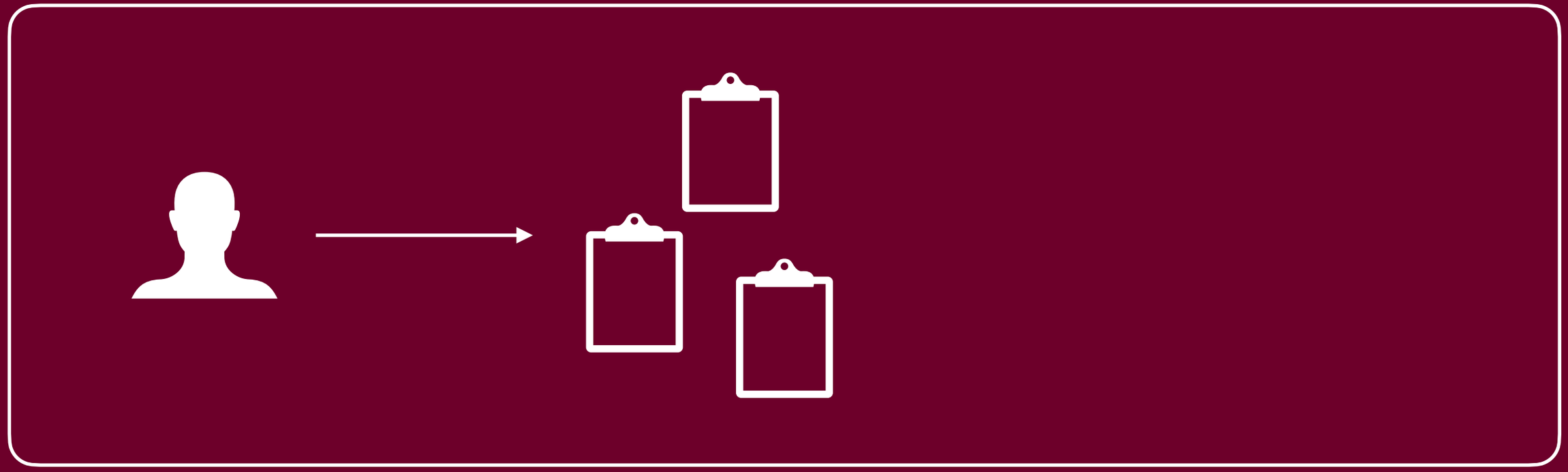
Understanding Flaky Tests: The Developer's Perspective

Moritz Eck,* **Fabio Palomba**,* Marco Castelluccio,¶ Alberto Bacchelli*

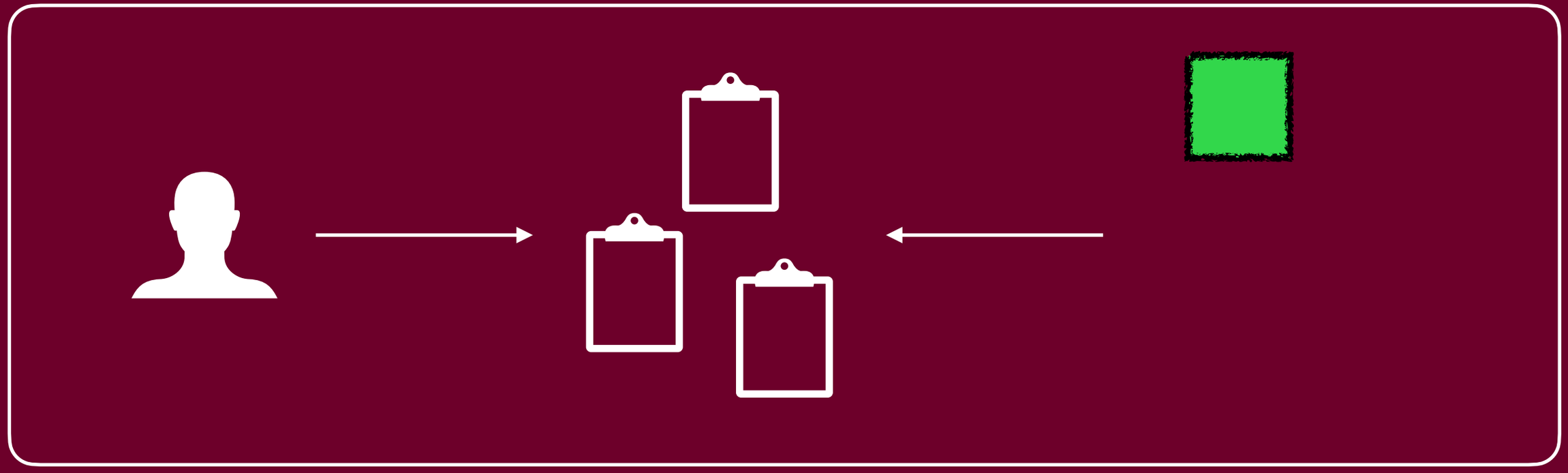
*University of Zurich, Switzerland — ¶Mozilla Software Foundation, UK

