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

Thomas Bailey Cristian Cadar

SOFTWARE RELIABILITY GROUP

Imperial College London

Funded by

European Research Council
Established by the European Commission

ICST 2025
4 April 2025
Napoli, Italy

How Are Mature C/C++ Software Projects, Tested By Developers?

ISSTA 2014

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

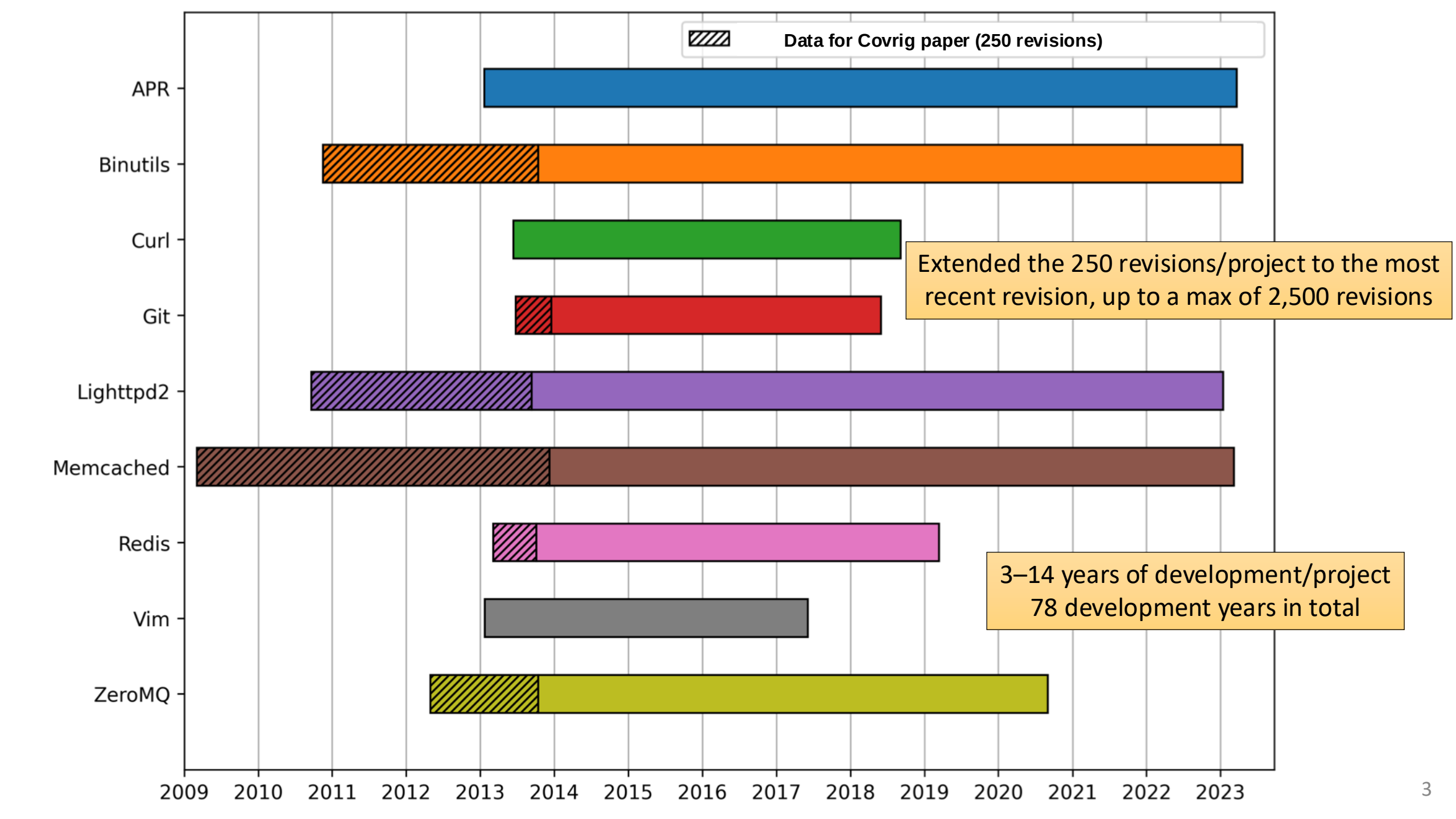
Paul Marinescu, Petr Hosek, Cristian Cadar
Department of Computing
Imperial College London, UK
{p.marinescu,p.hosek,c.cadar}@imperial.ac.uk

