ISSTA 24

Revisiting Test-Case Prioritization on Long-Running Test Suites

Runxiang Chen, **Shuai Wang**, Reyhaneh Jabbarvand, Darko Marinov

University of Illinois Urbana-Champaign

ISSTA 2024, Vienna, Austria

09/20/2024

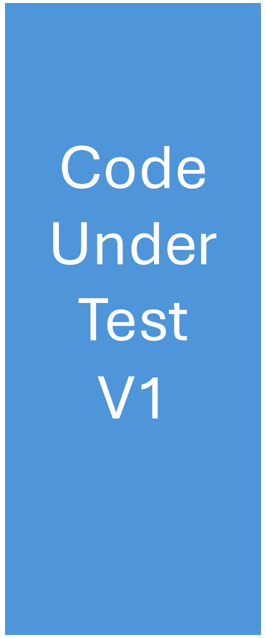
CCF-1763788
CCF-1956374
CNS-2238045

Regression testing for code changes

- Checking that code changes do not break working functionality
- Widely used in modern CI/CD environments

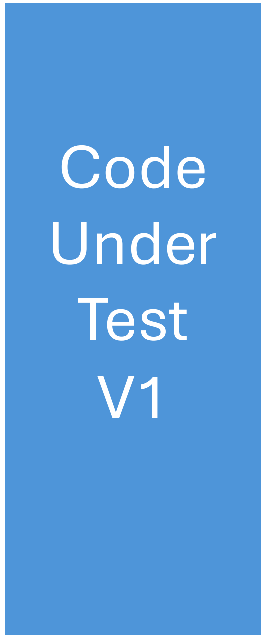
Regression testing for code changes

- Checking that code changes do not break working functionality
- Widely used in modern CI/CD environments



Regression testing for code changes

- Checking that code changes do not break working functionality
- Widely used in modern CI/CD environments

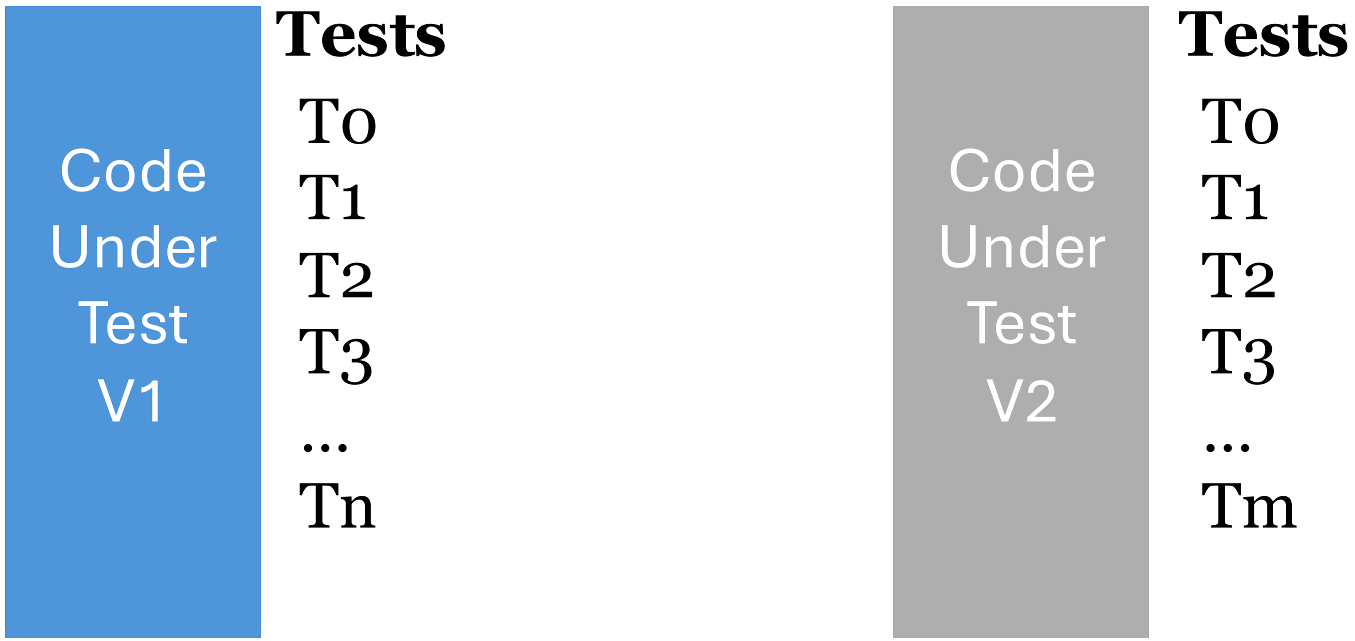


Tests

T0
T1
T2
T3
…
Tn

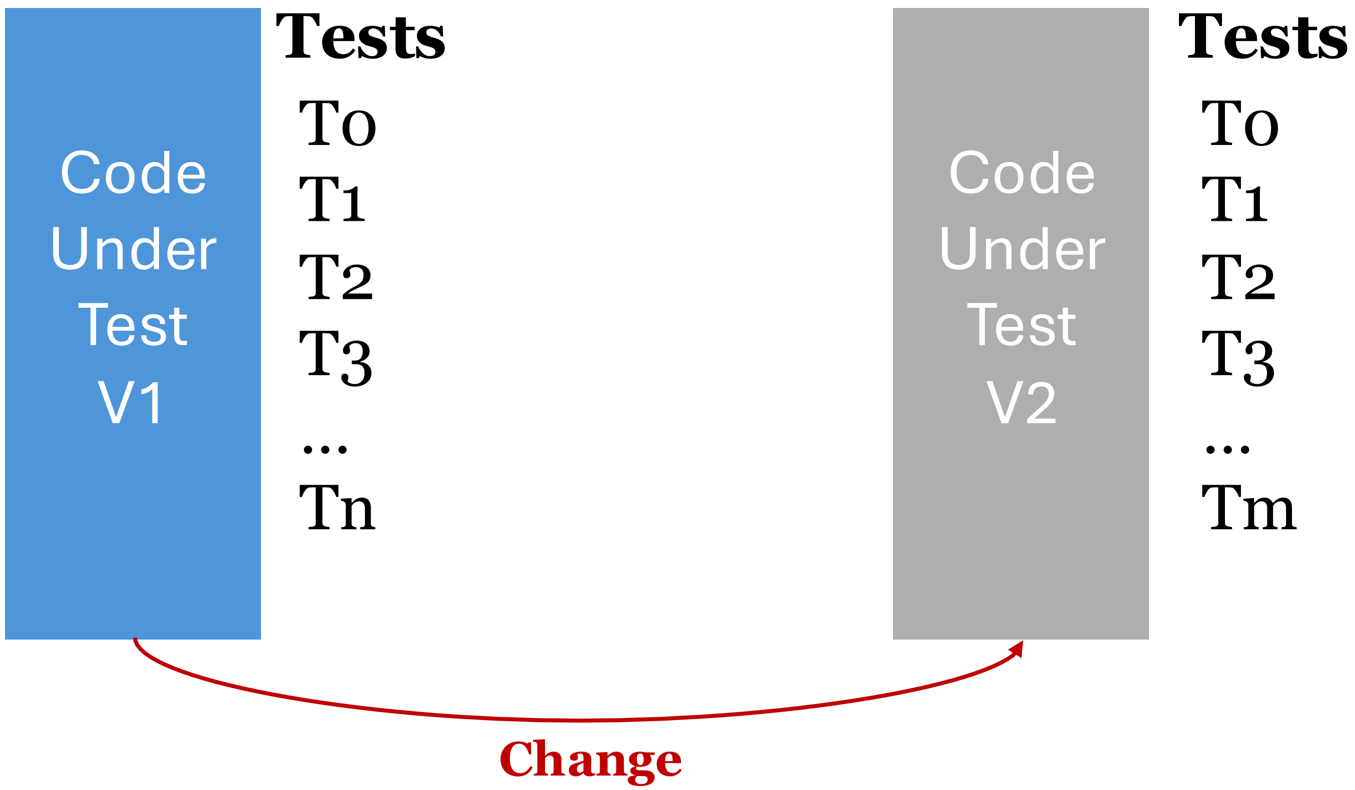
Regression testing for code changes

- Checking that code changes do not break working functionality
- Widely used in modern CI/CD environments



