

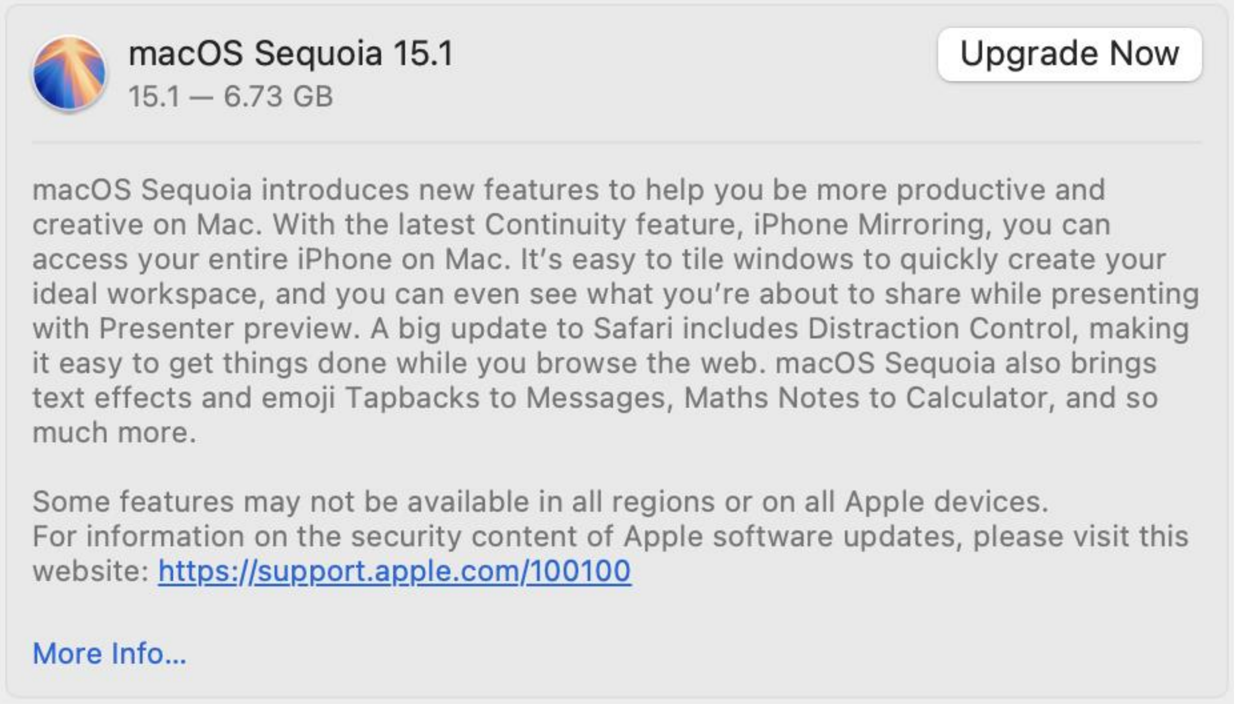
Update All 164

Microsoft PowerPoint
Yesterday
Update

• Bug fixes
more

Microsoft To Do
Yesterday
Update

We fixed some bugs to improve the app experience.
more



macOS Sequoia 15.1
15.1 — 6.73 GB
Upgrade Now

macOS Sequoia introduces new features to help you be more productive and creative on Mac. With the latest Continuity feature, iPhone Mirroring, you can access your entire iPhone on Mac. It's easy to tile windows to quickly create your ideal workspace, and you can even see what you're about to share while presenting with Presenter preview. A big update to Safari includes Distraction Control, making it easy to get things done while you browse the web. macOS Sequoia also brings text effects and emoji Tapbacks to Messages, Maths Notes to Calculator, and so much more.

Some features may not be available in all regions or on all Apple devices.
For information on the security content of Apple software updates, please visit this website: https://support.apple.com/100100

More Info…

Evolving Software

- Poorly validated code changes often introduce bugs & vulnerabilities
- Some with catastrophic impact

Heartbleed (2014)

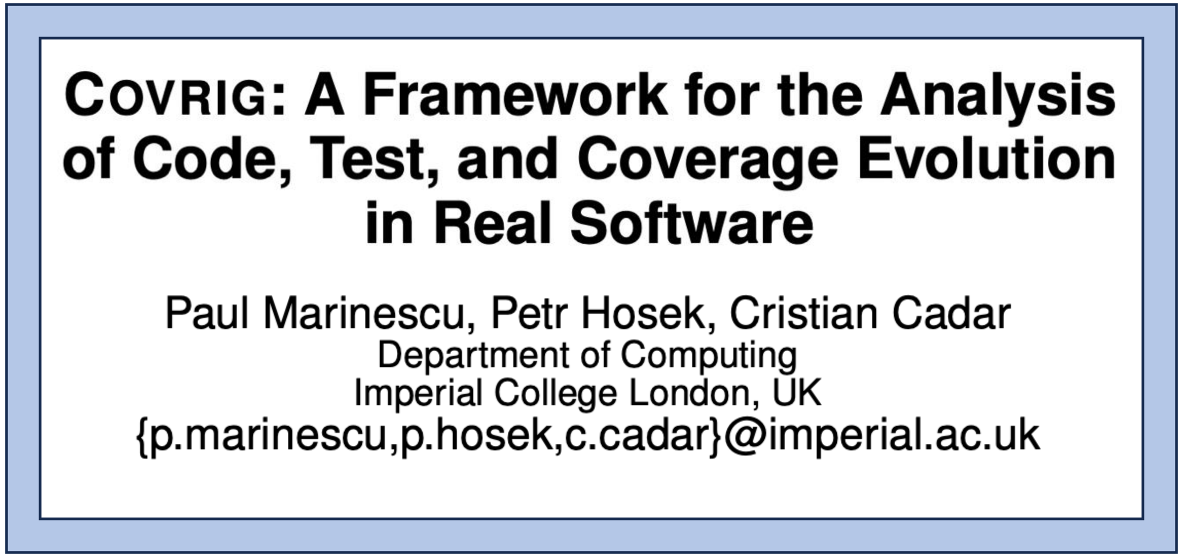
Shellshock (2014)

Stagefright (2016)

Channel File 291 Incident

Crowdstrike (2024)

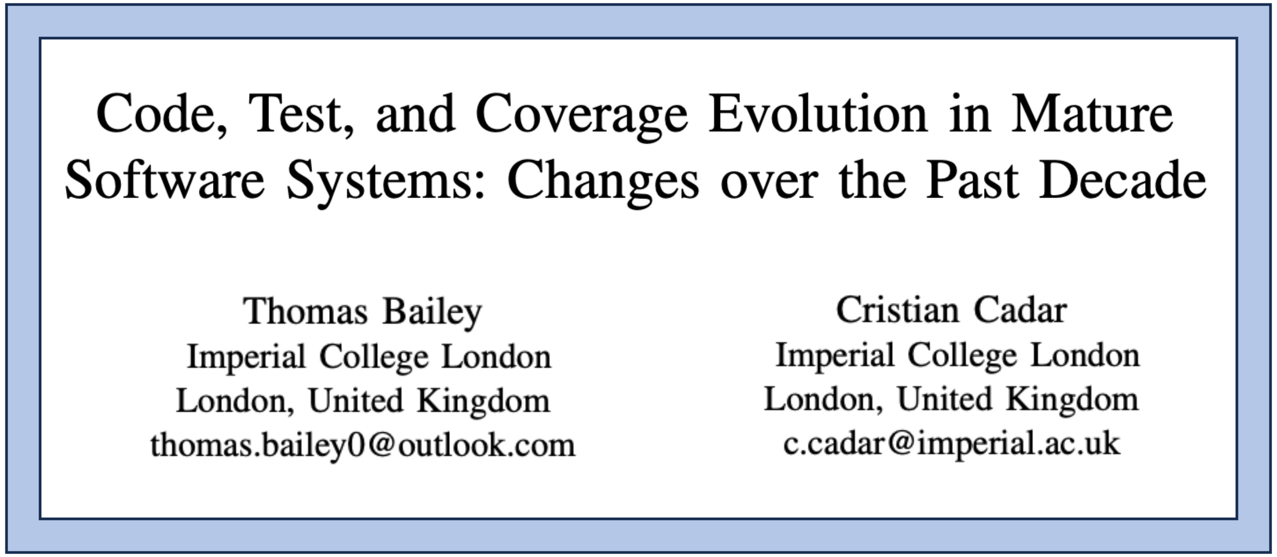
ISSTA 2014



COVRIG: A Framework for the Analysis of Code, Test, and Coverage Evolution in Real Software

Paul Marinescu, Petr Hosek, Cristian Cadar
Department of Computing
Imperial College London, UK
{p.marinescu,p.hosek,c.cadar}@imperial.ac.uk

ICST 2025



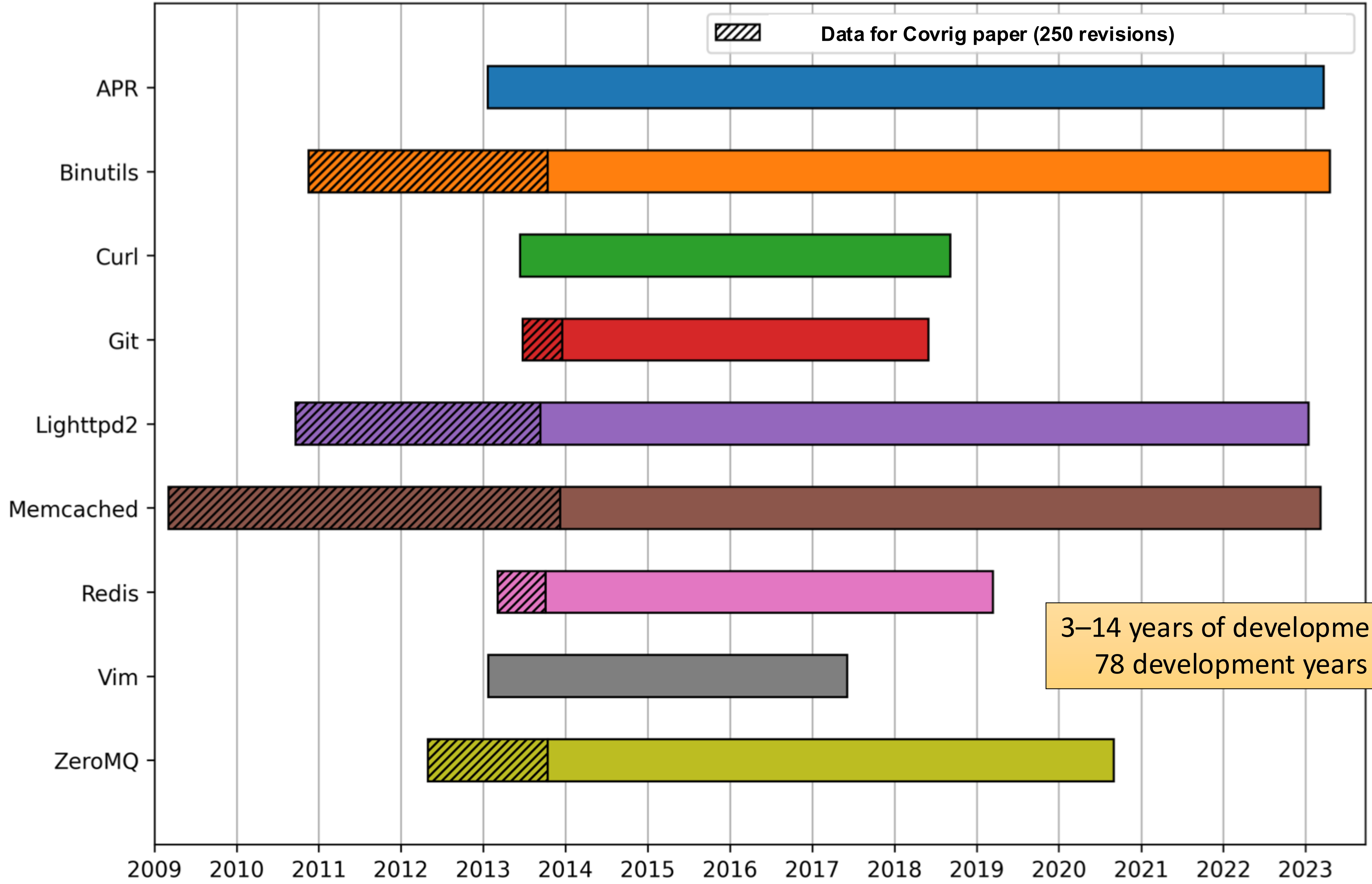
Code, Test, and Coverage Evolution in Mature Software Systems: Changes over the Past Decade