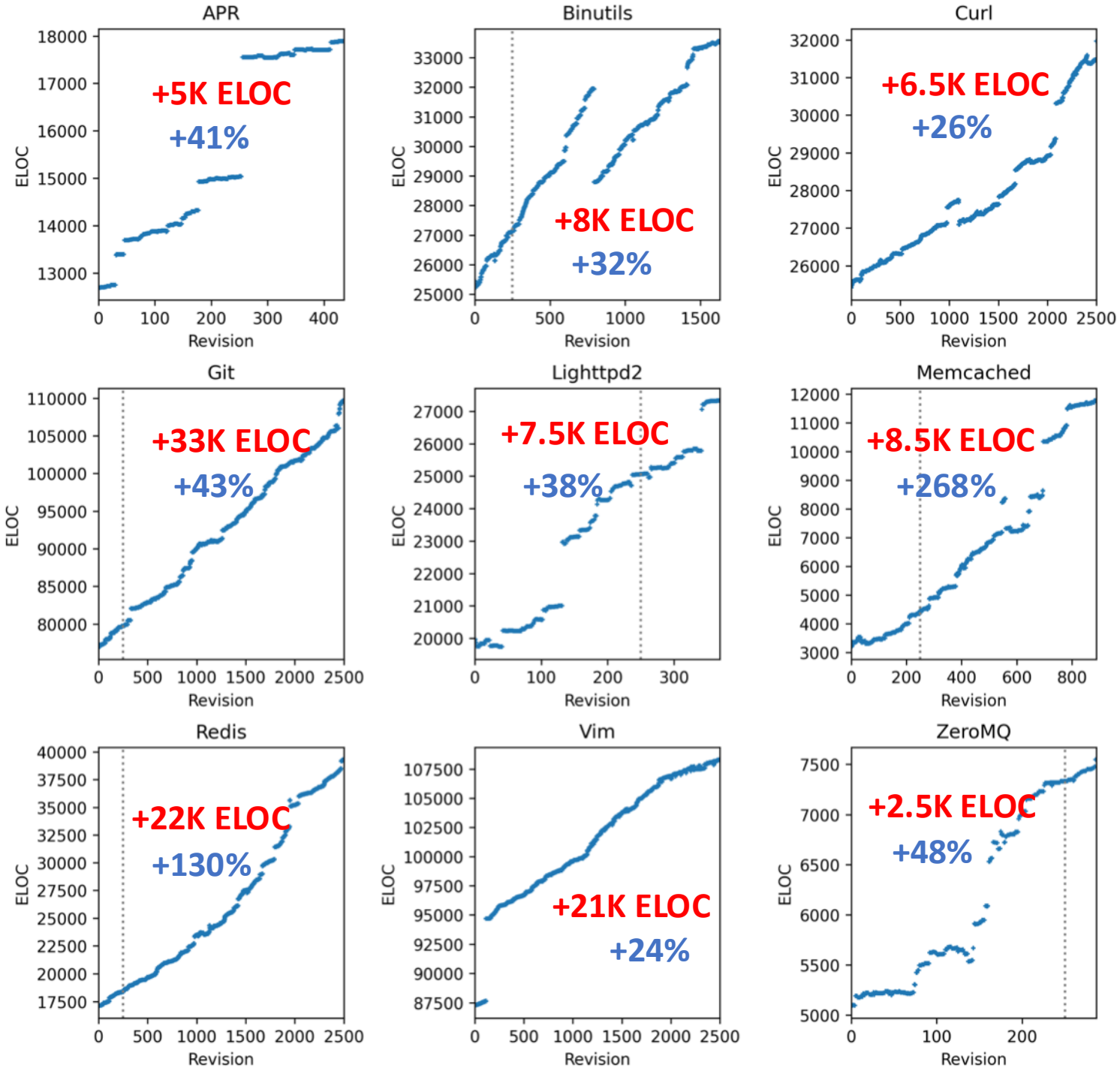
- 6 popular C/C++ open-source projects
- Analysed 250 revisions per project
- Conclusion: LOTS of code added or modified without being tested

A Decade Later

Software engineering has seen many advances

How has software testing of mature C/C++ OSS projects changed?