Test cases form the first line of defence against the introduction of software faults, especially when testing for regressions

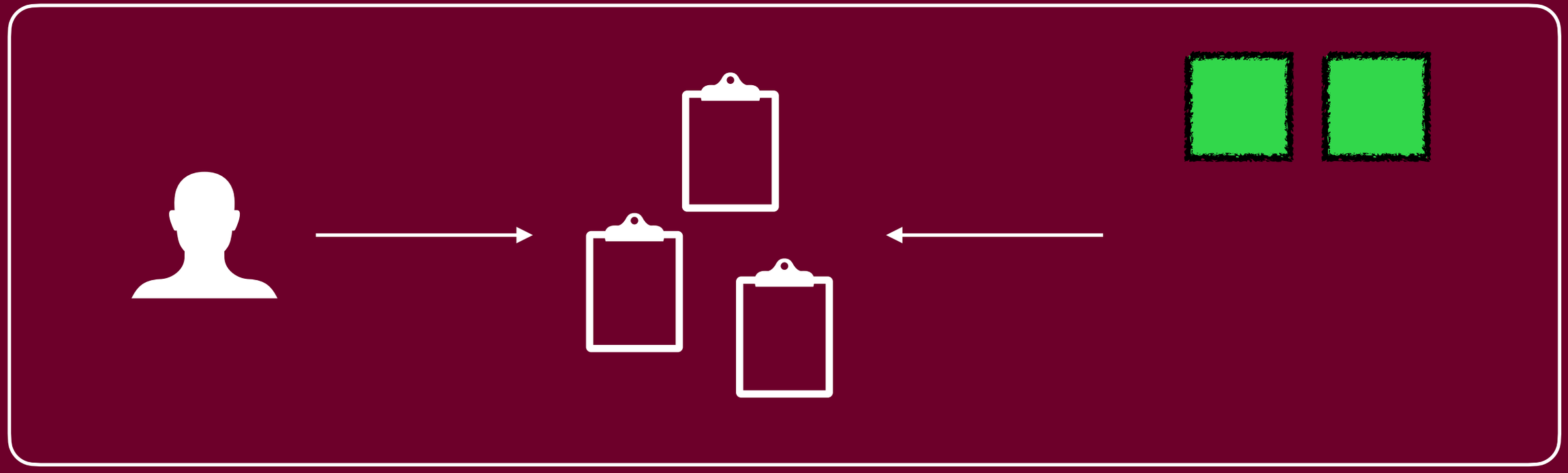
Test cases form the first line of defence against the introduction of software faults, especially when testing for regressions