Thomas Bailey
Imperial College London
London, United Kingdom
thomas.bailey0@outlook.com

Cristian Cadar
Imperial College London
London, United Kingdom
c.cadar@imperial.ac.uk

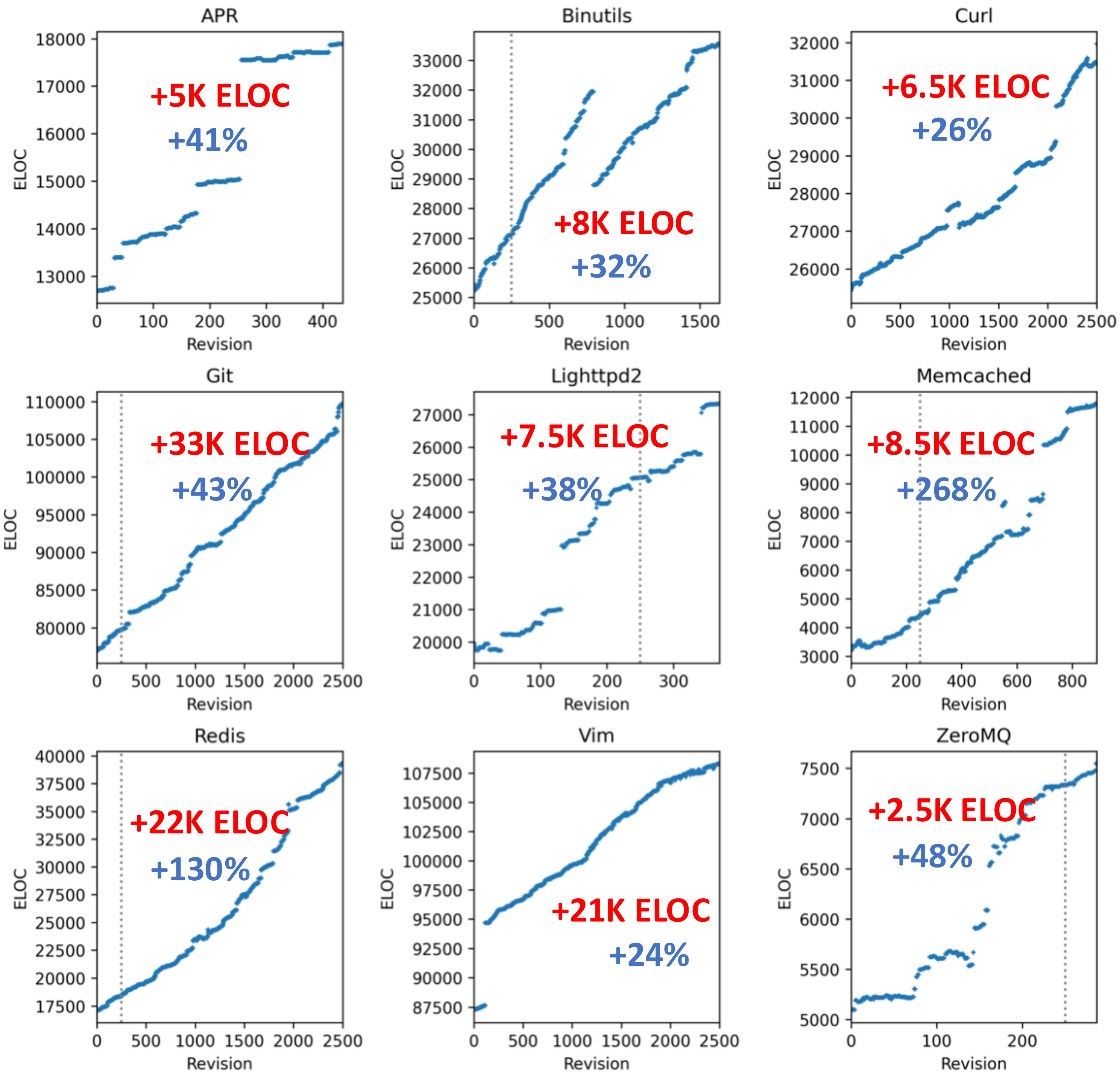
- 6 popular open-source systems
- Analysed 250 revisions per app
- Conclusion: LOTS of code added or modified without being tested

A decade later: Have things changed?