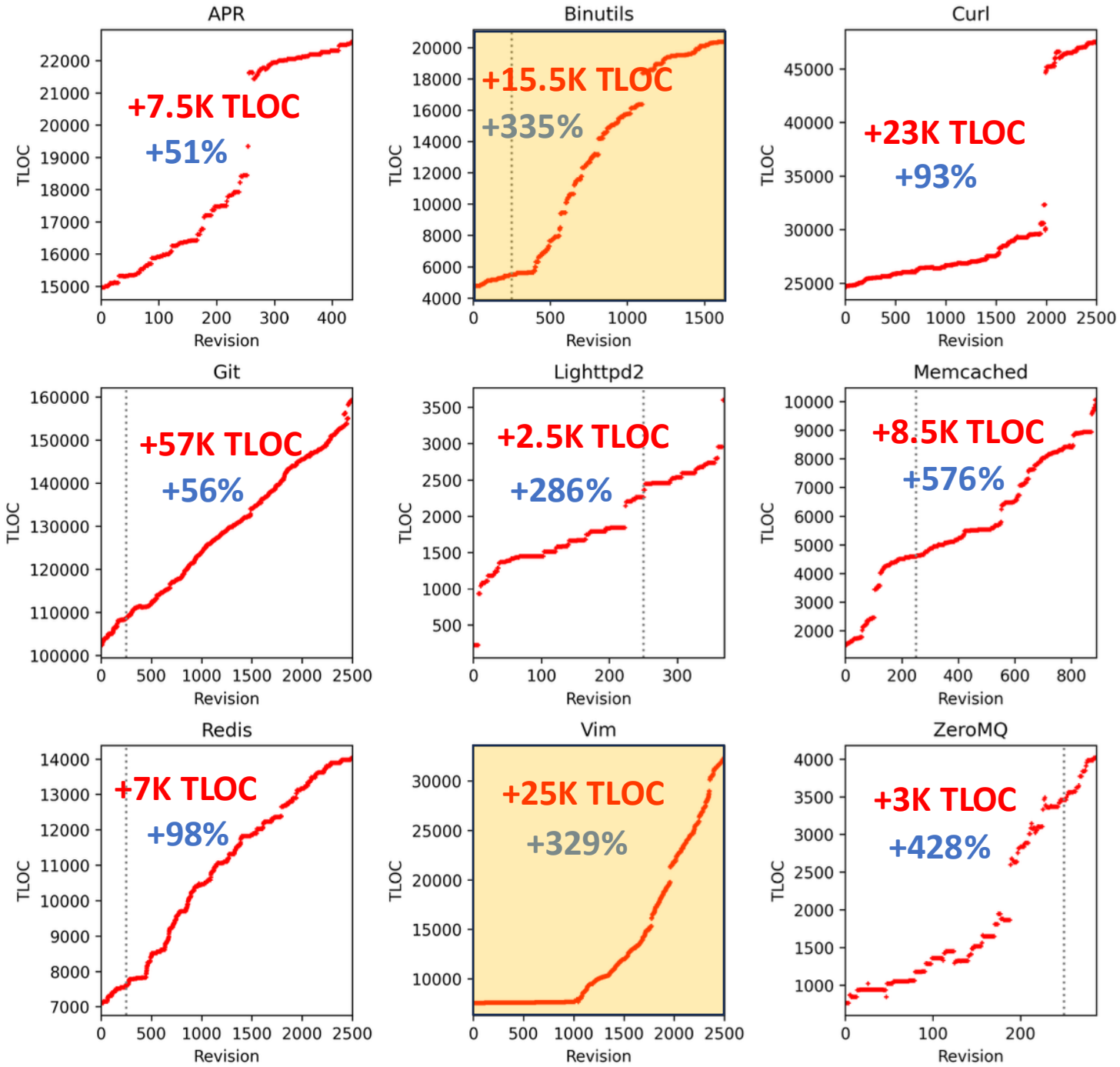

Data for Covrig paper (250 revisions)
APR
Binutils
Curl
Git
Lighttpd2
Memcached
Redis
Vim
ZeroMQ
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
Extended the 250 revisions/project to the most recent revision, up to a max of 2,500 revisions
3–14 years of development/project
78 development years in total



ELOC/time

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

*Rounded to the closest 0.5K



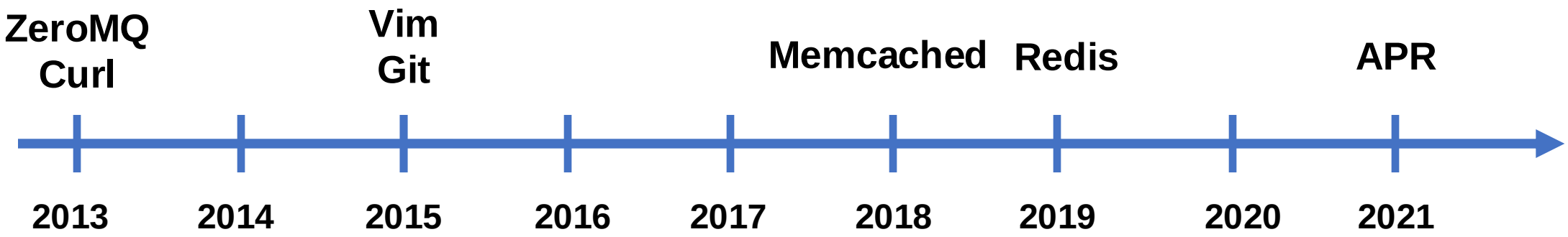