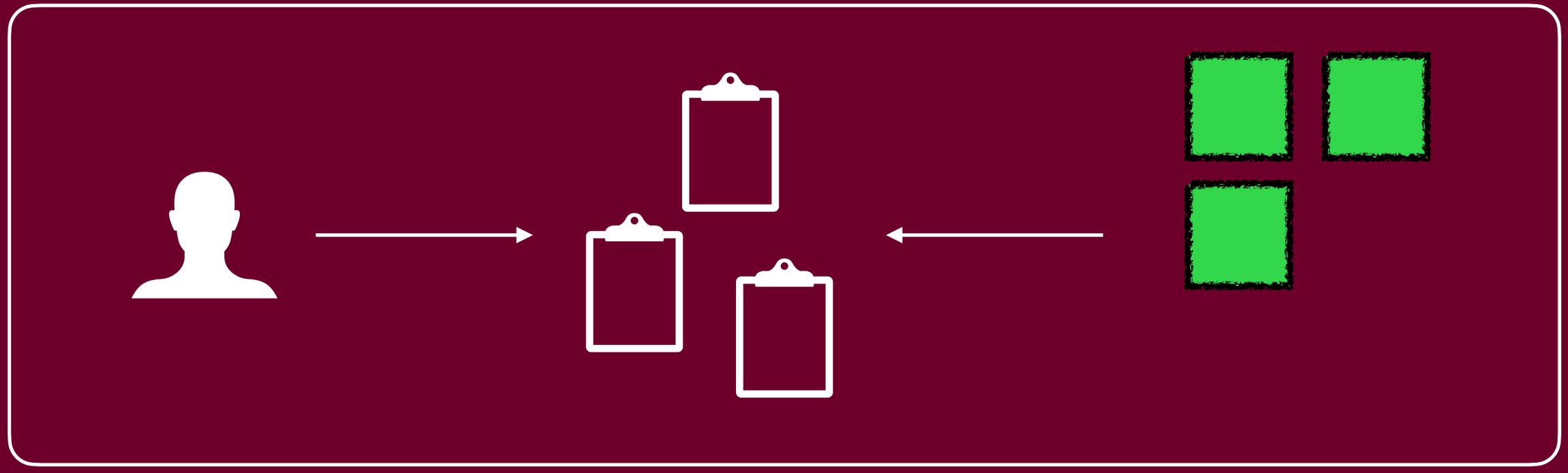
Test cases form the first line of defence against the introduction of software faults, especially when testing for regressions



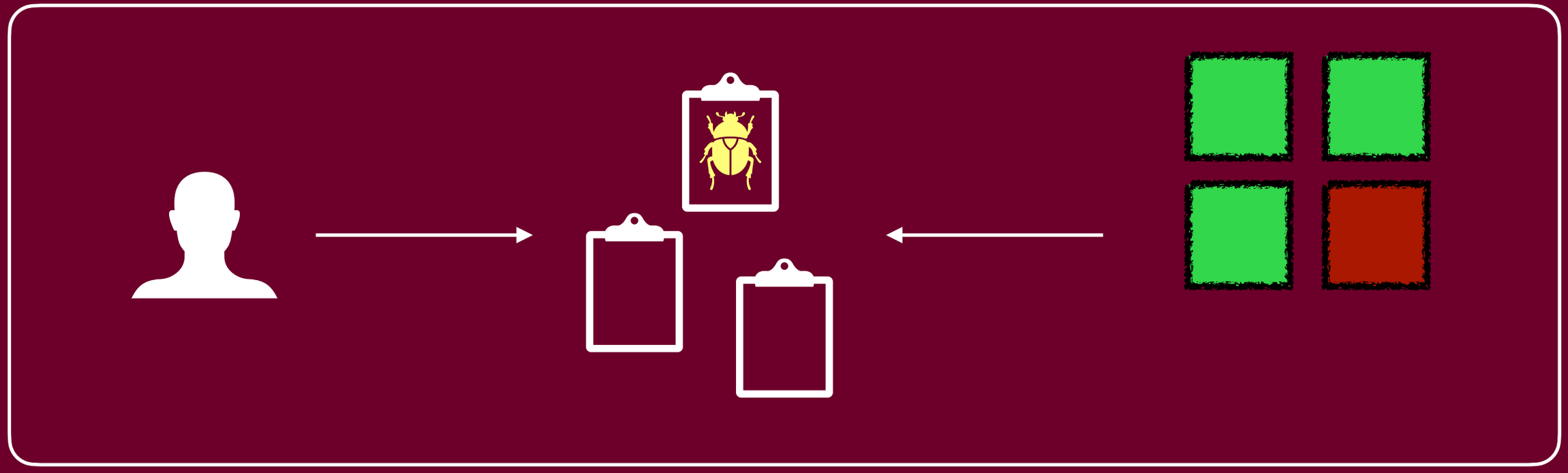
Test cases form the first line of defence against the introduction of software faults, especially when testing for regressions