3–14 years of development/project
78 development years in total

ELOC/time


APR: +5K ELOC +41%

Binutils: +8K ELOC +32%

Curl: +6.5K ELOC +26%

Git: +33K ELOC +43%

Lighttpd2: +7.5K ELOC +38%

Memcached: +8.5K ELOC +268%

Redis: +22K ELOC +130%

Vim: +21K ELOC +24%

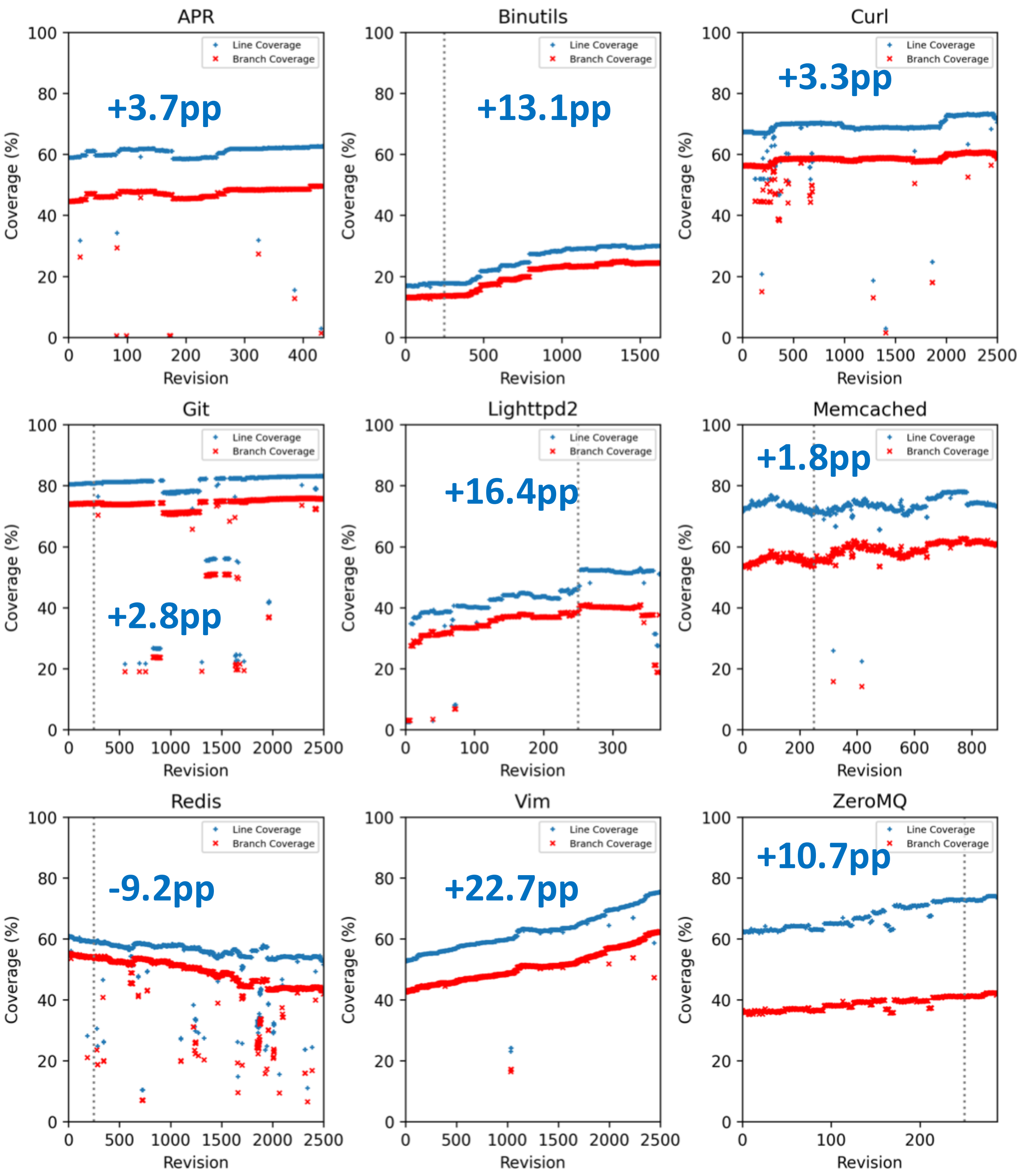
ZeroMQ: +2.5K ELOC +48%

Code increases of
2.5K – 33K ELOC,
24% – 268%

Coverage Evolution

Line coverage increases by 2.8 – 22.7pp
It decreases in Redis by 9.2pp

5/9 projects have
under 50% branch coverage



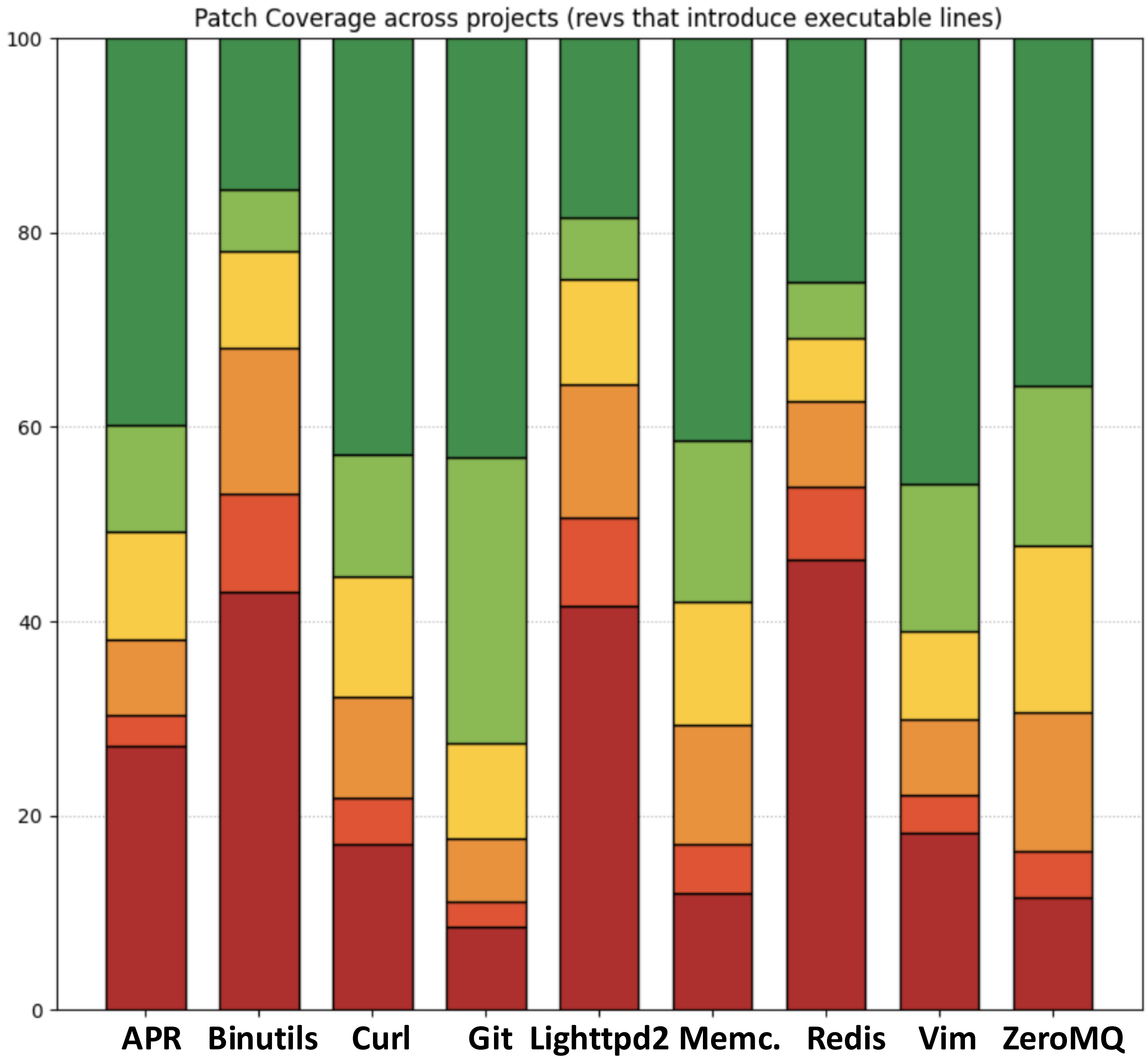
🟦 Line coverage

🟥 Branch coverage

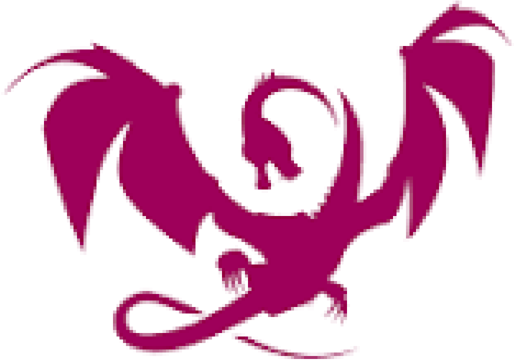
Patch Coverage

Percentage of ELOC in a patch covered by the test suite

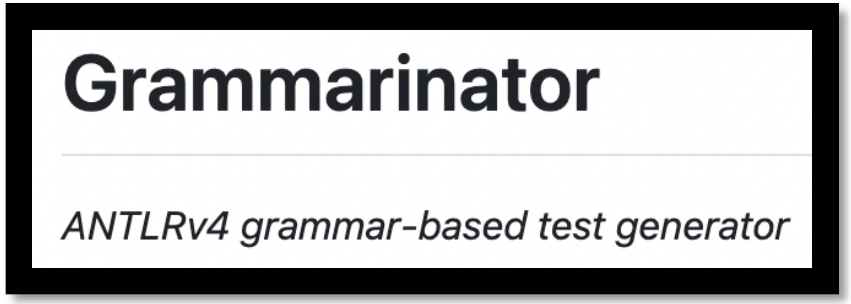
Low bar: reaching the patch does not mean testing it



Can Program Analysis Tools Help?

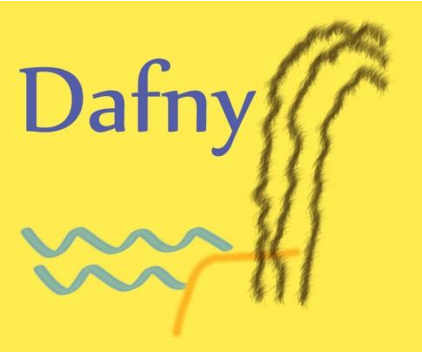


Clang Static Analyzer



Grammarinator

ANTLRv4 grammar-based test generator



Clang Static Analyzer

cbmc

AFL++

Designed for whole program testing

EVOSUITE

Csmith

Grammarinator

ANTLRv4 grammar-based test generator

Dafny

Whole-Program Testing
i.e. Testing from Scratch

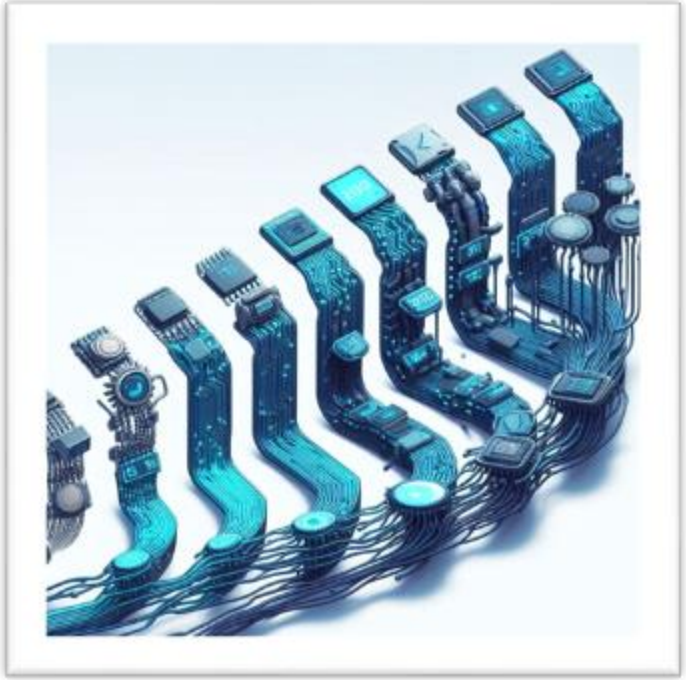
Expensive and wasteful

- Lots of **wasteful repetition** across versions
- New bugs are often **missed** with patch sometimes not even reached
- Same bugs found over and over again, with the need for **deduplication**
- Bugs reported with significant delay: **expensive context switching**

Developers need feedback within *minutes* of patch submission
Quick directed testing **campaigns required in a CI/CD context**

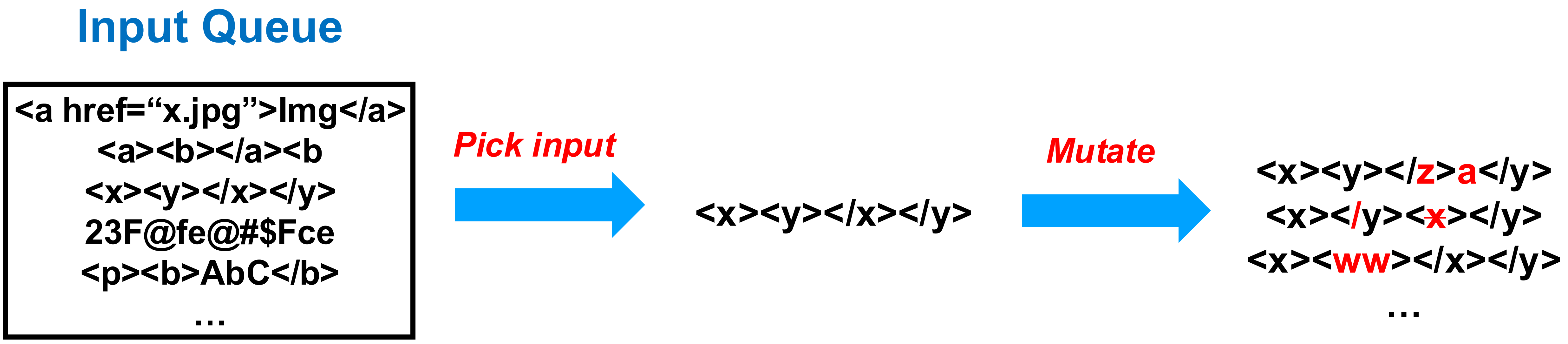
Testing Evolving Software

Reuse testing results
of previous versions



Direct testing effort
toward the changes

Greybox Fuzzing:
Coverage-guided Mutation-based Fuzzing



Input Queue

<a href=“x.jpg”>Img</a>
<a><b></a><b
<x><y></x></y>
23F@fe@#$Fce
<p><b>AbC</b>
…

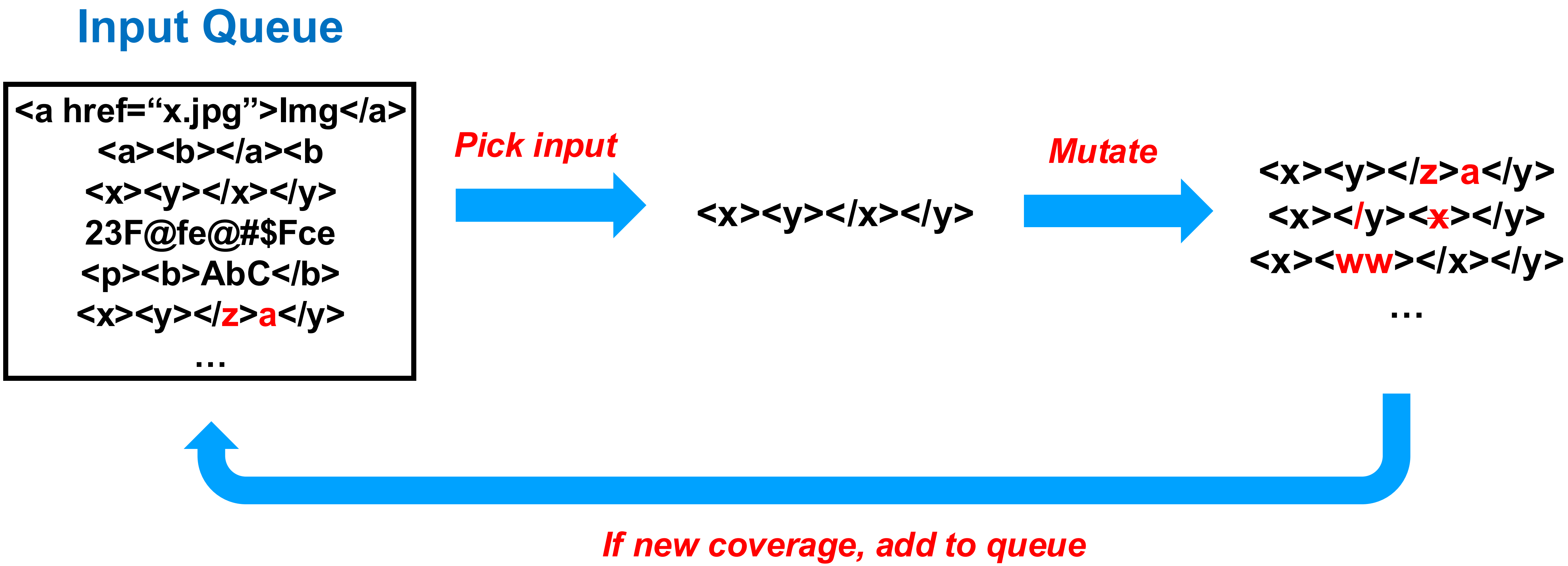
Pick input

<x><y></x></y>

Mutate

<x><y></z>a</y>
<x></y><~~x~~></y>
<x><ww></x></y>
…

Greybox Fuzzing:
Coverage-guided Mutation-based Fuzzing



Input Queue

<a href=“x.jpg”>Img</a>
<a><b></a><b
<x><y></x></y>
23F@fe@#$Fce
<p><b>AbC</b>
<x><y></z>a</y>
…