TLOC/time

6/9 projects add MORE TLOC than ELOC

Vim and Binutils experience a step change in their testing efforts

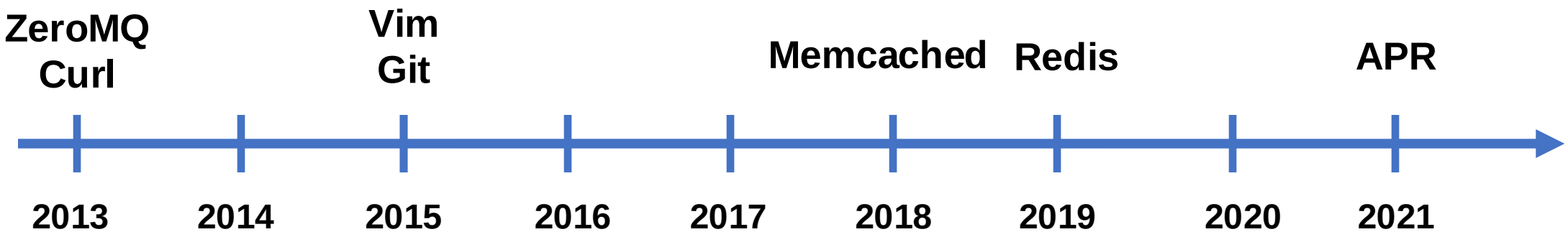
*Rounded to the closest 0.5K

CI Adoption: 7/9 projects now use a CI system



	APR	Binutils	Curl	Git	Lighttpd	MemC.	Redis	Vim	ZeroMQ
First CIs	Travis		Travis Zuul	Travis		Travis	Circle	Travis	Travis
Current CIs	GitHub		GitHub Circle Cirrus AppVeyor Azure	GitHub Azure		GitHub	GitHub	GitHub Cirrus AppVeyor	Travis