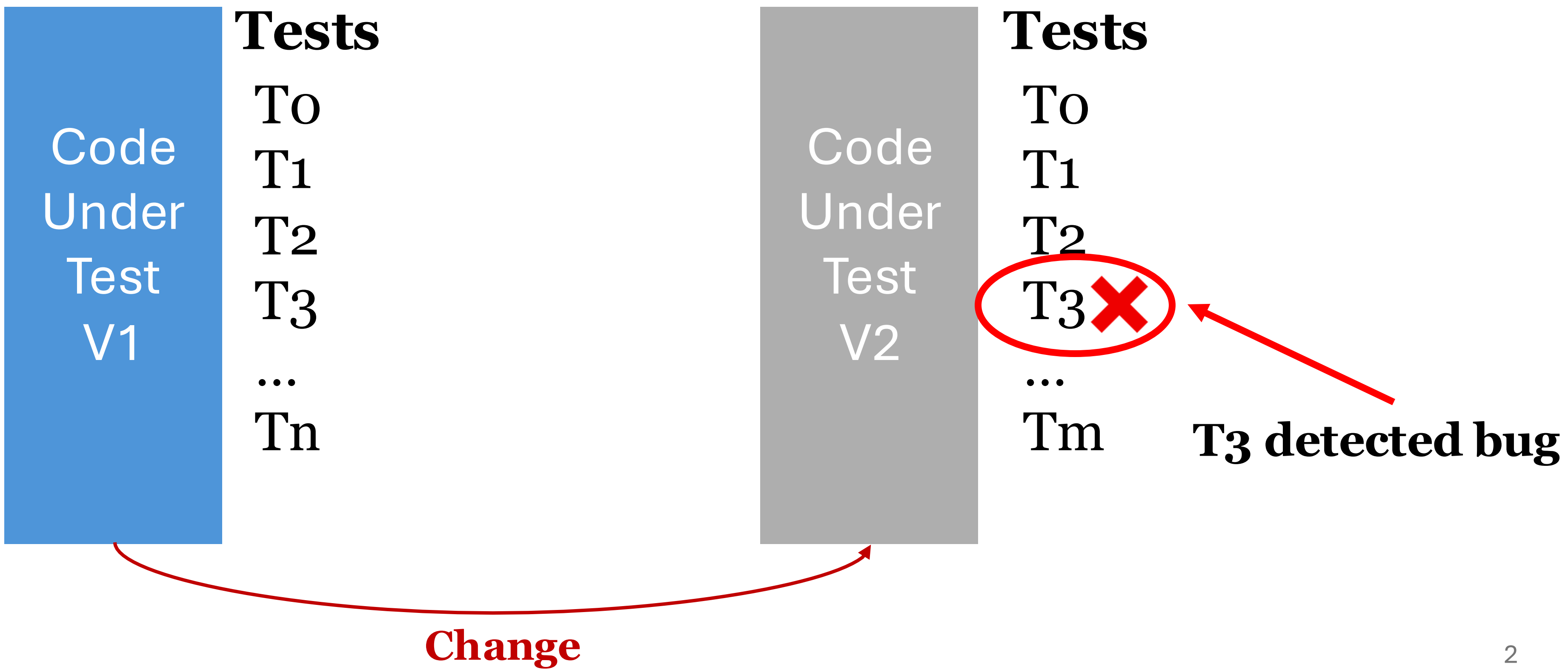
Regression testing for code changes

- Checking that code changes do not break working functionality
- Widely used in modern CI/CD environments



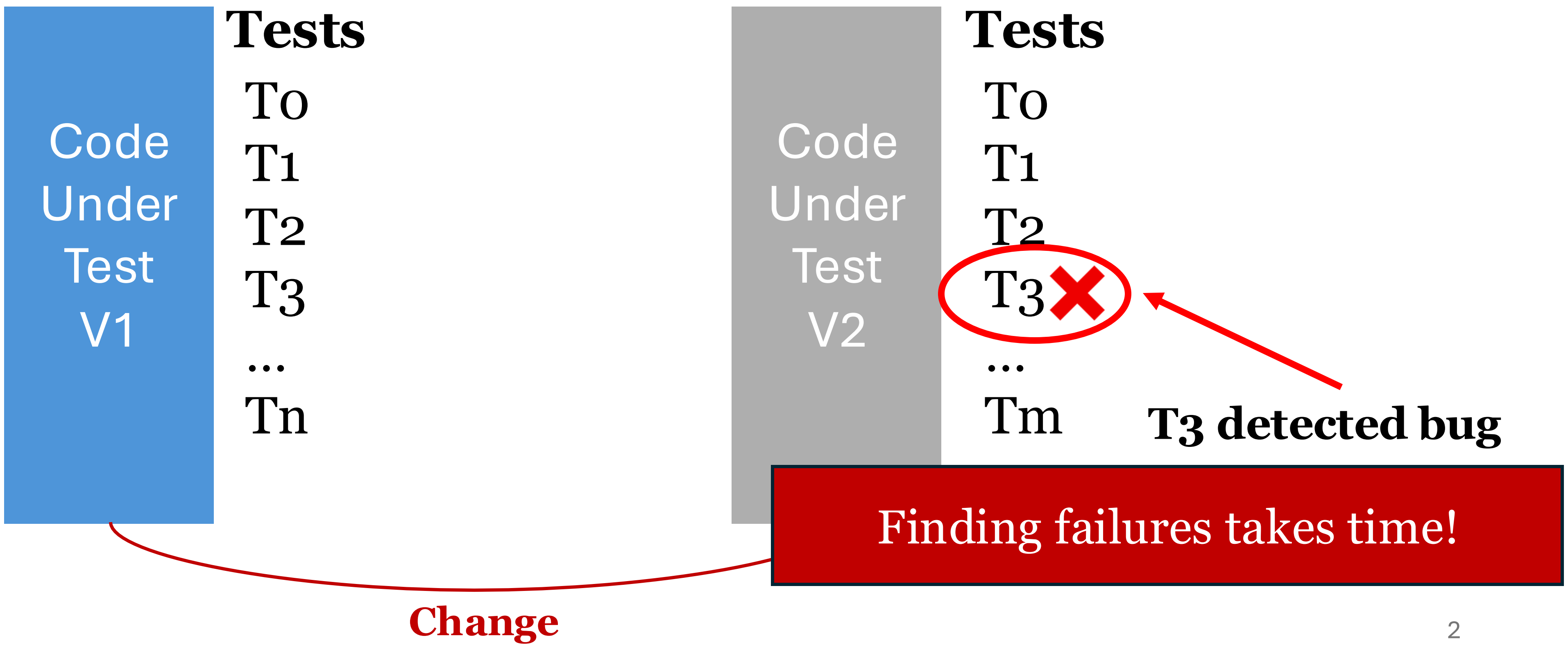
Regression testing for code changes

- Checking that code changes do not break working functionality
- Widely used in modern CI/CD environments



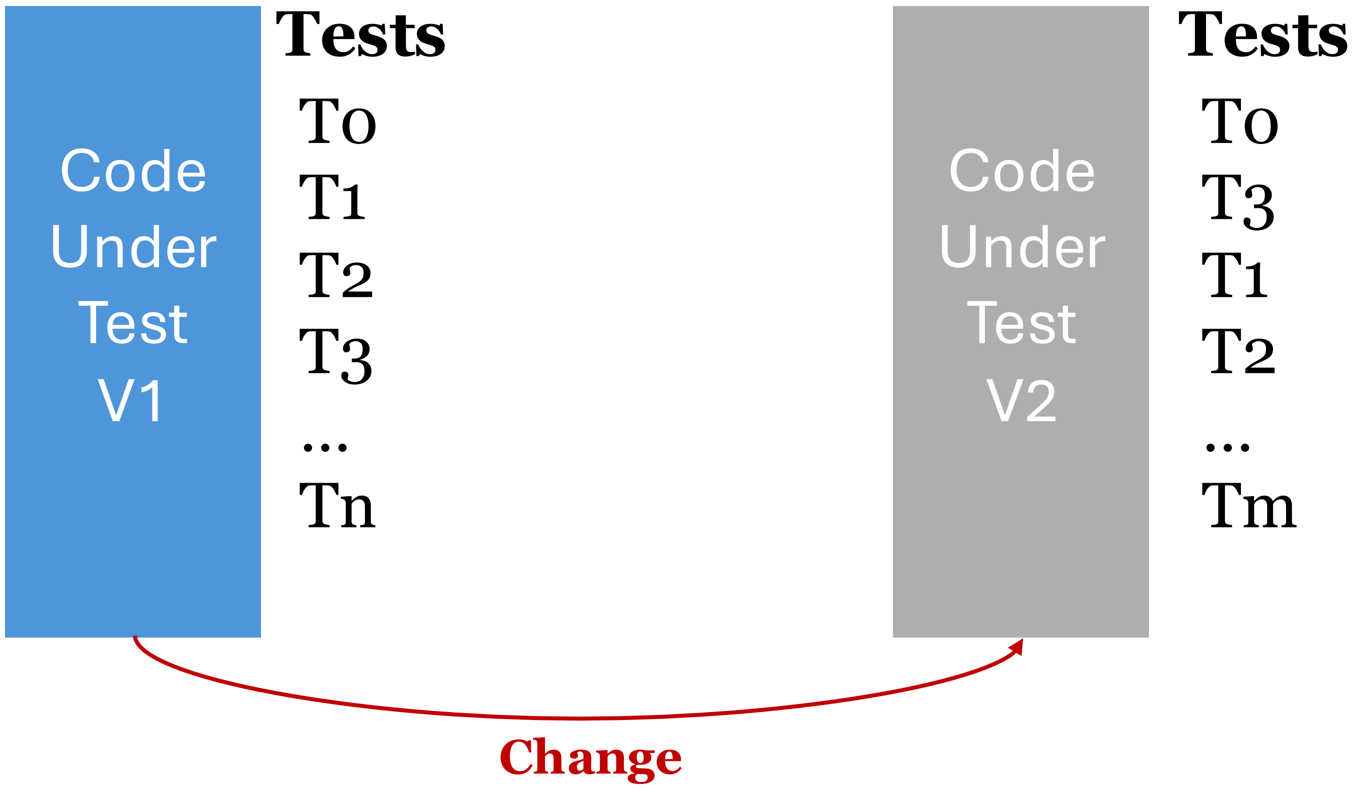
Regression testing for code changes

- Checking that code changes do not break working functionality
- Widely used in modern CI/CD environments