Pick input

<x><y></x></y>

Mutate

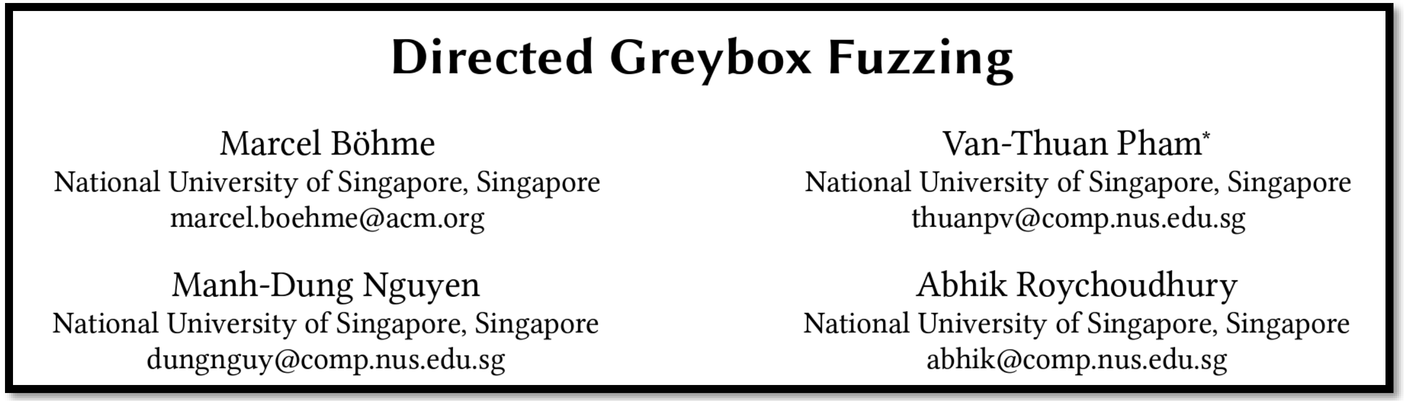
<x><y></z>a</y>
<x></y><~~x~~></y>
<x><ww></x></y>
…

If new coverage, add to queue

AFLGo:
State-of-the-Art Directed Greybox Fuzzing

- AFLGo is a pioneering tool for directed greybox fuzzing
- It extends traditional fuzzing by targeting specific code areas
- Computes distance estimates to prioritize inputs close to the target

  - But distance computation can be expensive
  - Fuzzing budget may be exhausted before any fuzzing is done



Directed Greybox Fuzzing

Marcel Böhme
National University of Singapore, Singapore
marcel.boehme@acm.org

Van-Thuan Pham*
National University of Singapore, Singapore
thuanpv@comp.nus.edu.sg

Manh-Dung Nguyen
National University of Singapore, Singapore
dungnguy@comp.nus.edu.sg

Abhik Roychoudhury
National University of Singapore, Singapore
abhik@comp.nus.edu.sg

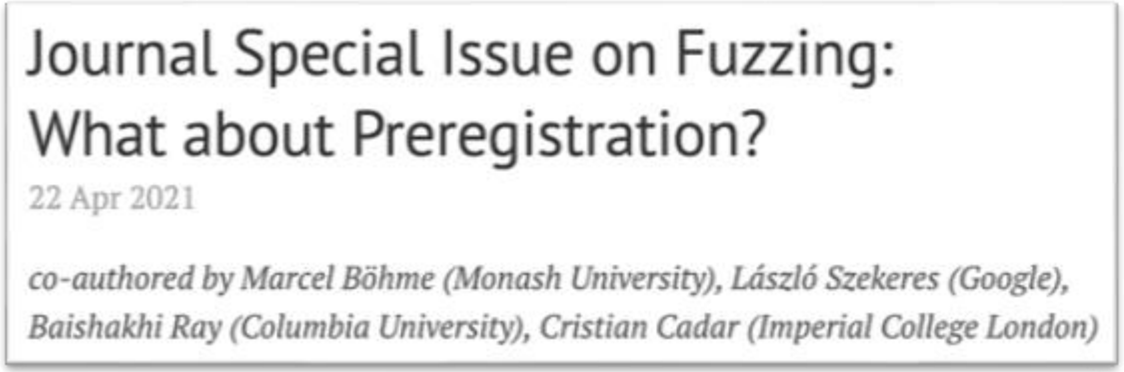
PaZZER = Patch + Fuzzer

- Designed to be practical for short CI/CD runs
- Aims to find a sweet spot between time spent in distance computation and effectiveness
- Relies on less precise but quick distance estimates (using only the call graph)
- Computes distances incrementally (LPA*, Anytime-D*)

Pazzer Case Study

ObjDump (>0.5 million LOC)

CVE-2018-8392



Journal Special Issue on Fuzzing: What about Preregistration?

22 Apr 2021

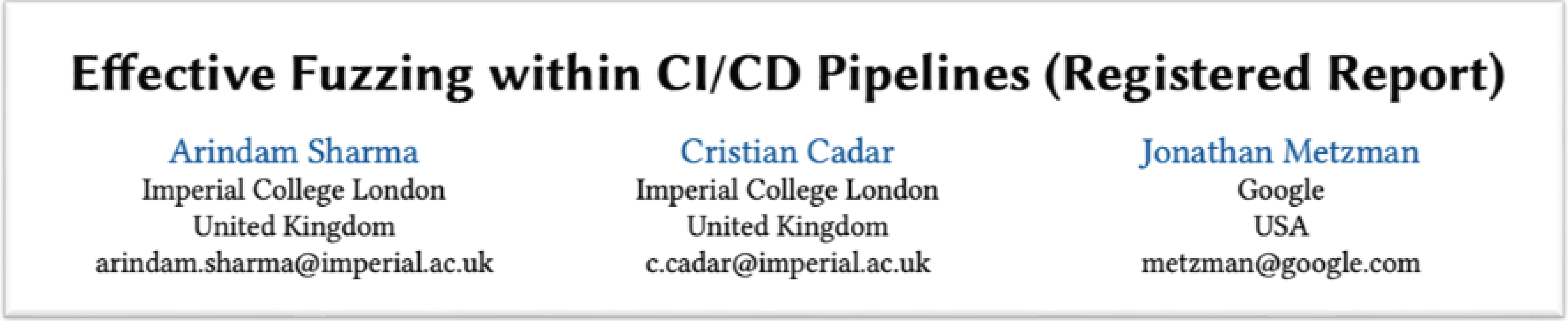
co-authored by Marcel Böhme (Monash University), László Szekeres (Google), Baishakhi Ray (Columbia University), Cristian Cadar (Imperial College London)

Time-to-Exposure (TTE)

AFLGo		
Distance	**Fuzzing**	**Total**
34 min	4 min	38 min

Pazzer (non-incremental)		
Distance	**Fuzzing**	**Total**
< 3 min	< 5 min	7 min

Pazzer (incremental)		
Distance	**Fuzzing**	**Total**
14 sec	< 5 min	5 min



Effective Fuzzing within CI/CD Pipelines (Registered Report)

Arindam Sharma
Imperial College London
United Kingdom
arindam.sharma@imperial.ac.uk

Cristian Cadar
Imperial College London
United Kingdom
c.cadar@imperial.ac.uk

Jonathan Metzman
Google
USA
metzman@google.com

Dynamic Symbolic Execution (DSE)

Program analysis technique for ***automatically exploring paths*** through a program

Applications in:

- Bug finding
- Test generation
- Vulnerability detection and exploitation
- Equivalence checking
- Debugging
- Program repair
- Bounded verification
- etc. etc.

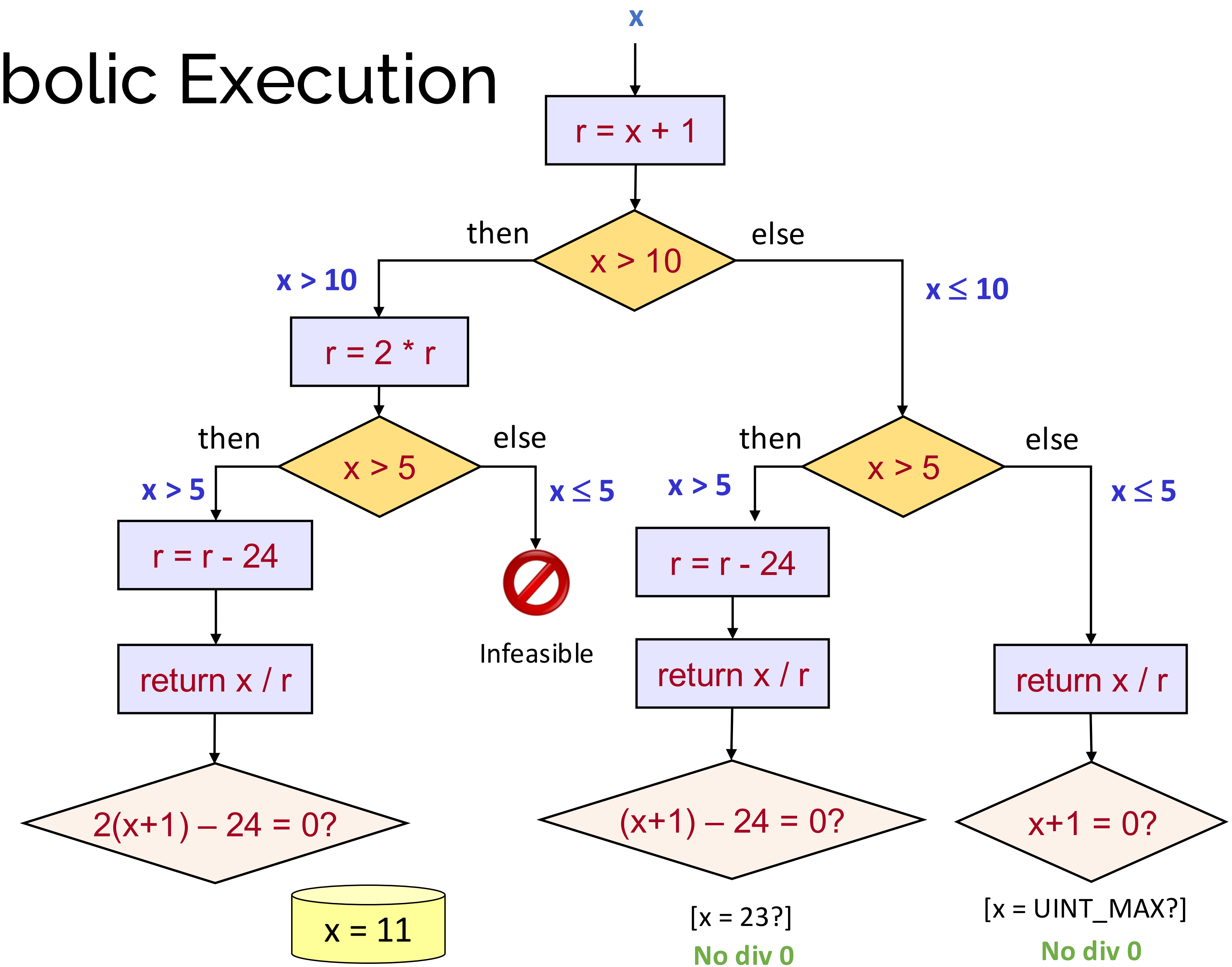
Dynamic Symbolic Execution

```
int foo(unsigned x) {
  int r = x + 1;

  if (x > 10)
    r = 2 * r;

  if (x > 5)
    r = r - 24;

  return x / r;
}
```