CI Adoption: 7/9 projects now use a CI system



	APR	Binutils	Curl	Git	Lighttpd	MemC.	Redis	Vim	ZeroMQ
First CIs	Travis		Travis Zuul	Travis		Travis	Circle	Travis	Travis
Current CIs	GitHub		GitHub Circle Cirrus AppVeyor Azure	GitHub Azure		GitHub	GitHub	GitHub Cirrus AppVeyor	Travis

Coverage Tracking

Only 2/9 projects explicitly track coverage:

- **Curl** and **Vim**, both via **Coveralls**

Challenge: distinguishing b/w problematic and superficial coverage drops

Vim: configured their CI to tolerate coverage decreases of < 0.05%

> "Problem: Codecov reports every little coverage drop. Solution: Tolerate a 0.05% drop." (Vim, 2021)

Curl: dropped Coveralls coverage tracking due to this issue

> "The coveralls service and test coverage numbers are just too unreliable. Removed badge from README.md as well." (Curl, 2019)

Fuzzing Adoption

Google's OSS-Fuzz

- Fuzzing platform for OSS
- Found 36K+ bugs
  in 1K+ C/C++ projects

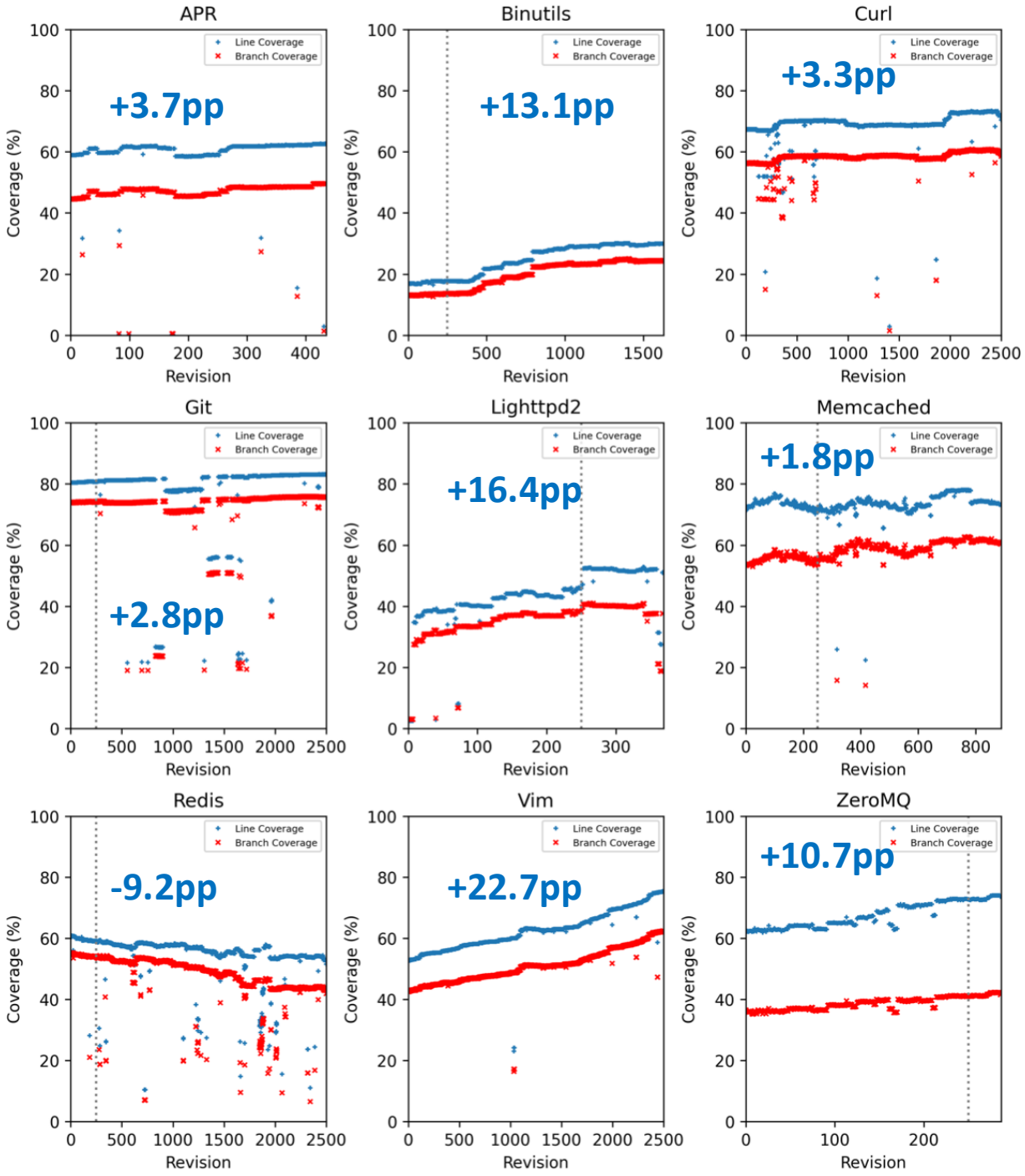
	Line Coverage	Fuzz Targets
APR		
Binutils	32.2%	26
Curl	21.8%	17
Git	10.8%	10
Lighttpd2	34.7%	1
Memcached		
Redis		
Vim		
ZeroMQ		