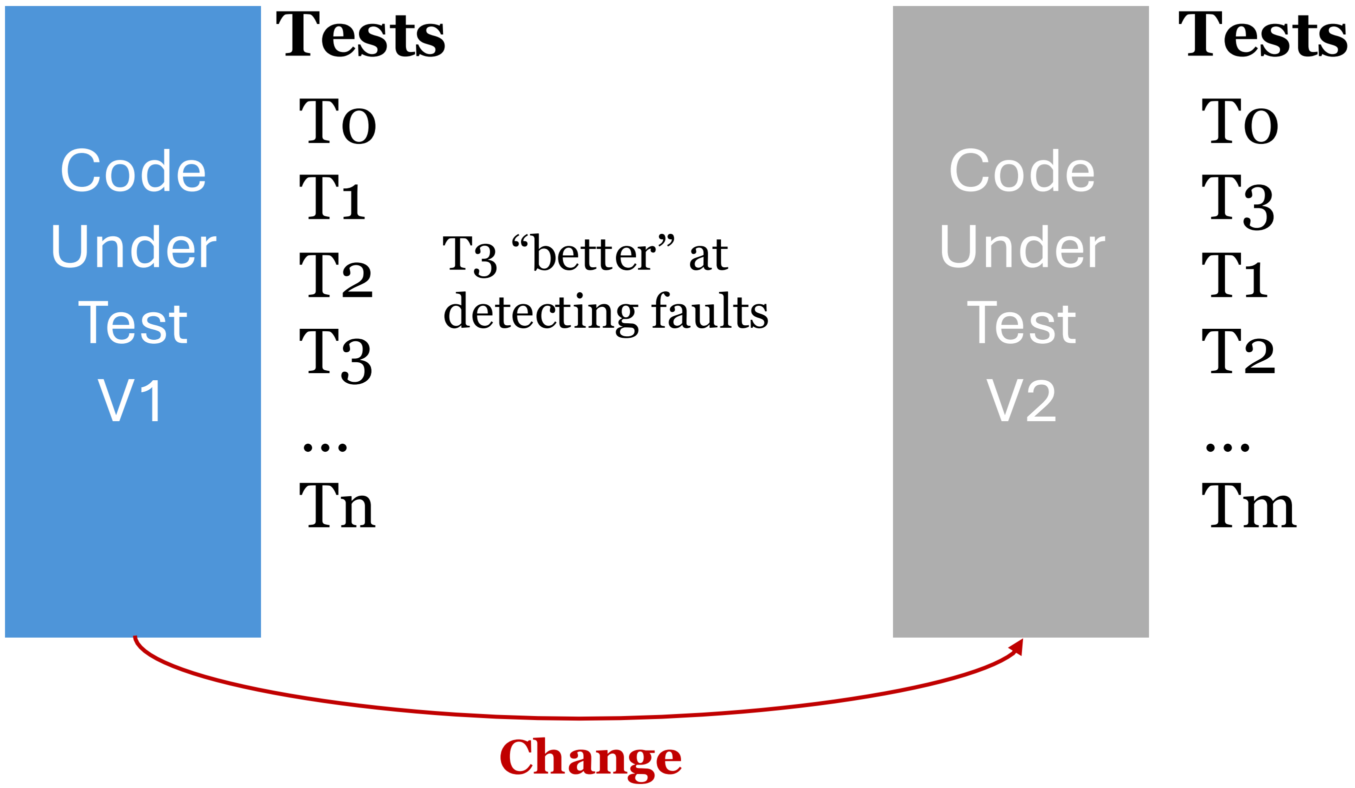
Test-Case Prioritization (TCP)

- Reorder tests to expose potential faults sooner



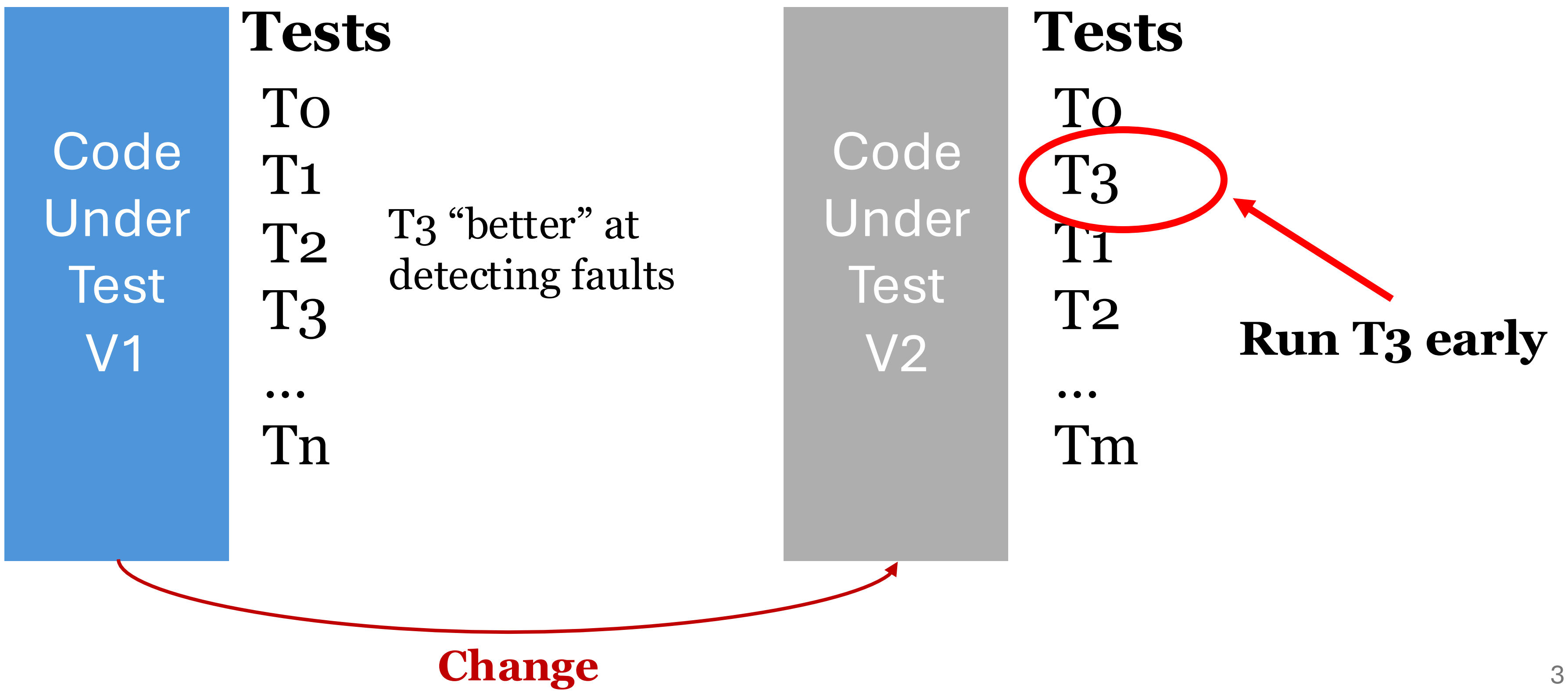
Test-Case Prioritization (TCP)

- Reorder tests to expose potential faults sooner



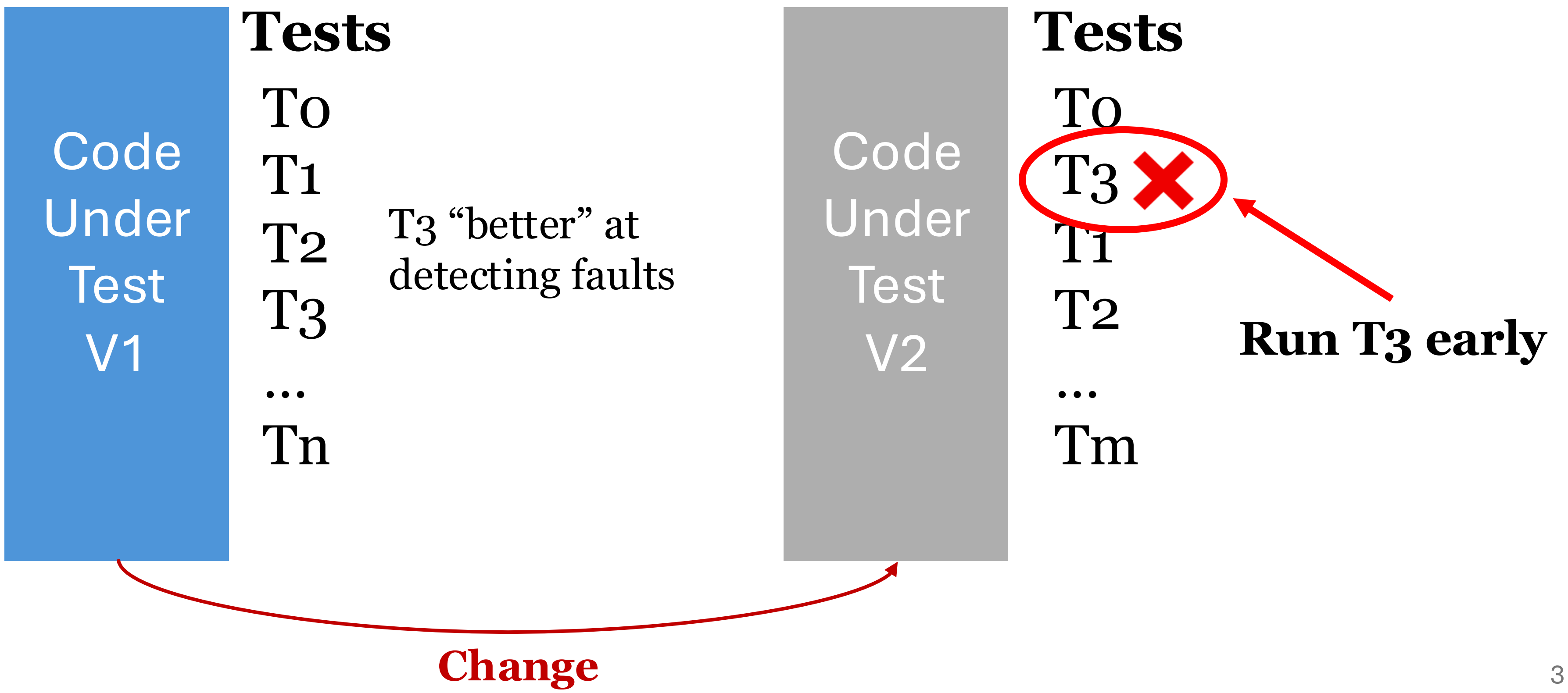
Test-Case Prioritization (TCP)

- Reorder tests to expose potential faults sooner



Test-Case Prioritization (TCP)

- Reorder tests to expose potential faults sooner



How to prioritize tests?

How to prioritize tests?

- Time-based TCP:
  - Prioritize tests that run **faster**

How to prioritize tests?

- Time-based TCP:
  - Prioritize tests that run **faster**
- History-based TCP:
  - Prioritize tests that have **historical data**, e.g., failed more frequently

How to prioritize tests?

- Time-based TCP:
  - Prioritize tests that run **faster**
- History-based TCP:
  - Prioritize tests that have **historical data**, e.g., failed more frequently
- IR-based TCP:
  - Prioritize tests that are **more textually similar** to the code changes
- Learning-based TCP:
  - Use **ML algorithms** to predict the ranking of tests

How to prioritize tests?