Dynamic Symbolic Execution

Key advantages:

- Systematically explores unique control-flow paths
- Produces test cases
- No false positives

- Reasons about all possible values on each explored path
- Per-path verification

Key challenges:

- Efficiently solving lots of constraints
- Path explosion, particularly in the presence of loops

https://klee-se.org/
https://github.com/klee/

Popular dynamic symbolic executor primarily developed and maintained at Imperial

Works at the LLVM level: C (full support), C++, Rust

Active user and developer base:

- 100+ contributors to KLEE and its subprojects
- 400+ mailing list subscribers
- 600+ forks
- 2500+ stars
- 400+ participants across the first four KLEE workshops

4th International KLEE Workshop on Symbolic Execution

15–16 April 2024 • Lisbon, Portugal • Co-located with ICSE 2024

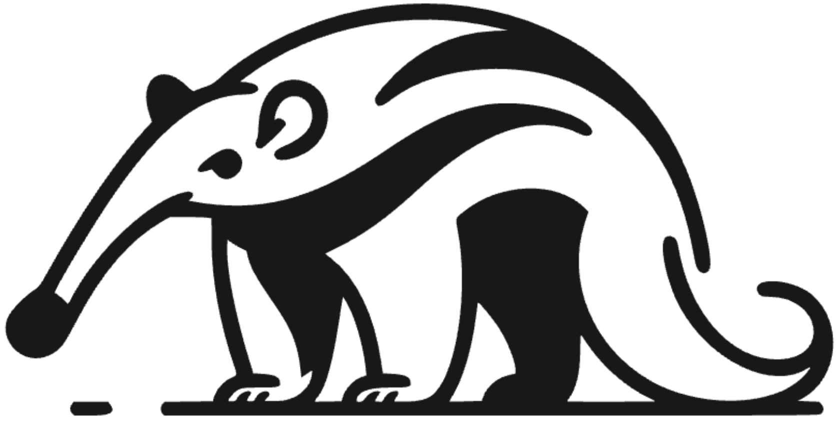
https://klee-se.org/
https://github.com/klee/

Academic impact:

- ACM SIGOPS Hall of Fame Award and ACM CCS Test of Time Award
- Over 4,500 citations to original KLEE paper (OSDI 2008)
- From many different research communities: testing, verification, systems, software engineering, PL, security, etc.
- Many different systems using KLEE: AEG, Angelix , BugRedux , Cloud9, GKLEE, KleeNet, KLEE-UC, S2E, SemFix, etc.

Growing impact in industry:

- **Baidu**: [KLEE 2018]
- **Fujitsu**: [PPoPP 2012], [CAV 2013], [ICST 2015], [IEEE Software 2017], [KLEE 2018]
- **Google**: [2x KLEE 2021]
- **Hitachi**: [CPSNA 2014], [ISPA 2015], [EUC 2016], [KLEE 2021]
- **Intel**: [WOOT 2015]
- **NASA Ames**: [NFM 2014]
- **Samsung**: 2 x [KLEE 2018], [KLEE 2024]
- **Trail of Bits** [blog.trailofbits.com/]
- **etc.**



KLOVO

...eats your bugs!

DSE for Evolving Software

Direct DSE Effort Toward Testing the Change

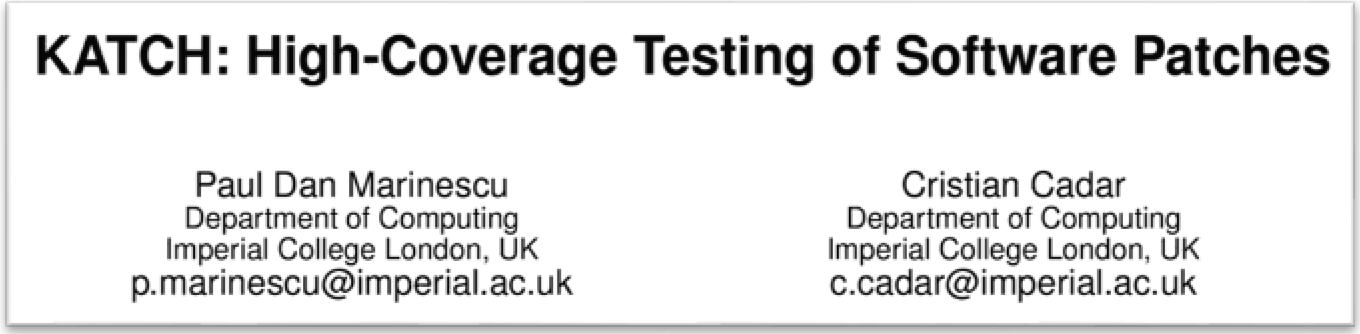
1. Use distance estimates to favour paths close to the change
2. Prioritise paths that explore the changes in behaviour

KLEE for Evolving Software

KATCH = KLEE + PATCH

FINDING BUGS WITH STYLE

- Use distance estimates to the patch guide path exploration
- Use constraint and program analysis to smartly backtrack, when exploration cannot make progress toward the patch

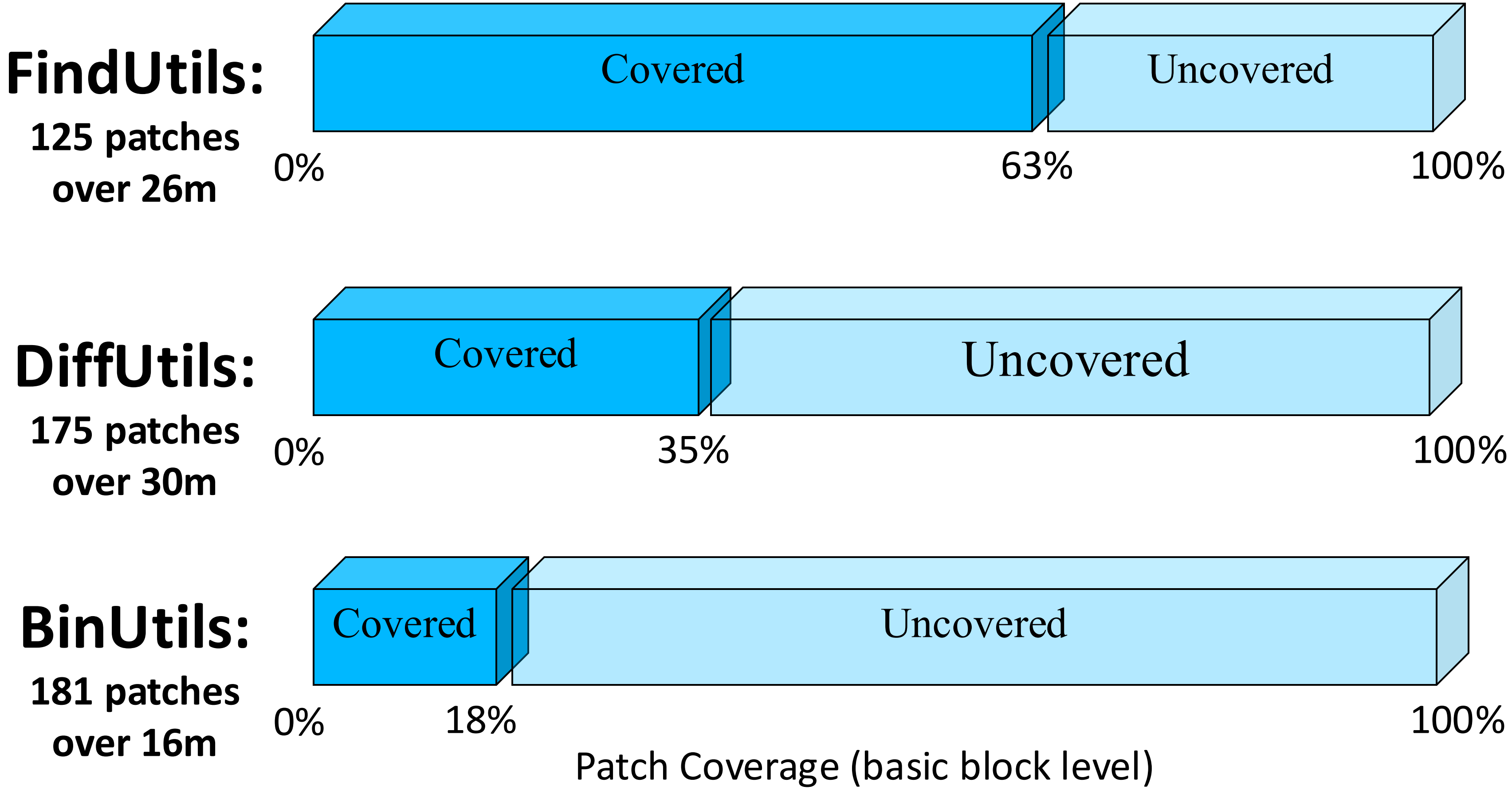


KATCH: High-Coverage Testing of Software Patches

Paul Dan Marinescu
Department of Computing
Imperial College London, UK
p.marinescu@imperial.ac.uk

Cristian Cadar
Department of Computing
Imperial College London, UK
c.cadar@imperial.ac.uk

Developers' Patch Testing



KATCH Patch Testing

FindUtils:

125 patches over 26m

10min/BB

DiffUtils:

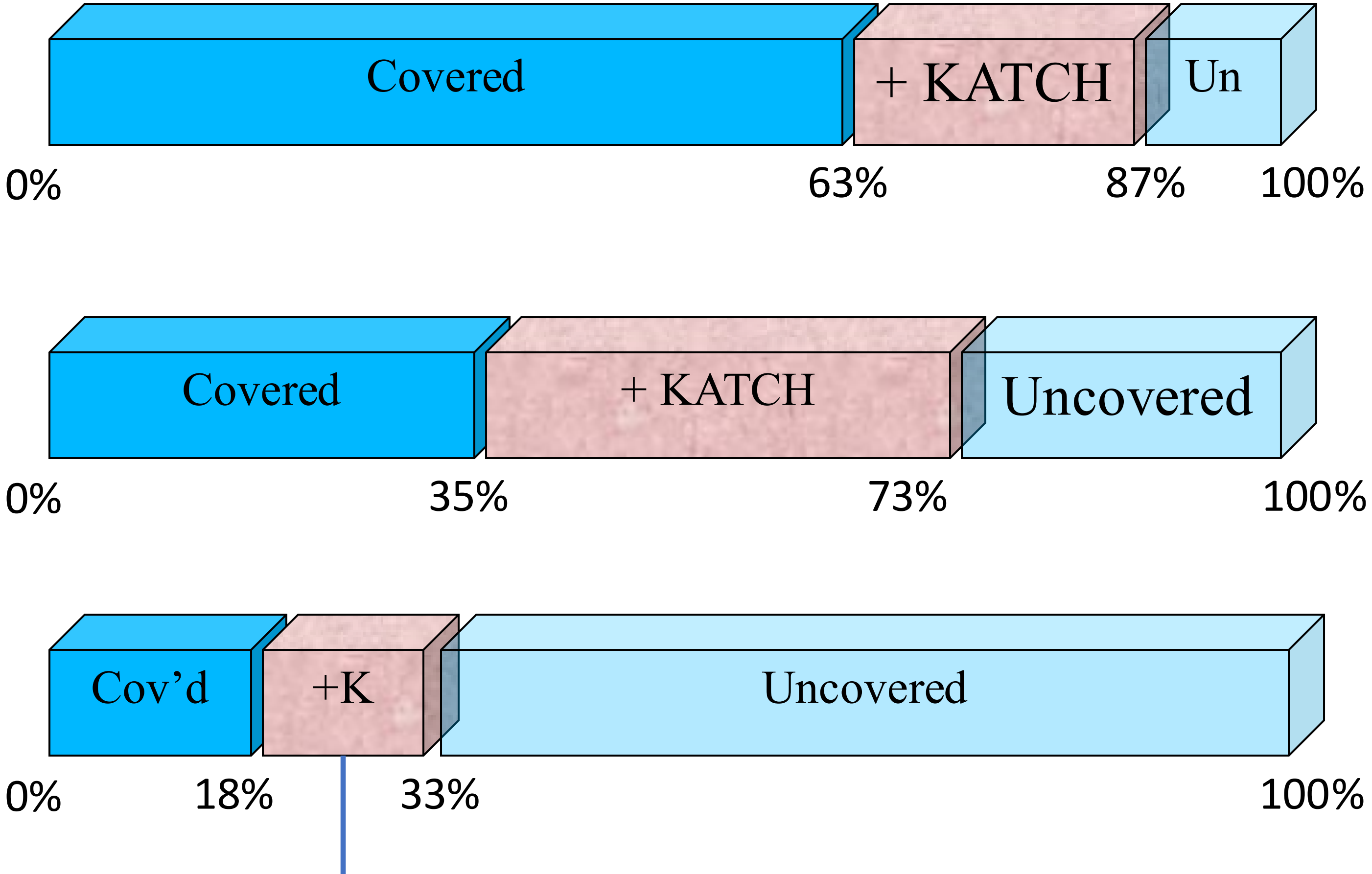
175 patches over 30m

10min/BB

BinUtils:

181 patches over 16m

15min/BB



14 distinct crash bugs

(12 still present and fixed, 10 related to patches)

Reaching the Patch is Not Sufficient

Consider the patch:

Previous

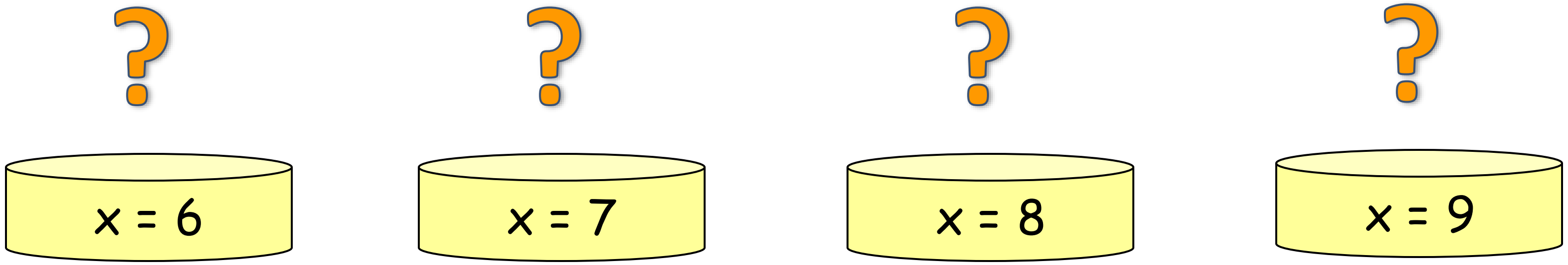
```
if (x % 2 == 0)
     . . .
```

No further uses of x

Current

```
if (x % 3 == 0)
     . . .
```

No further uses of x



Reaching the Patch is Not Sufficient

Consider the patch:

Previous

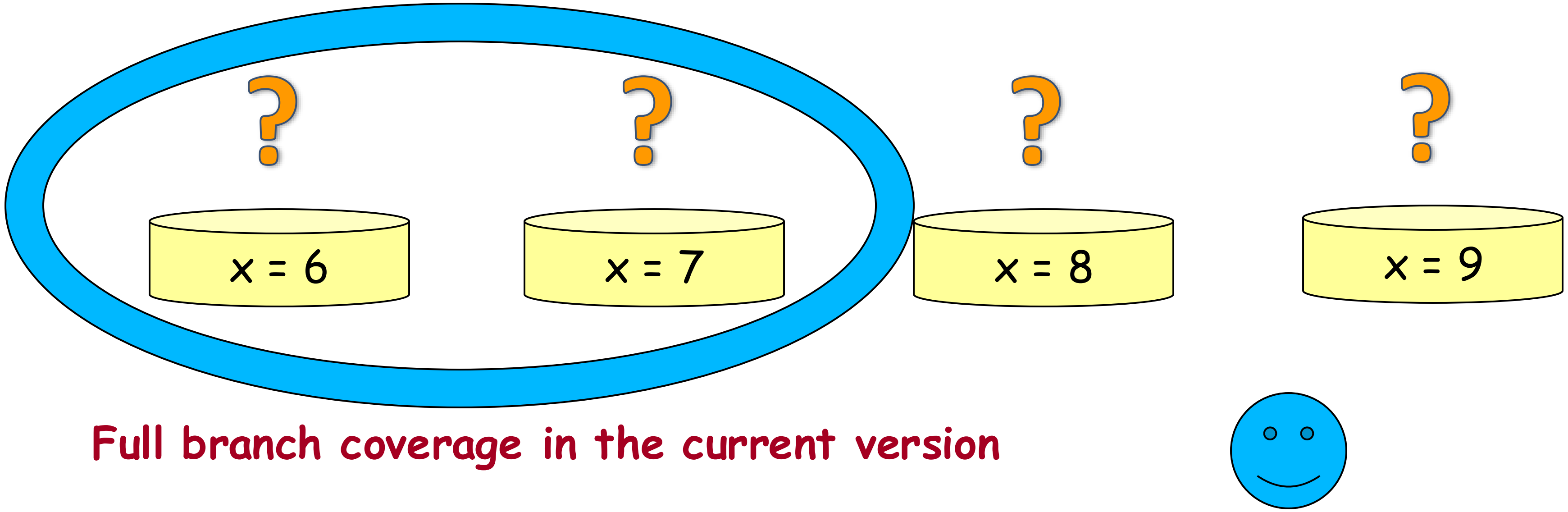
```
if (x % 2 == 0)
     . . .
```

No further uses of x

Current

```
if (x % 3 == 0)
     . . .
```

No further uses of x



Full branch coverage in the current version

Reaching the Patch is Not Sufficient

Consider the patch:

Previous

```
if (x % 2 == 0)
   . . .
```

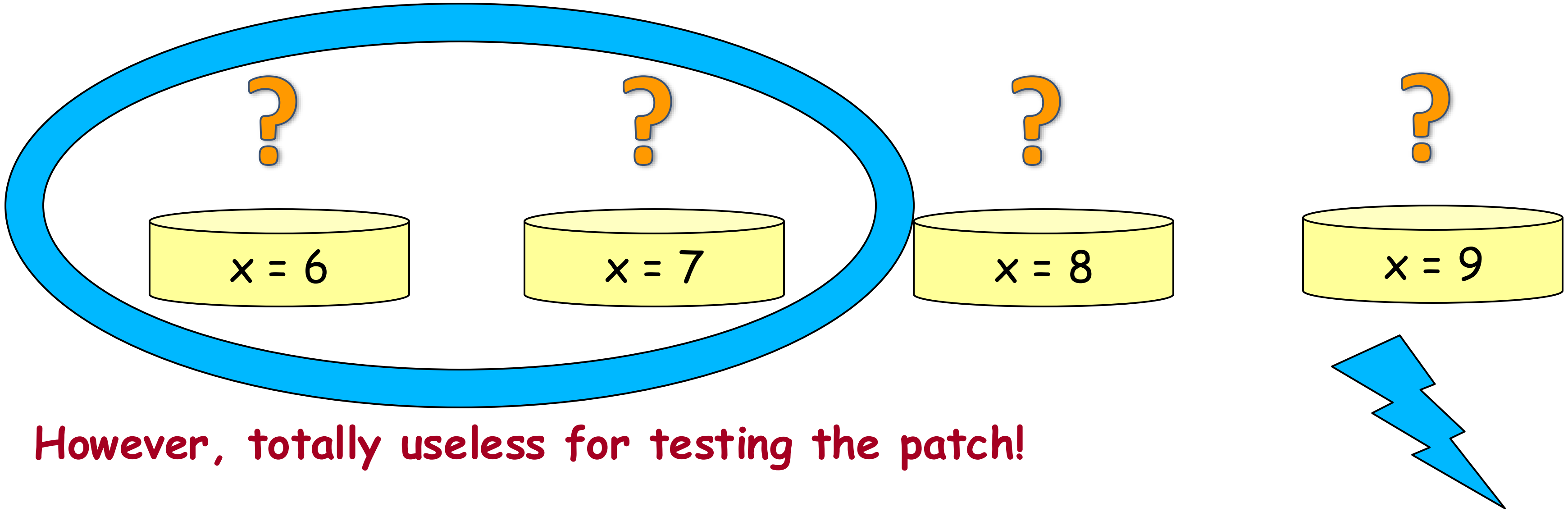
No further uses of x

→

Current

```
if (x % 3 == 0)
   . . .
```

No further uses of x



However, totally useless for testing the patch!