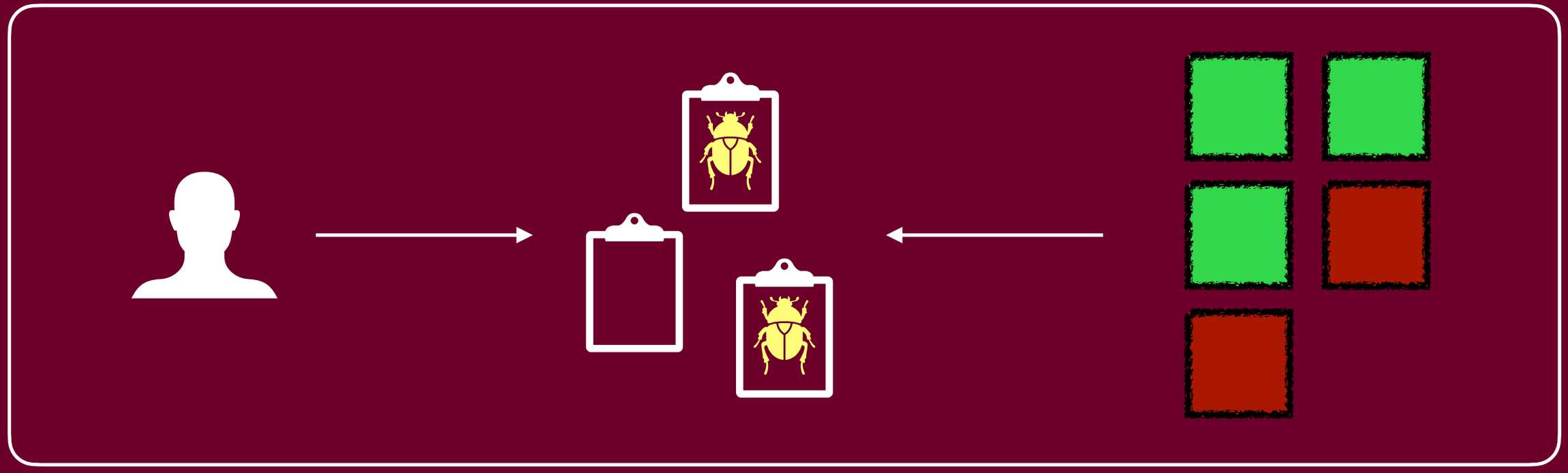
Test cases form the first line of defence against the introduction of software faults, especially when testing for regressions



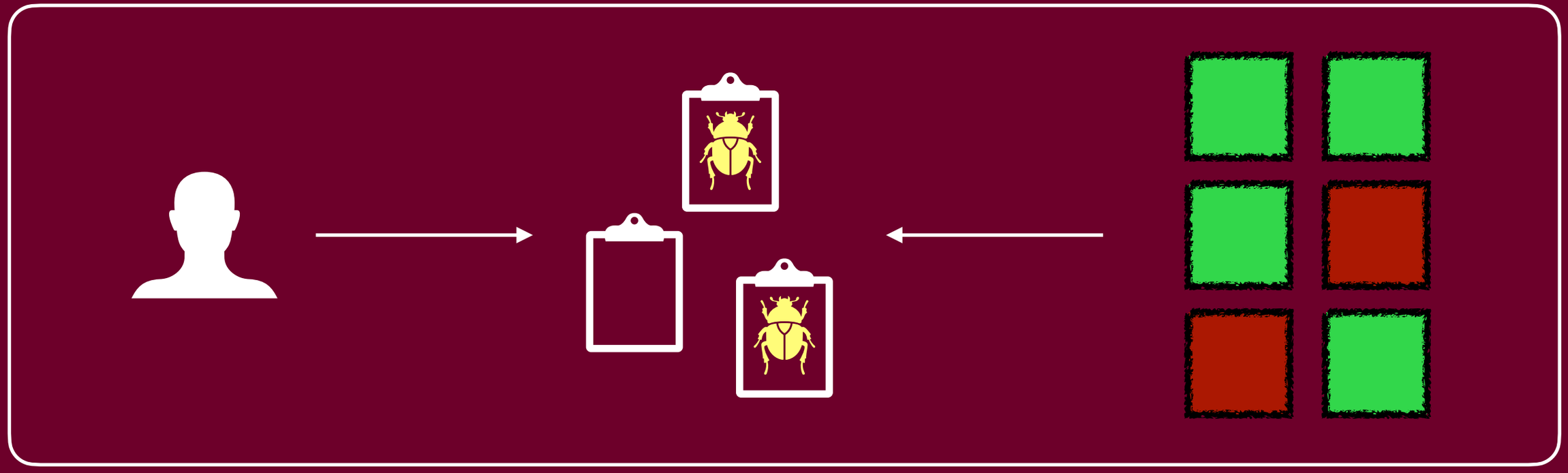
Test cases form the first line of defence against the introduction of software faults, especially when testing for regressions