- Time-based TCP:
  - Prioritize tests that run **faster**
- History-based TCP:
  - Prioritize tests that have **historical data**, e.g., failed more frequently
- IR-based TCP:
  - Prioritize tests that are **more textually similar** to the code changes
- Learning-based TCP:
  - Use **ML algorithms** to predict the ranking of tests
- Hybrid TCP:
  - **Combine heuristics** from previous categories

Information Retrieval-based TCP

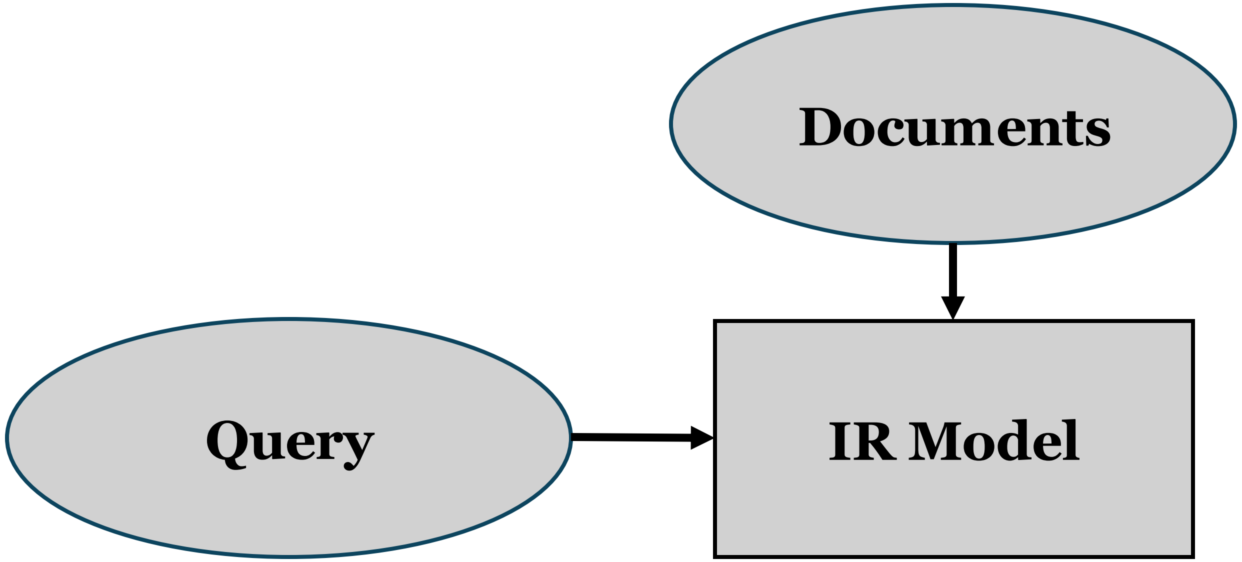
- Information Retrieval (IR)
  - Rank text documents based on the relevance to a query

Information Retrieval-based TCP

- Information Retrieval (IR)
  - Rank text documents based on the relevance to a query

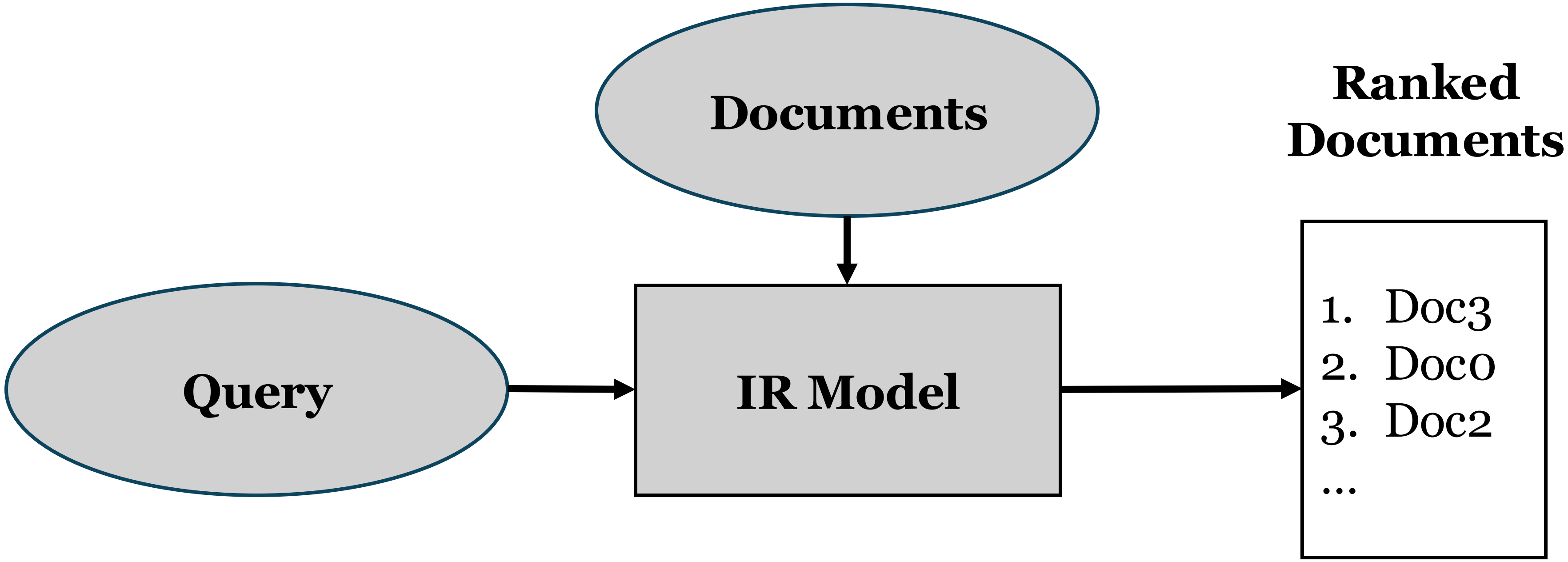
Information Retrieval-based TCP

- Information Retrieval (IR)
  - Rank text documents based on the relevance to a query



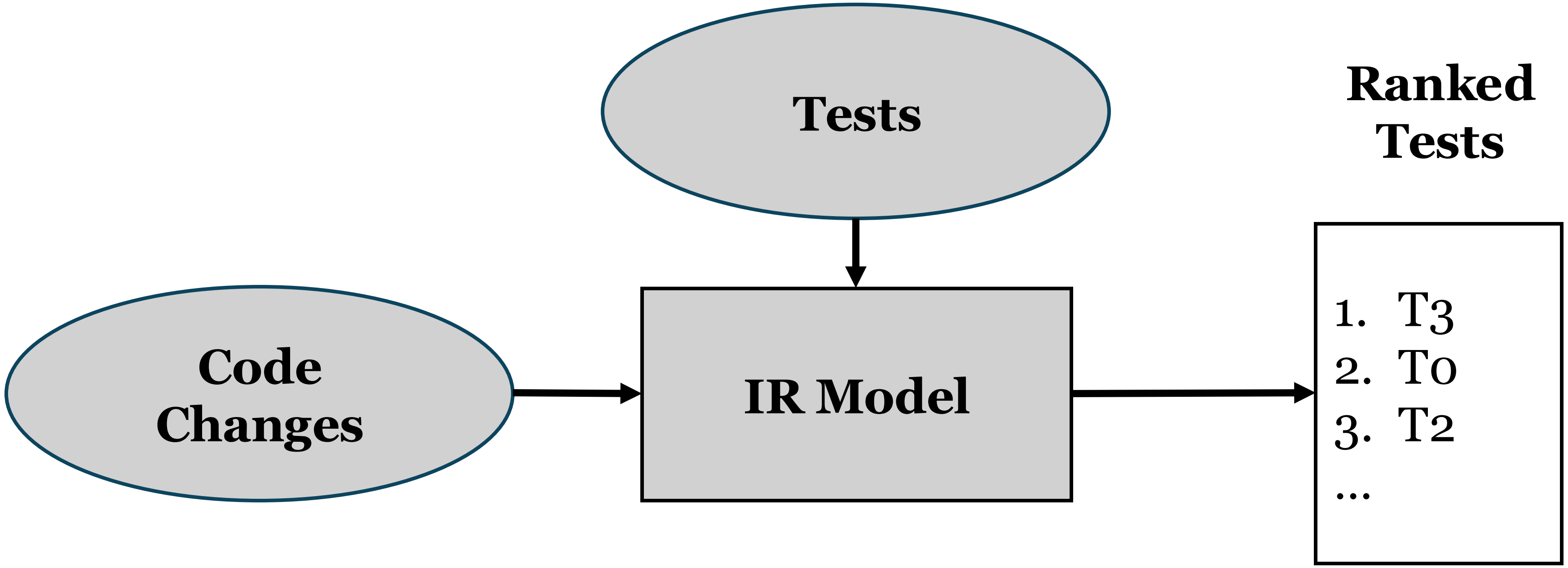
Information Retrieval-based TCP

- Information Retrieval (IR)
  - Rank text documents based on the relevance to a query



Information Retrieval-based TCP

- Information Retrieval (IR)
  - Rank tests based on the relevance to code changes



Learning-based TCP

Learning-based TCP

- Learning-to-Rank (LTR):

Learning-based TCP

- Learning-to-Rank (LTR):
  - Use **supervised learning** algorithms
  - Train on historical test runs to predict ranking of tests for future runs
- Ranking-to-Learn (RTL):

Learning-based TCP

- Learning-to-Rank (LTR):
  - Use **supervised learning** algorithms
  - Train on historical test runs to predict ranking of tests for future runs

- Ranking-to-Learn (RTL):
  - Use **reinforcement learning** algorithms
  - Continuously rank tests based on test states of the current run
  - Receive feedback from the ranking to improve its policy for next run

How to prioritize tests?

- Time-based TCP:
  - Prioritize tests that run **faster**

Which TCP should I use?