Reaching the Patch is Not Sufficient

Consider the patch:

Previous

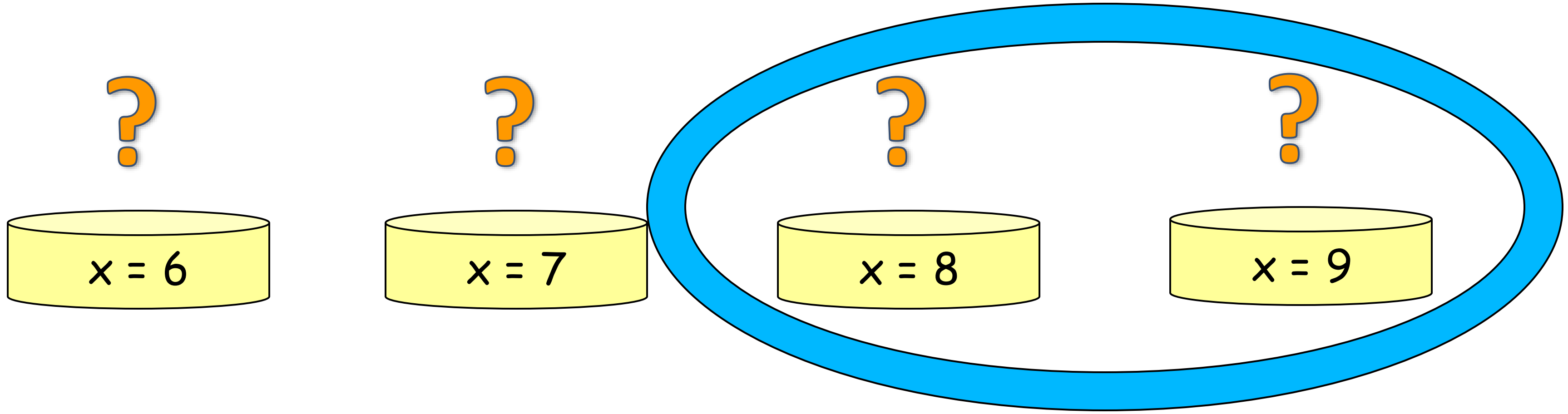
```
if (x % 2 == 0)
      . . .
```

No further uses of x

Current

```
if (x % 3 == 0)
      . . .
```

No further uses of x



x = 6

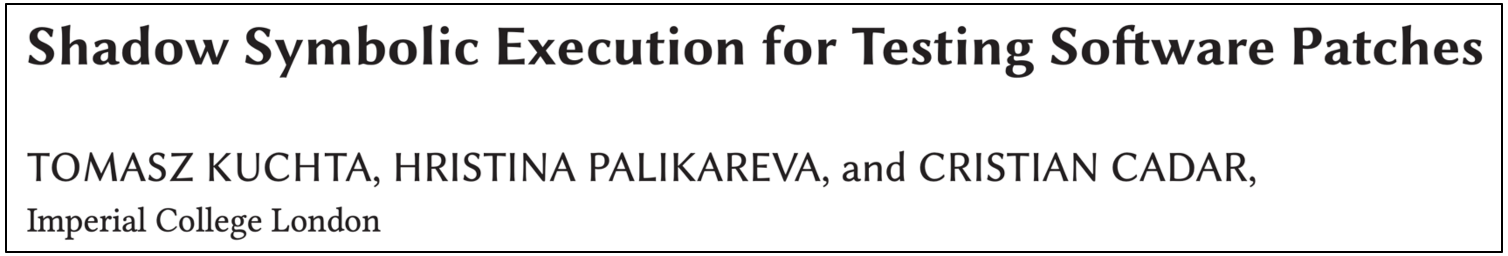
x = 7

x = 8

x = 9

previous → then
current → else

previous → else
current → then

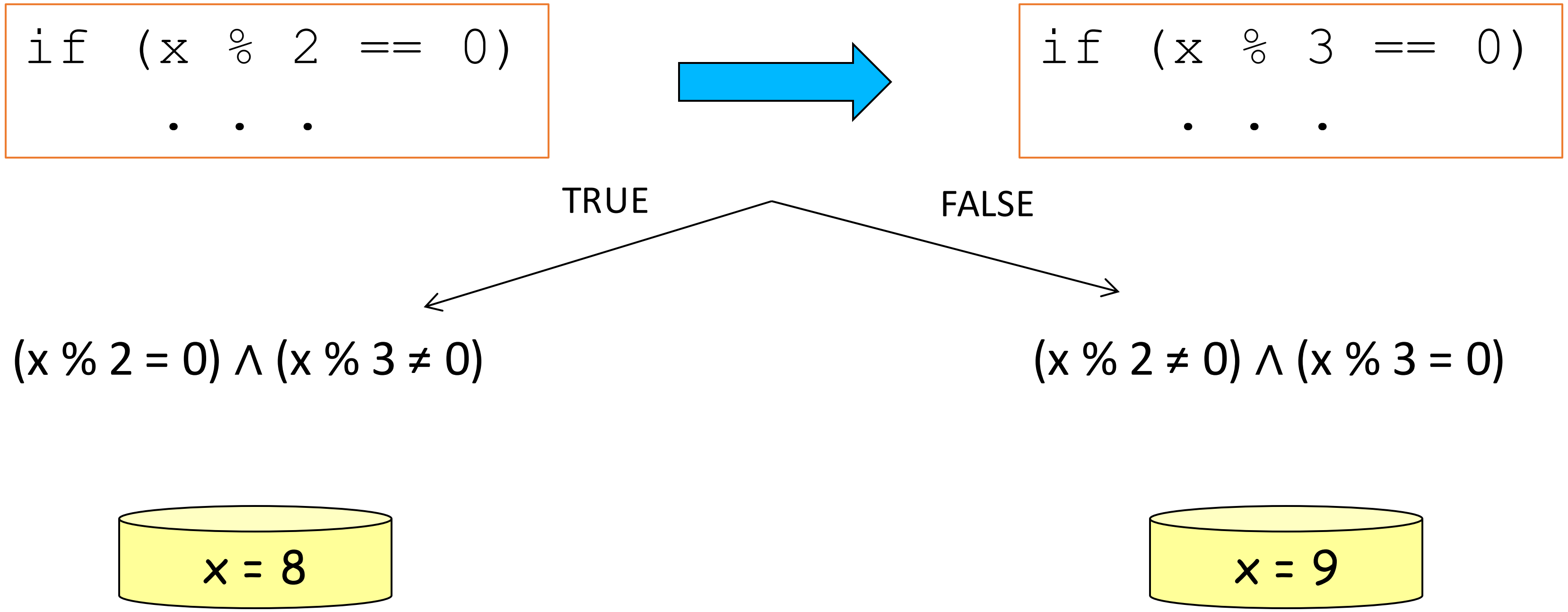


Shadow Symbolic Execution for Testing Software Patches

TOMASZ KUCHTA, HRISTINA PALIKAREVA, and CRISTIAN CADAR,
Imperial College London

Symbolic Execution on Both Versions Concurrently

Previous **Current**



```
if (x % 2 == 0)
    . . .
```

```
if (x % 3 == 0)
    . . .
```

$(x \% 2 = 0) \land (x \% 3 \neq 0)$

$(x \% 2 \neq 0) \land (x \% 3 = 0)$

x = 8

x = 9

Shadow Symbolic Execution

- Can prune large parts of the search space, for which the two versions behave identically
- Provides the ability to simplify path constraints
- Is memory-efficient by sharing large parts of the symbolic constraints
- Does not execute unchanged computations twice

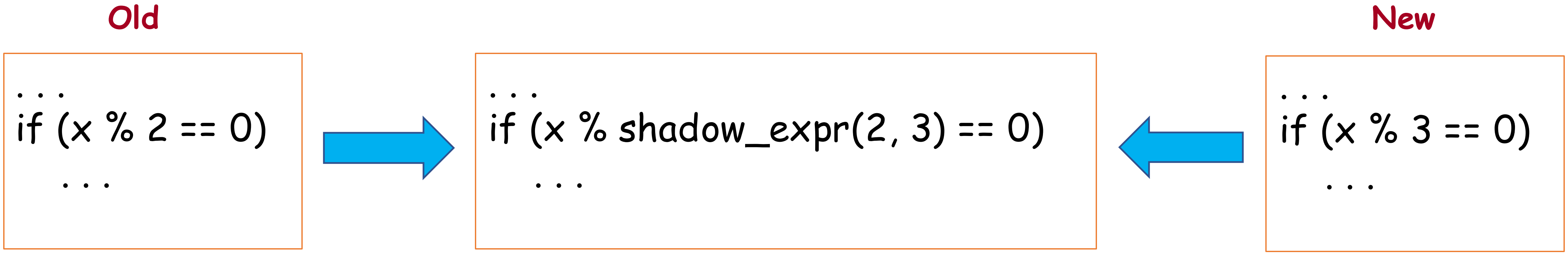
Case Study: cut

Input	**Old**	**New**
cut –c1-3,8- -output-d=: file (file is “abcdefg”)	abc	abc + *buffer overflow*
cut -c1-7,8- --output-d=: file file contains “abcdefg”	abcdef	abcdef + *buffer overflow*
cut -b0-2,2- --output-d=: file file contains “abc”	abc	signal abort
cut -s -d: -f0- file (file is “:::\n:1”)	:::\n:1	\n\n
cut –d: -f1,0- file (file is “a:b:c”)	a:b:c	a

Need for specifications!

Test cases as documentation!

Challenge: Joining the Two Versions



Old

```
. . .
if (x % 2 == 0)
    . . .
```

```
. . .
if (x % shadow_expr(2, 3) == 0)
    . . .
```

New

```
. . .
if (x % 3 == 0)
    . . .
```

Product Programs

Used to reason about hyperproperties in a security context

- Particularly non-interference
- Product program of program P with itself

G. Barthe, J. M. Crespo, C. Kunz, “Relational verification using product programs”
Proc. of the 17th International Symposium on Formal Methods (FM’11)

We use them as a mechanism for merging multiple program versions into a single program

Example

Previous version

```
x = y - 1;
z = x / 4;
```

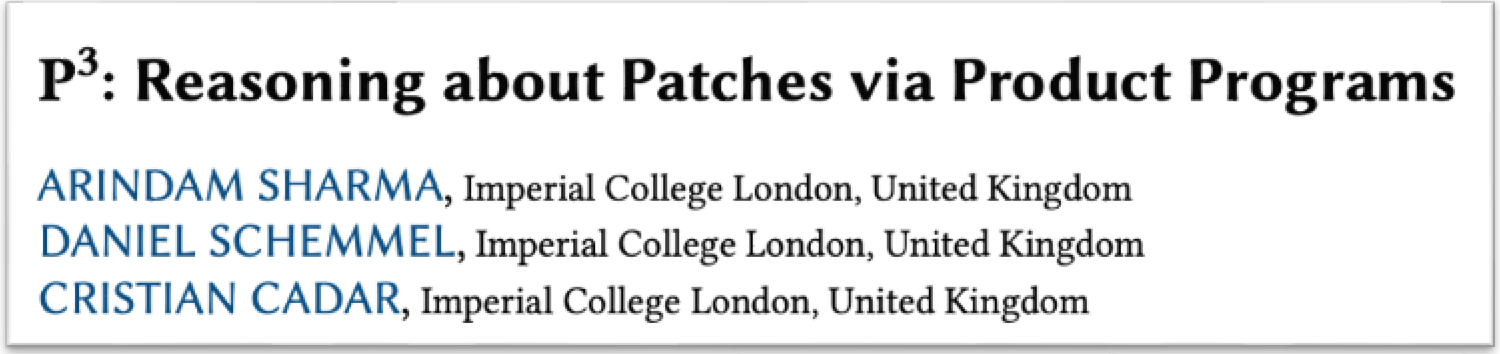
Current version

```
x = y - 1;
z = x >> 2;
```

Product program

```
x_prev = y_prev - 1;
x = y - 1;

z_prev = x_prev / 4;
z = x >> 2;
```



P^3: Reasoning about Patches via Product Programs

ARINDAM SHARMA, Imperial College London, United Kingdom
DANIEL SCHEMMEL, Imperial College London, United Kingdom
CRISTIAN CADAR, Imperial College London, United Kingdom