https://introspector.oss-fuzz.com/, March 2024

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

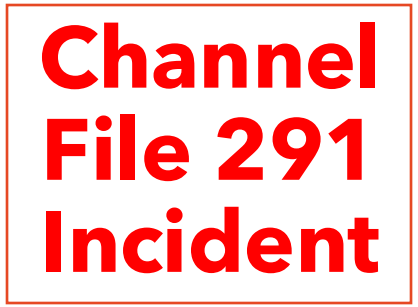
Evolving Software

- Software evolves on a constant basis
- Poorly validated changes (patches) can have a significant impact
- Sometimes catastrophic:

Heartbleed (2014)

Shellshock (2014)

Stagefright (2016)

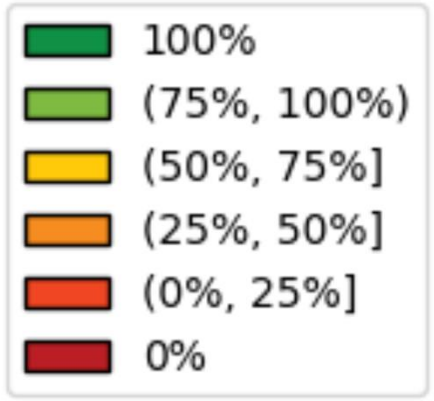


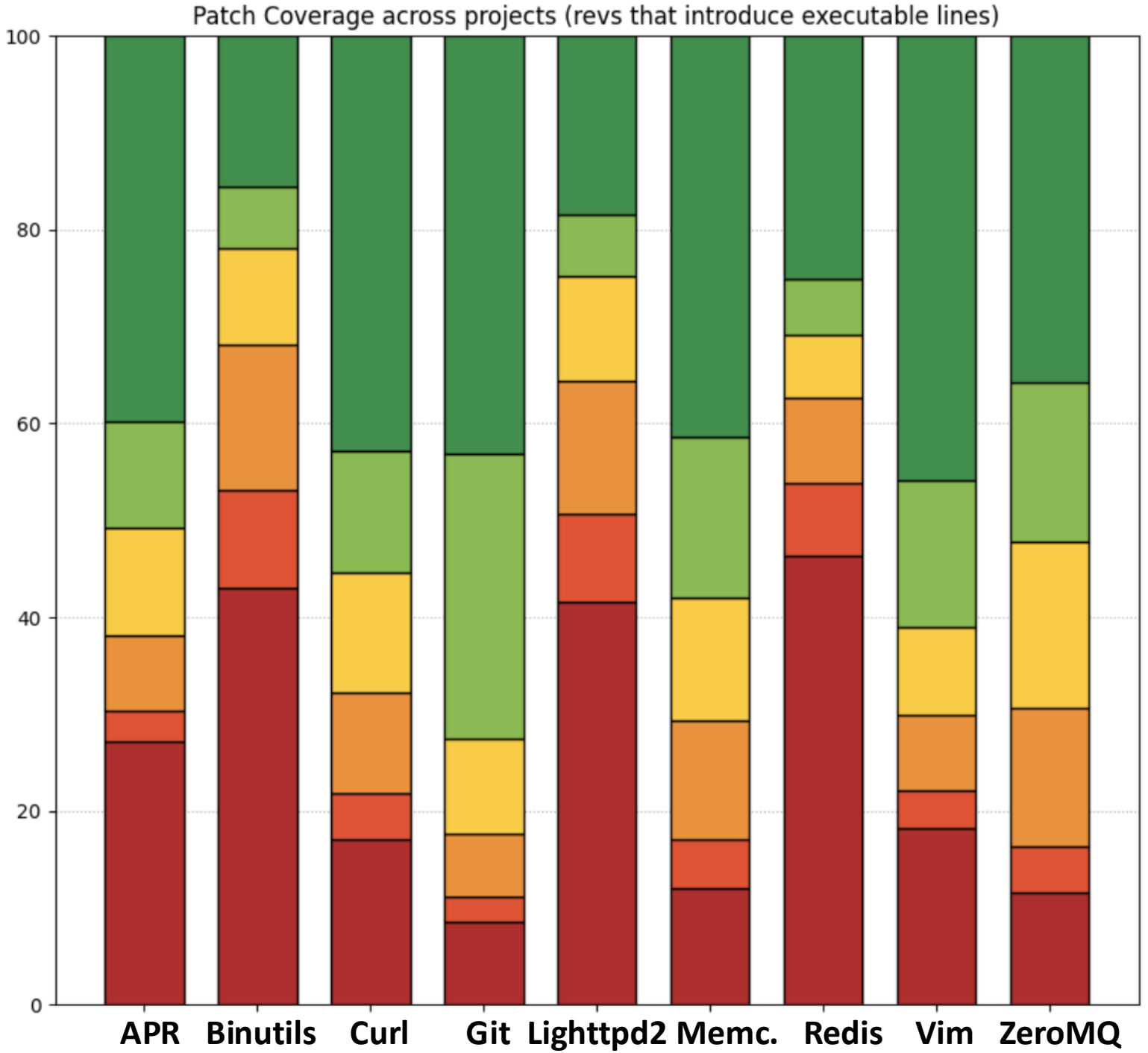
Crowdstrike (2024)

Patch Coverage

Percentage of ELOC in a patch covered by the test suite

Low bar: reaching the patch does not mean testing it





Flaky Revisions

	Covrig [ISSTA 2024]	This study [ICST 2025]
APR	-	0.5%
Binutils	0%	18.6%
Curl	-	4.1%
Git	0.4%	1.7%
Lighttpd2	0.4%	2.7%
Memcached	8.4%	10.9%
Redis	6.4%	59.8%
Vim	-	2.0%
ZeroMQ	12.8%	17.8%

Number of flaky revisions has significantly increased in the last decade, despite research progress in flakiness detection.

Some flaky tests lasts for many revisions until fixed

Conclusion & Actionable Insights

Developers more aware of the importance of testing:

- Most projects now use a CI system
- A few projects track coverage and use fuzzing
- In 6/9 projects, developers have added more TLOC than ELOC
- Overall coverage increases over time in all projects but one

But challenges remain:

- Better standards & solutions to facilitate switching CI, fuzzing, and analysis providers
- We need ways to more accurately track and reason about coverage changes
- We need better automatic techniques for fuzz target generation
- We still need better test generation techniques
  - 5/9 projects still have under 50% branch coverage
- We need targeted testing techniques for code changes
  - The number of untested and poorly tested patches is really high
- We should understand why flakiness has increased over time
  - And how to bridge the gap between research and practice

https://srg.doc.ic.ac.uk/projects/covrig/