  - Prioritize tests that are **more relevant to changes** by textual similarity
- Learning-based TCP:
  - Use **ML algorithms** to predict the ranking of tests
- Hybrid TCP:
  - **Combine heuristics** from previous categories

Datasets are essential for TCP research

Datasets are essential for TCP research

- **Prior datasets for TCP are rather limited**
  - Consist of short-running test suites, e.g., runs for several minutes
  - Some from proprietary projects, e.g., test results of Google products
  - Outdated CI builds, e.g., >10-year-old builds in TravisTorrent [1]

Datasets are essential for TCP research

- **Prior datasets for TCP are rather limited**
  - Consist of short-running test suites, e.g., runs for several minutes
  - Some from proprietary projects, e.g., test results of Google products
  - Outdated CI builds, e.g., >10-year-old builds in TravisTorrent [1]

- **TCP is most useful on long-running test suites**
  - More tests, more complex tests
    → SAVE MORE TIME!
    → Harder to find and prioritize failing tests
  - On the contrary, TCP provides little value on short-running test suites

Datasets are essential for TCP research

- **Prior datasets for TCP are rather limited**
  - Consist of short-running test suites, e.g., runs for several minutes
  - Some from proprietary projects, e.g., test results of Google products
  - Outdated CI builds, e.g., >10-year-old builds in TravisTorrent [1]

- **TCP is most useful on long-running test suites**
  - More tests, more complex tests
    - → SAVE MORE TIME!
    - → Harder to find and prioritize failing tests
  - On the contrary, TCP provides little value on short-running test suites

- Will the same TCP techniques remain the most effective on long-running test suites?

Contributions

- **Dataset:** An extensive dataset focused on recent (2020-2023) **long-running test suites (LRTS)** that consists of 21K CI builds with 57K test-suite runs from 10 open-source projects
  - LRTS currently has 100K+ test-suite runs (an additional 43K+ test-suite runs since this ISSTA paper was accepted)
- **Extensive Study:** Evaluated 59 previously proposed TCP techniques on LRTS
- **Findings:** Revisited 11 key findings from recent TCP studies, confirming 9 and refuting 2 findings; presented 3 new findings
- **Data/code release:** https://github.com/lrtsuser/LRTS **QR code:**

LRTS Dataset

- Identify 10 Apache software projects that hosted long-running CI builds in public Jenkins CI servers

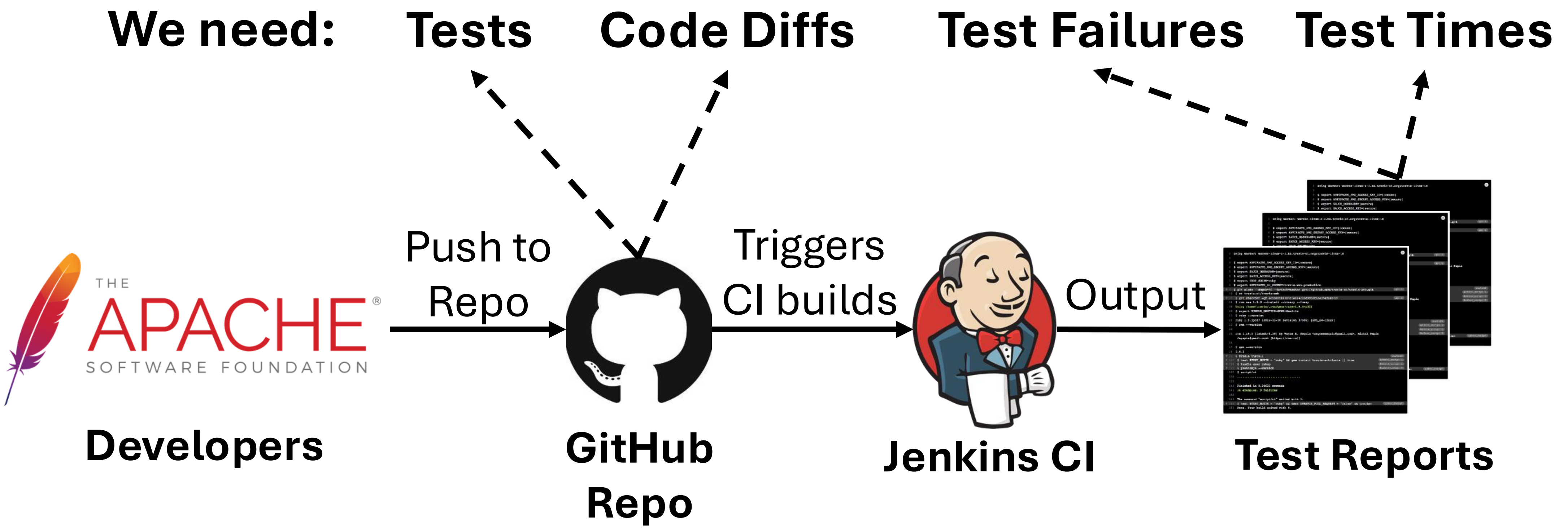
LRTS Dataset

- Identify 10 Apache software projects that hosted long-running CI builds in public Jenkins CI servers

We need: Tests Code Diffs Test Failures Test Times

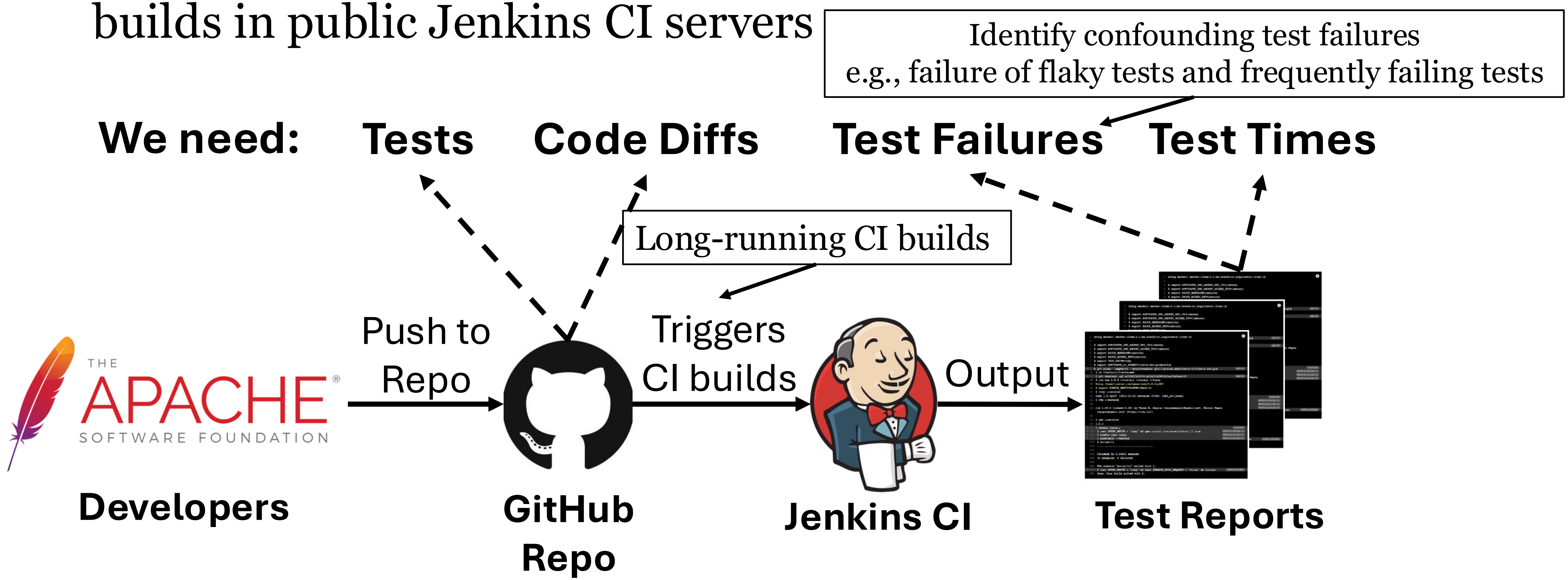
LRTS Dataset

- Identify 10 Apache software projects that hosted long-running CI builds in public Jenkins CI servers



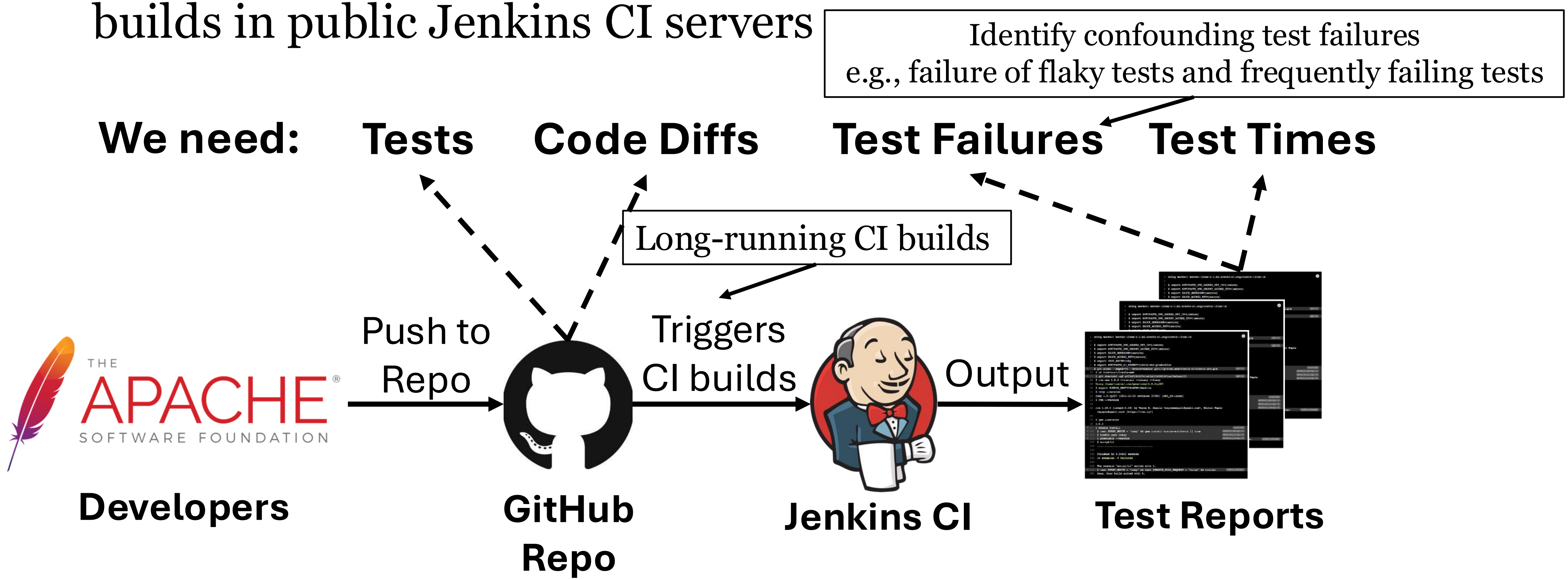
LRTS Dataset

- Identify 10 Apache software projects that hosted long-running CI builds in public Jenkins CI servers