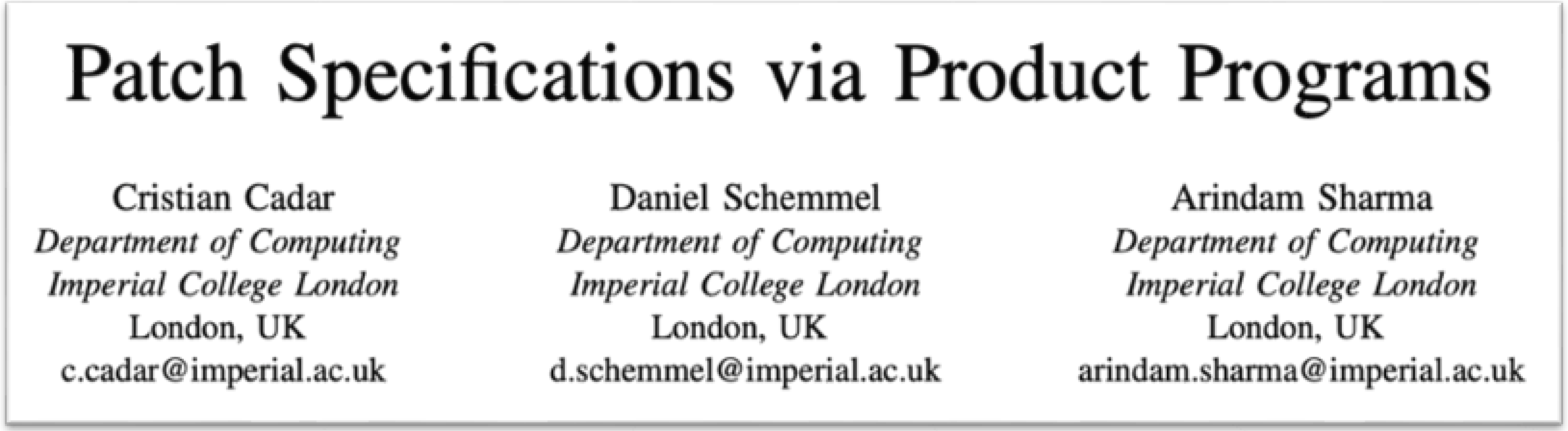
- Designed P^3 to generate product programs for real-world C code and *different* program versions
- P^3 can transform ANY program analyser into a differential program analyser
- We were able to find the all the bugs found via shadow symbolic execution using P^3 + KLEE
- We found different bugs using P^3 + AFL++

P^3

KLEE

AFL++

KLEE
FINDING BUGS WITH STYLE



Patch Specifications via Product Programs

Cristian Cadar
Department of Computing
Imperial College London
London, UK
c.cadar@imperial.ac.uk

Daniel Schemmel
Department of Computing
Imperial College London
London, UK
d.schemmel@imperial.ac.uk

Arindam Sharma
Department of Computing
Imperial College London
London, UK
arindam.sharma@imperial.ac.uk

Specifications encoding cross-patch properties

```
x_prev = y_prev - 1;
x = y - 1;

z_prev = x_prev / 4;
z = x >> 2;

assert(z == z_prev);
```

Preliminary Experience

- Wrote patch specs for several patches from CoreBench: collection of complex real-world patches [Böhme and Roychoudhury]
- We used P^3 with AFL++ and KLEE to look for violations of the patch specs

P^3
KLEE
FINDING BUGS WITH STYLE
KLEE
AFL++

Patch in ls

```
static char * make_link_name (char const *name,
                                                 char const *linkname);


make_link_name(”A/B/f.txt", "g.txt") = ”A/B/g.txt"
```

“Do not hard-code ’/’. Use IS_ABSOLUTE_FILE_NAME and dir_len instead. Use stpcpy/stpncpy in place of strncpy/strcpy.”

Patch in ls

P³ with both AFL++ and KLEE found a spec violation:

```
name = /a
linkname = x
```

```
if (*linkname == '/')
   return xstrdup (linkname);
char const *linkbuf = strrchr (name, '/’);
if (linkbuf == NULL)
   return xstrdup (linkname);
size_t bufsiz = linkbuf - name + 1;
char *p = xmalloc (bufsiz + strlen (linkname) + 1);
strncpy (p, name, bufsiz);
strcpy (p + bufsiz, linkname);
return p;
```

Bug made it into a release, was reported by a user and fixed

```
if (IS_ABSOLUTE_FILE_NAME (linkname))
   return xstrdup (linkname);
size_t prefix_len = dir_len (name);
if (prefix_len == 0)
   return xstrdup (linkname);
char *p = xmalloc (prefix_len + 1 + strlen (linkname) + 1);
stpcpy (stpncpy (p, name, prefix_len + 1), linkname);


return p;
assert( strcmp(p, p_prev) == 0 );
```

Patch in ls

P^3 with both AFL++ and KLEE found new spec violation:

```
name = /x//y
linkname = a
```

```
if (*linkname == '/')
   return xstrdup (linkname);
char const *linkbuf = strrchr (name, '/’);
if (linkbuf == NULL)
   return xstrdup (linkname);
size_t bufsiz = linkbuf - name + 1;
char *p = xmalloc (bufsiz + strlen (linkname) + 1);
strncpy (p, name, bufsiz);
strcpy (p + bufsiz, linkname);
return p;
```

```
if (IS_ABSOLUTE_FILE_NAME (linkname))
   return xstrdup (linkname);
size_t prefix_len = dir_len (name);
if (prefix_len == 0)
   return xstrdup (linkname);
char *p = xmalloc (prefix_len + 1 + strlen (linkname) + 1);
stpcpy (stpncpy (p, name, prefix_len + 1), linkname);
 if ( ! ISSLASH (name[prefix_len - 1]))  ++prefix_len;
 stpcpy (stpncpy (p, name, prefix_len), linkname);
return p;
assert( strcmp(p, p_prev) == 0 );
```

Code patch to fix reported bug

Patch in ls

No more spec violations found if path-based equality is used

```
if (*linkname == '/')
   return xstrdup (linkname);
char const *linkbuf = strrchr (name, '/’);
if (linkbuf == NULL)
   return xstrdup (linkname);
size_t bufsiz = linkbuf - name + 1;
char *p = xmalloc (bufsiz + strlen (linkname) + 1);
strncpy (p, name, bufsiz);
strcpy (p + bufsiz, linkname);
return p;
```

```
if (IS_ABSOLUTE_FILE_NAME (linkname))
   return xstrdup (linkname);
size_t prefix_len = dir_len (name);
if (prefix_len == 0)
   return xstrdup (linkname);
char *p = xmalloc (prefix_len + 1 + strlen (linkname) + 1);
stpcpy (stpncpy (p, name, prefix_len + 1), linkname);
if ( ! ISSLASH (name[prefix_len - 1]))  ++prefix_len;
stpcpy (stpncpy (p, name, prefix_len), linkname);
return p;
 assert( patheq(p, p_prev) == 0 );
```

Additional Directions

- Pruning paths that are unrelated to the change
  [Trabish et al, ICSE 2018], [Trabish et al, ESEC/FSE 2020]
- Generating test drivers to start close to the change using program analysis and LLMs
  [Zaki et al, SANER 2025], ongoing work

Program Analysis for Safe and Secure Software Evolution

Cristian Cadar

SOFTWARE RELIABILITY GROUP
Imperial College London

Funded by UKRI Engineering and Physical Sciences Research Council erc

University of Stuttgart
Stuttgart, Germany
15 April 2025

Evolving Software

- Poorly validated code changes often introduce bugs & vulnerabilities
- Some with catastrophic impact

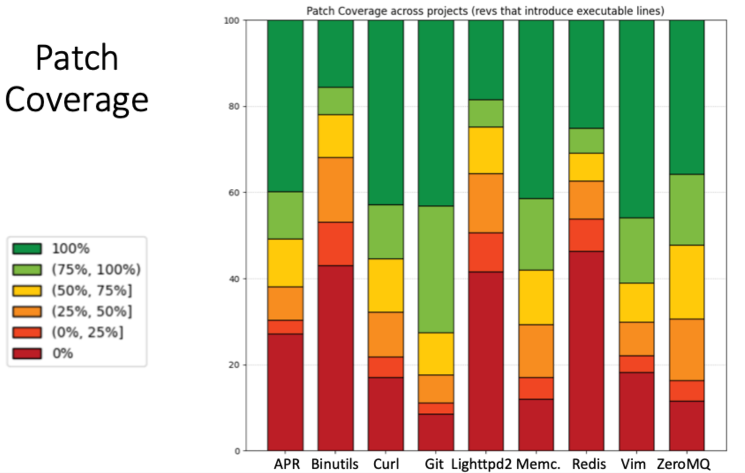
Heartbleed (2014)

Shellshock (2014)

Stagefright (2016)

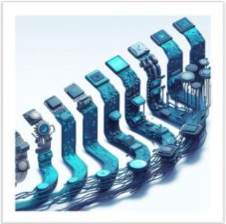
Crowdstrike (2024)

Patch Coverage



Testing Evolving Software

Reuse testing results of previous versions



Direct testing effort toward the changes

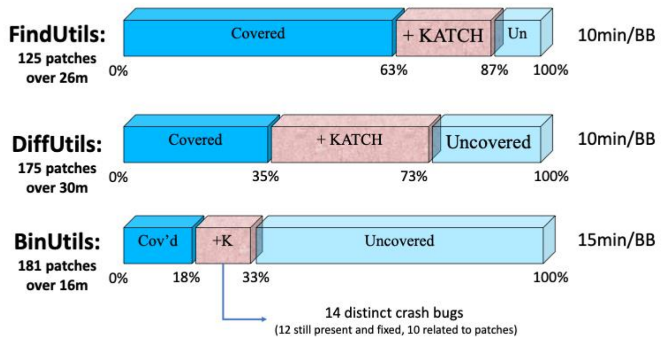
PaZZER = Patch + Fuzzer

- Designed to be practical for short CI/CD runs
- Aims to find a sweet spot between time spent in distance computation and effectiveness
- Relies on less precise but quick distance estimates (using only the call graph)
- Computes distances incrementally (LPA*, Anytime-D*)

IMPERIAL

Google

KATCH Patch Testing



14 distinct crash bugs
(12 still present and fixed, 10 related to patches)

Reaching the Patch is Not Sufficient

Consider the patch:

Old

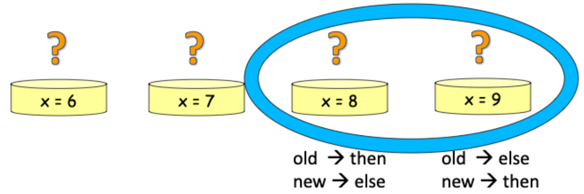
```
if (x % 2 == 0)
  . . .
```

No further uses of x

New

```
if (x % 3 == 0)
  . . .
```

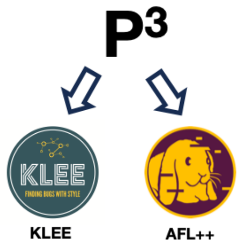
No further uses of x



P³: Reasoning about Patches via Product Programs

ARINDAM SHARMA, Imperial College London, United Kingdom
DANIEL SCHEMMEL, Imperial College London, United Kingdom
CRISTIAN CADAR, Imperial College London, United Kingdom

- Designed P³ to generate product programs for real-world C code and *different* program versions
- P³ can transform ANY program analyser into a differential program analyser
- We were able to find the all the bugs found via shadow symbolic execution using P³ + KLEE
- We found different bugs using P³ + AFL++



Patch Specifications via Product Programs

Cristian Cadar
Department of Computing
Imperial College London
London, UK
c.cadar@imperial.ac.uk

Daniel Schemmel
Department of Computing
Imperial College London
London, UK
d.schemmel@imperial.ac.uk

Arindam Sharma
Department of Computing
Imperial College London
London, UK
arindam.sharma@imperial.ac.uk

Specifications encoding cross-patch properties

```
x_prev = y_prev - 1;
x = y - 1;

z_prev = x_prev / 4;
z = x >> 2;

assert(z == z_prev);
```