LRTS Dataset

- Identify 10 Apache software projects that hosted long-running CI builds in public Jenkins CI servers



More details in paper

LRTS Dataset

LRTS Dataset

- **Recent:** LRTS spans from 2020 to 2023 from 10 projects

LRTS Dataset

- **Recent:** LRTS spans from 2020 to 2023 from 10 projects

LRTS Dataset

- **Recent:** LRTS spans from 2020 to 2023 from 10 projects
- **Long-running:** Average test-suite run duration of 6.5 hours, 18x longer than prior datasets except Chrome [7]

LRTS Dataset

- **Recent:** LRTS spans from 2020 to 2023 from 10 projects
- **Long-running:** Average test-suite run duration of 6.5 hours, 18x longer than prior datasets except Chrome [7]

TCP Dataset	#Project	#TSR	Test Suite Run Duration (hours)
RTPTorrent [2]	20	100K	0.17
Peng et al. [3]	123	3K	0.09
RT-CI [4]	6	3K	< 0.01
Pan and Pradel [5]	242	15K	0.35
TCP-CI [6]	25	21K	0.27
Chrome [7]	1	50K	7.96
LRTS (Ours)	**10**	**57K**	**6.50**

LRTS Dataset

- **Recent:** LRTS spans from 2020 to 2023 from 10 projects
- **Long-running:** Average test-suite run duration of 6.5 hours, 18x longer than prior datasets except Chrome [7]

APACHE ACTIVEMQ · hadoop · APACHE HBASE · HIVE · tvm · Apache Jackrabbit™ · James Enterprise Mail Server · Apache Kafka · Karaf · LOG4J

LRTS Dataset

- **Recent:** LRTS spans from 2020 to 2023 from 10 projects
- **Long-running:** Average test-suite run duration of 6.5 hours, 18x longer than prior datasets except Chrome [7]
- **Large-scale:** 21,255 unique CI builds from 10 projects, 57,437 test-suite runs and 30,118 (59%) had at least one failed test

LRTS Dataset

- **Recent:** LRTS spans from 2020 to 2023 from 10 projects
- **Long-running:** Average test-suite run duration of 6.5 hours, 18x longer than prior datasets except Chrome [7]
- **Large-scale:** 21,255 unique CI builds from 10 projects, 57,437 test-suite runs and 30,118 (59%) had at least one failed test
- **Diverse test failures:** 75% of the failed tests failed < 8 times

LRTS Dataset

- **Recent:** LRTS spans from 2020 to 2023 from 10 projects
- **Long-running:** Average test-suite run duration of 6.5 hours, 18x longer than prior datasets except Chrome [7]
- **Large-scale:** 21,255 unique CI builds from 10 projects, 57,437 test-suite runs and 30,118 (59%) had at least one failed test
- **Diverse test failures:** 75% of the failed tests failed < 8 times
- **Scripts:** We also released code to build and update LRTS

Evaluation setup

Evaluation setup

- Evaluation Metrics:
  - **A**verage **P**ercentage **F**aults **D**etected (APFD):
  - **A**verage **P**ercentage **F**aults **D**etected per **C**ost (APFDc)

Evaluation setup

- Evaluation Metrics:
  - **A**verage **P**ercentage **F**aults **D**etected (APFD):

$$1 - \frac{\sum_{i=1}^{m} TF_i}{n \times m} + \frac{1}{2n}$$

  - **A**verage **P**ercentage **F**aults **D**etected per **C**ost (APFDc)

$$1 - \frac{\sum_{i=1}^{m}(\sum_{j=TF_i}^{n} t_j - \frac{1}{2} t_{TF_i})}{\sum_{j=1}^{n} t_j \times m}$$

Evaluation setup

- Evaluation Metrics:
  - **A**verage **P**ercentage **F**aults **D**etected (APFD):

$$1 - \frac{\sum_{i=1}^{m} TF_i}{n \times m} + \frac{1}{2n}$$

  - **A**verage **P**ercentage **F**aults **D**etected per **C**ost (APFDc)

$$1 - \frac{\sum_{i=1}^{m}(\sum_{j=TF_i}^{n} t_j - \frac{1}{2} t_{TF_i})}{\sum_{j=1}^{n} t_j \times m}$$

- No direct mapping of test failures to faults:
  - $FFMap_S$: assumes all failures map to the same fault
  - $FFMap_U$: assumes each failure maps to a unique fault

Evaluation setup

Evaluation setup

- Study **26** basic TCP techniques:
  - 2 Time-based, 6 History-based, 6 IR-based, 5 LTR, 6 RTL
  - Random as baseline

Evaluation setup

- Study **26** basic TCP techniques:
  - 2 Time-based, 6 History-based, 6 IR-based, 5 LTR, 6 RTL
  - Random as baseline
- Apply 2 cost-cognizant hybrid TCP approaches to the basic techniques to construct **33** hybrid techniques

Evaluation setup

- Study **26** basic TCP techniques:
    - 2 Time-based, 6 History-based, 6 IR-based, 5 LTR, 6 RTL
    - Random as baseline
- Apply 2 cost-cognizant hybrid TCP approaches to the basic techniques to construct **33** hybrid techniques
    - Cost-cognizant (CC): prioritize tests with a short execution time

Evaluation setup

- Study **26** basic TCP techniques:
  - 2 Time-based, 6 History-based, 6 IR-based, 5 LTR, 6 RTL
  - Random as baseline
- Apply 2 cost-cognizant hybrid TCP approaches to the basic techniques to construct **33** hybrid techniques
  - Cost-cognizant (CC): prioritize tests with a short execution time
  - Cost-history-cognizant (CCH): prioritize tests that failed more often

Evaluation setup

More details about the studied TCP techniques in paper

- Study **26** basic TCP techniques:
  - 2 Time-based, 6 History-based, 6 IR-based, 5 LTR, 6 RTL
  - Random as baseline
- Apply 2 cost-cognizant hybrid TCP approaches to the basic techniques to construct **33** hybrid techniques
  - Cost-cognizant (CC): prioritize tests with a short execution time
  - Cost-history-cognizant (CCH): prioritize tests that failed more often

Evaluation setup

More details about the studied TCP techniques in paper

- Study **26** basic TCP techniques:
  - 2 Time-based, 6 History-based, 6 IR-based, 5 LTR, 6 RTL
  - Random as baseline
- Apply 2 cost-cognizant hybrid TCP approaches to the basic techniques to construct **33** hybrid techniques
  - Cost-cognizant (CC): prioritize tests with a short execution time
  - Cost-history-cognizant (CCH): prioritize tests that failed more often
- 3 LRTS versions:
  - LRTS-All: Keeps all test failures
  - LRTS-DeConf: Omits identified confounding test failures
  - LRTS-FirstFail: Only keeps the first failure of each non-flaky test

Evaluating TCP techniques on LRTS

Evaluating TCP techniques on LRTS

- RQ1: How do TCP techniques perform on long-running test suites from recent builds? (8 confirmations and 2 refutations)

Evaluating TCP techniques on LRTS

- RQ1: How do TCP techniques perform on long-running test suites from recent builds? (8 confirmations and 2 refutations)
- RQ2: How do confounding test failures impact TCP effectiveness? (1 confirmation and 2 new findings)

Evaluating TCP techniques on LRTS

- RQ1: How do TCP techniques perform on long-running test suites from recent builds? (8 confirmations and 2 refutations)

- RQ2: How do confounding test failures impact TCP effectiveness? (1 confirmation and 2 new findings)

- RQ3: How do TCP techniques perform in detecting the first failure throughout CI history for each failed test? (1 new finding)

Evaluating TCP techniques on LRTS

- RQ1: How do TCP techniques perform on long-running test suites from recent builds? (8 confirmations and 2 refutations)

- RQ2: How do confounding test failures impact TCP effectiveness? (1 confirmation and 2 new findings)

- RQ3: How do TCP techniques perform in detecting the first failure throughout CI history for each failed test? (1 new finding)

- RQ1 - LRTS-DeConf; RQ2 - LRTS-All; RQ3 - LRTS-FirstFail

RQ1: Effectiveness of TCP techniques

- **Confirmed 8 and refuted 2 prior findings, e.g.,**

*Confirmations are colored **green**; refutations **red**; new findings **purple***

RQ1: Effectiveness of TCP techniques

- **Confirmed 8 and refuted 2 prior findings, e.g.,**
  - Simpler TCPs (time-based, history-based) outperform sophisticated ones (IR-based, ML/RL-based) [3, 9]

*Confirmations are colored **green**; refutations **red**; new findings **purple***

RQ1: Effectiveness of TCP techniques

- **Confirmed 8 and refuted 2 prior findings, e.g.,**
  - Simpler TCPs (time-based, history-based) outperform sophisticated ones (IR-based, ML/RL-based) [3, 9]
  - Cost-cognizant hybrids substantially improve basic TCPs [3]
    - Prioritizing faster tests that failed recently is the best [3]

*Confirmations are colored **green**; refutations **red**; new findings **purple***

RQ1: Effectiveness of TCP techniques

- **Confirmed 8 and refuted 2 prior findings, e.g.,**
  - Simpler TCPs (time-based, history-based) outperform sophisticated ones (IR-based, ML/RL-based) [3, 9]
  - Cost-cognizant hybrids substantially improve basic TCPs [3]
    - Prioritizing faster tests that failed recently is the best [3]
  - IR-based TCP performed worse when test suites have more failures or longer-running tests [3]

*Confirmations are colored **green**; refutations **red**; new findings **purple***

RQ1: Effectiveness of TCP techniques

- **Confirmed 8 and refuted 2 prior findings, e.g.,**
  - Simpler TCPs (time-based, history-based) outperform sophisticated ones (IR-based, ML/RL-based) [3, 9]
  - Cost-cognizant hybrids substantially improve basic TCPs [3]
    - Prioritizing faster tests that failed recently is the best [3]
  - IR-based TCP performed worse when test suites have more failures or longer-running tests [3]
  - Different configurations have little impact on the effectiveness of IR-based techniques [3, 8]

Confirmations are colored ***green****; refutations* ***red****; new findings* ***purple***

RQ2: Impact* of confounding test failures (CTF)

- CTF: failures of flaky tests and frequently failing tests
- CTF should NOT be prioritized over other failures

RQ2: Impact* of confounding test failures (CTF)

- CTF: failures of flaky tests and frequently failing tests
- CTF should NOT be prioritized over other failures

**A TCP is negatively impacted by CTFs if APFD(c) decreases because it prioritizes CTFs over non-CTFs that are true failures.*

RQ2: Impact* of confounding test failures (CTF)

- CTF: failures of flaky tests and frequently failing tests
- CTF should NOT be prioritized over other failures

- **Confirmed 1 prior finding and present 2 new findings,**

**A TCP is negatively impacted by CTFs if APFD(c) decreases because it prioritizes CTFs over non-CTFs that are true failures.*

RQ2: Impact* of confounding test failures (CTF)

- CTF: failures of flaky tests and frequently failing tests
- CTF should NOT be prioritized over other failures

- **Confirmed 1 prior finding and present 2 new findings,**
  - History-based TCP are the most negatively impacted* by CTF [7], but those favoring recent history are resilient to CTF

**A TCP is negatively impacted by CTFs if APFD(c) decreases because it prioritizes CTFs over non-CTFs that are true failures.*

RQ2: Impact* of confounding test failures (CTF)

- CTF: failures of flaky tests and frequently failing tests
- CTF should NOT be prioritized over other failures

- **Confirmed 1 prior finding and present 2 new findings,**
  - History-based TCP are the most negatively impacted* by CTF [7], but those favoring recent history are resilient to CTF
  - Time-based and change-aware (IR-based) TCP are the least impacted by CTF

**A TCP is negatively impacted by CTFs if APFD(c) decreases because it prioritizes CTFs over non-CTFs that are true failures.*

RQ3: TCP Effectiveness on finding first failures

- First failure of a test: the first time a test failed in CI
- Goal: understand how TCP perform when most tests do not fail

RQ3: TCP Effectiveness on finding first failures

- First failure of a test: the first time a test failed in CI
- Goal: understand how TCP perform when most tests do not fail

- **Presented 1 new finding, i.e.,**
  - Time-based and change-aware TCP are more effective in finding first failures, then Random, then history-based TCP

Conclusions

- **Dataset:** An extensive dataset focused on recent (2020-2023) **long-running test suites (LRTS)** that consists of 21K CI builds with 57K test-suite runs from 10 open-source projects
  - LRTS currently has 100K+ test-suite runs (an additional 43K+ test-suite runs since this ISSTA paper was accepted)
- **Extensive Study:** Evaluated 59 previously proposed TCP techniques on LRTS
- **Threat:** Due to high cost, didn't run generated test orders [ISSRE'24]
- **Findings:** Revisited 11 key findings from recent TCP studies, confirming 9 and refuting 2 findings; presented 3 new findings
- **Data/code release:** https://github.com/lrtsuser/LRTS

QR code:

References

[1] Moritz Beller, Georgios Gousios, and Andy Zaidman. TravisTorrent: Synthesizing Travis CI and GitHub for Full-Stack Research on Continuous Integration. In MSR (2017).

[2] Toni Mattis, Patrick Rein, Falco Dursch, and Robert Hirschfeld. RTPTorrent: An Open-source Dataset for Evaluating Regression Test Prioritization. In MSR (2020).

[3] Qianyang Peng, August Shi, and Lingming Zhang. Empirically Revisiting and Enhancing IR-Based Test-Case Prioritization. In ISSTA (2020).

[4] Antonia Bertolino, Antonio Guerriero, Breno Miranda, Roberto Pietrantuono, and Stefano Russo. Learning-to-Rank vs Ranking-to-Learn: Strategies for Regression Testing in Continuous Integration. In ICSE (2020).

[5] Cong Pan and Michael Pradel. Continuous Test Suite Failure Prediction. In ISSTA (2021).

[6] Ahmadreza Saboor Yaraghi, Mojtaba Bagherzadeh, Nafiseh Kahani, and Lionel C Briand. Scalable and Accurate Test Case Prioritization in Continuous Integration Contexts. In TSE (2022).

[7] Emad Fallahzadeh and Peter C Rigby. The Impact of Flaky Tests on Historical Test Prioritization on Chrome. In ICSE-SEIP (2022).

[8] Ripon K Saha, Lingming Zhang, Sarfraz Khurshid, and Dewayne E Perry. An Information Retrieval Approach for Regression Test Prioritization Based on Program Changes. In ICSE (2021).

[9] Daniel Elsner, Florian Hauer, Alexander Pretschner, and Silke Reimer. Empirically Evaluating Readily Available Information for Regression Test Optimization in Continuous Integration. In ISSTA (2021).

Backup

CI Build CDFs

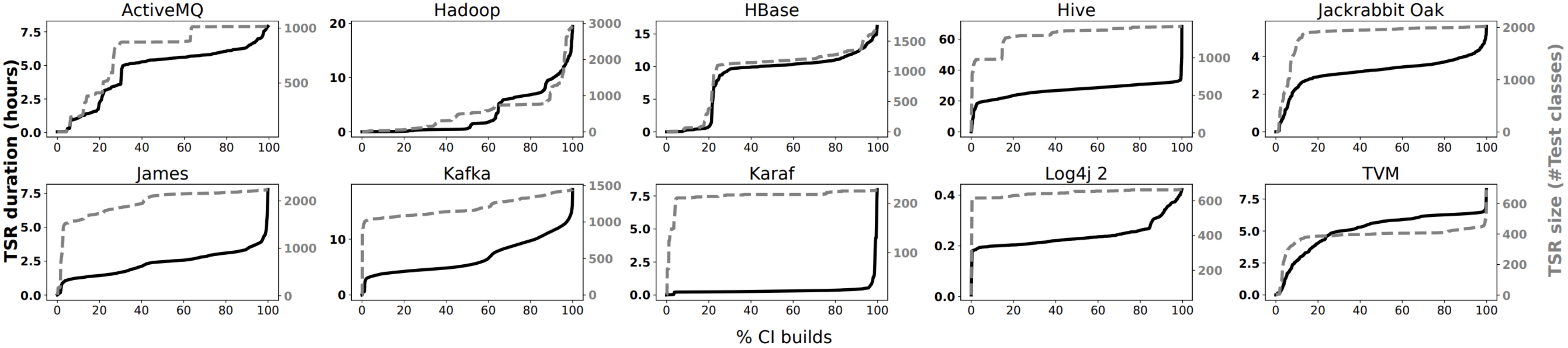


Figure 1: Distribution of CI builds by the duration (hours) and size (number of test classes) of all (not only failed) TSRs. The solid dark lines and left y-axes show CDFs by TSR duration. The dashed lighter lines and right y-axes show CDFs by TSR size.

Evaluation Setup

Table 2: *LRTS* dataset summary. TSR denotes test-suite run, TC denotes test class, and TM denotes test method.

Project	Main PLs	SLOC	Period (days)	#CI build	#TSR	#Failed TSR	Statistics (Averages) on failed TSRs				
							#TC	#Failed TC	#TM	#Failed TM	Duration (hours)
ActiveMQ	Java	669K	827	207	207	109	676	3	6,081	34	4.36
Hadoop	Java	4M	1,094	1,299	1,299	543	829	6	7,289	24	5.57
HBase	Java	1M	504	278	553	215	1,061	2	6,369	3	9.28
Hive	Java, HiveQL	2M	618	2,056	2,056	1,419	1,273	9	40,921	83	26.12
Jackrabbit Oak	Java	694K	745	860	860	639	1,897	12	19,699	107	3.27
James	Java, Scala	793K	786	2,404	3,147	1,399	1,864	6	34,718	37	2.15
Kafka	Java, Scala	905K	984	11,843	39,006	24,047	1,232	4	19,399	12	7.59
Karaf	Java, Scala	186K	959	620	620	174	205	2	841	2	0.58
Log4j 2	Java	277K	436	270	528	162	641	3	3,918	4	0.25
TVM	Python, C++	818K	631	1,418	9,161	1,411	526	3	8,564	37	4.83
Total				**21,255**	**57,437**	**30,118**					

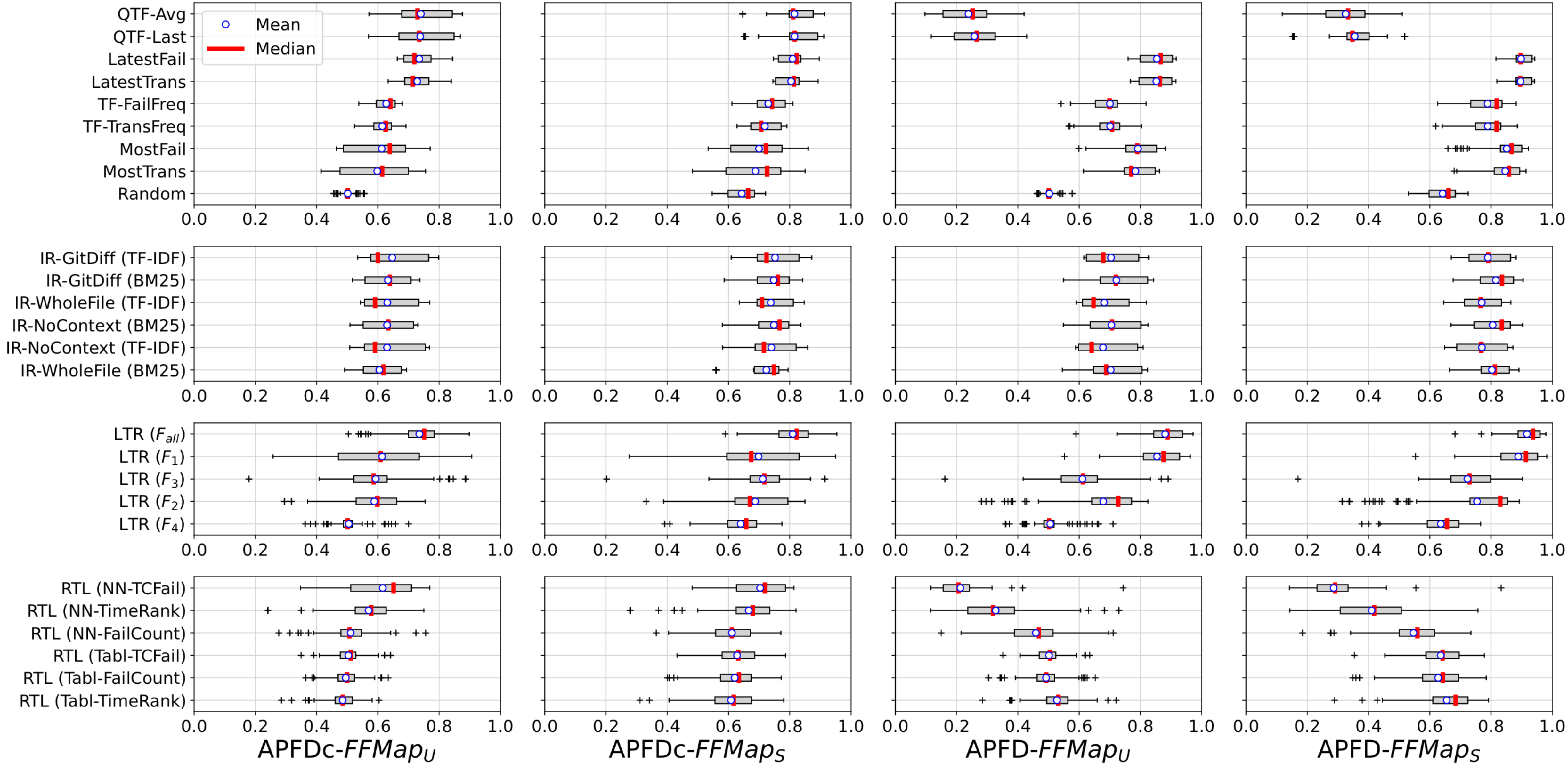
Table 6: Dataset versions.

Version	#Failed TSR
LRTS-All	30,118
LRTS-DeConf	9,683
LRTS-FirstFail	2,076

All Findings

F1 Different failure-to-fault mappings lead to similar ranking of TCP techniques [83].	✓
F2 APFD can be misleading and give different ranking of TCP techniques than APFDc [13, 64].	✓
F3 Basic time-based and history-based techniques can rival or outperform sophisticated IR-based and learning-based techniques [19, 83].	✓
F4 All IR-based techniques perform worse than time-based and history-based techniques [83].	✗
F5 Different configurations have little impact on the effectiveness of IR-based techniques [83, 92].	✗
F6 LTR TCP is among the most effective TCP techniques when training with all available features [19].	✓
F7 In LTR TCP, training with all features (F_{all}) outperforms every individual feature set; execution time and outcome features (F_1) outperform associated history and similarity features (F_2 and F_3) which outperform change features (F_4) [19, 110].	✓
F8 RTL techniques generally perform better than random [98] but worse than LTR techniques [9].	✓
F9 Cost-cognizant hybrid TCP approaches can substantially improve the effectiveness of basic TCP techniques [83].	✓
F10 Among all techniques, hybrid perform the best [83], specifically techniques that combine time-based and history-based heuristics.	✓
F11 Techniques that rely on test outcome frequency, e.g., *MostFail* and *LTR (F_1)*, are heavily impaired by confounding test failures [21].	✓
F12 Techniques that favor more recent test history, e.g., *LatestFail* and *RTL (NN-TestFail)*, are resilient to confounding test failures.	💡
F13 Time-based and change-aware techniques, e.g., IR-based, are the least affected by the presence of confounding test failures.	💡
F14 Time-based and change-aware techniques are effective in finding the first failures of tests, followed by *Random*, then history-based.	💡

Results (RQ1)


Mean
Median
QTF-Avg
QTF-Last
LatestFail
LatestTrans
TF-FailFreq
TF-TransFreq
MostFail
MostTrans
Random
IR-GitDiff (TF-IDF)
IR-GitDiff (BM25)
IR-WholeFile (TF-IDF)
IR-NoContext (BM25)
IR-NoContext (TF-IDF)
IR-WholeFile (BM25)
LTR (F_{all})
LTR (F_1)
LTR (F_3)
LTR (F_2)
LTR (F_4)
RTL (NN-TCFail)
RTL (NN-TimeRank)
RTL (NN-FailCount)
RTL (Tabl-TCFail)
RTL (Tabl-FailCount)
RTL (Tabl-TimeRank)
0.0 0.2 0.4 0.6 0.8 1.0
APFDc-$FFMap_U$
APFDc-$FFMap_S$
APFD-$FFMap_U$
APFD-$FFMap_S$

Results (RQ1)

Table 8: APFDc-$FFMap_U$ results on *LRTS-DeConf*. Horizontal lines separate TCP technique categories.

TCP Technique	Basic			*CC*		*CCH*	
	Avg	G.Cat	G.All	Avg	Imp	Avg	Imp
QTF-Avg	.740	A	A	-	-	-	-
QTF-Last	.739	A	A	-	-	-	-
LatestFail	.735	A	A	.835	13%	.797	8%
LatestTrans	.728	A	A	.830	13%	.795	9%
TF-FailFreq	.627	B	BCD	.788	25%	.773	23%
TF-TransFreq	.614	B	BCD	.777	26%	.764	24%
MostFail	.613	B	BCDE	.773	26%	-	-
MostTrans	.598	B	CDE	.765	27%	.743	24%
Random	.502	C	F	-	-	-	-
IR-GitDiff (TF-IDF)	.647	A	B	.767	18%	.789	21%
IR-GitDiff (BM25)	.633	AB	BC	.743	17%	.771	21%
IR-WholeFile (TF-IDF)	.631	AB	BCD	.761	20%	.785	24%
IR-NoContext (BM25)	.630	AB	BCD	.741	17%	.770	22%
IR-NoContext (TF-IDF)	.630	AB	BCD	.758	20%	.784	24%
IR-WholeFile (BM25)	.605	B	BCDE	.739	22%	.767	26%
LTR (F_{all})	.736	A	A	.809	9%	.781	6%
LTR (F_1)	.614	B	BCD	.767	24%	.739	20%
LTR (F_3)	.593	B	CDE	.706	19%	.741	24%
LTR (F_2)	.588	B	DE	.727	23%	.735	24%
LTR (F_4)	.505	C	F	.717	41%	.747	47%
RTL (NN-TCFail)	.616	A	BCD	-	-	-	-
RTL (NN-TimeRank)	.570	B	E	-	-	-	-
RTL (NN-FailCount)	.511	C	F	-	-	-	-
RTL (Tabl-TCFail)	.504	C	F	-	-	-	-
RTL (Tabl-FailCount)	.495	C	F	-	-	-	-
RTL (Tabl-TimeRank)	.485	C	F	-	-	-	-

Table 9: IR experiment.

Variable	Variable Value Range			
	<Q1	Q1-2	Q2-3	>Q3
Duration	.644	.642	.628	.605
#Failure	.679	.672	.640	.569
Fail ratio	.693	.686	.607	.577
Chg size	.617	.612	.632	.648

Table 1: LTR TCP feature sets.

F_1: test history features	**F_2: (Test,File)-history features**
Failure count	Max (test,file)-failure freq
Last failure	Max (test,file)-transition freq
Transition count	Max (test,file)-failure freq (relative)
Last transition	Max (test,file)-transition freq (relative)
Average duration	
F_3: (Test,File)-similarity features	**F_4: change features**
Min file path distance	Distinct authors
Max file path token similarity	Changeset cardinality
Min file name distance	Amount of commits

Results (RQ2 & RQ3)

Table 10: Mean APFDc-$FFMap_U$ and effectiveness group of TCP techniques on all three versions of *LRTS*.

TCP Technique	*LRTS-DeConf*		*LRTS-All*		*LRTS-FirstFail*	
QTF-Avg	**.740**	**A**	.671	CD	**.796**	**A**
QTF-Last	**.739**	**A**	.677	CD	**.798**	**A**
LatestFail	**.735**	**A**	**.795**	**A**	.467	DE
LatestTrans	**.728**	**A**	**.788**	**A**	.464	DEF
TF-FailFreq	.627	BCD	.666	CD	.440	EF
TF-TransFreq	.614	BCD	.656	D	.422	F
MostFail	.613	BCDE	**.720**	**B**	.312	G
MostTrans	.598	CDE	.701	BC	.313	G
Random	.502	F	.502	I	.504	D
IR-GitDiff (TF-IDF)	.647	B	.589	EF	**.691**	**B**
IR-GitDiff (BM25)	.633	BC	.576	FG	.667	BC
IR-WholeFile (TF-IDF)	.631	BCD	.576	FG	**.679**	**B**
IR-NoContext (BM25)	.630	BCD	.579	FG	.666	BC
IR-NoContext (TF-IDF)	.630	BCD	.583	EFG	**.680**	**B**
IR-WholeFile (BM25)	.605	BCDE	.557	FG	.632	C
LTR (F_{all})	**.736**	**A**	**.764**	**A**	-	-
LTR (F_1)	.614	BCD	**.724**	**B**	-	-
LTR (F_3)	.593	CDE	.548	GH	-	-
LTR (F_2)	.588	DE	.618	E	-	-
LTR (F_4)	.505	F	.505	I	-	-
RTL (NN-TCFail)	.616	BCD	.549	GH	-	-
RTL (NN-TimeRank)	.570	E	.516	HI	-	-
RTL (NN-FailCount)	.511	F	.481	I	-	-
RTL (Tabl-TCFail)	.504	F	.508	I	-	-
RTL (Tabl-FailCount)	.495	F	.501	I	-	-
RTL (Tabl-TimeRank)	.485	F	.517	HI	-	-