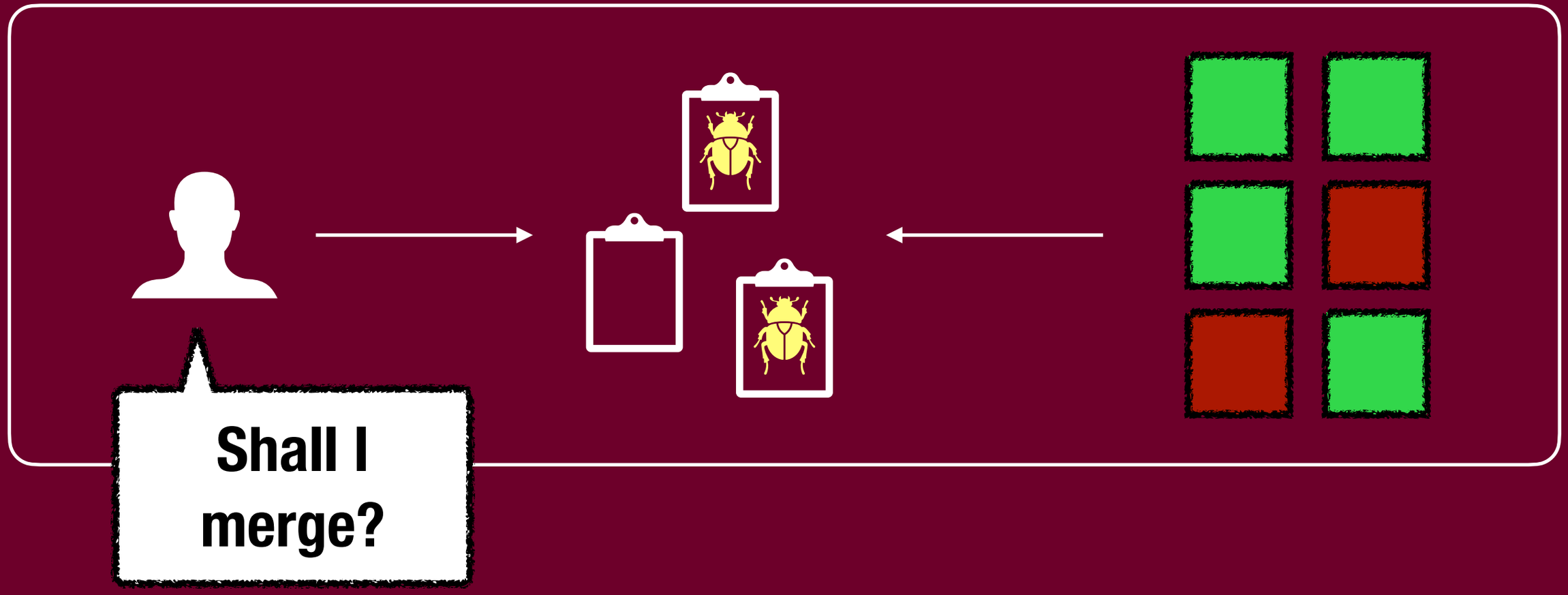
Test cases form the first line of defence against the introduction of software faults, especially when testing for regressions



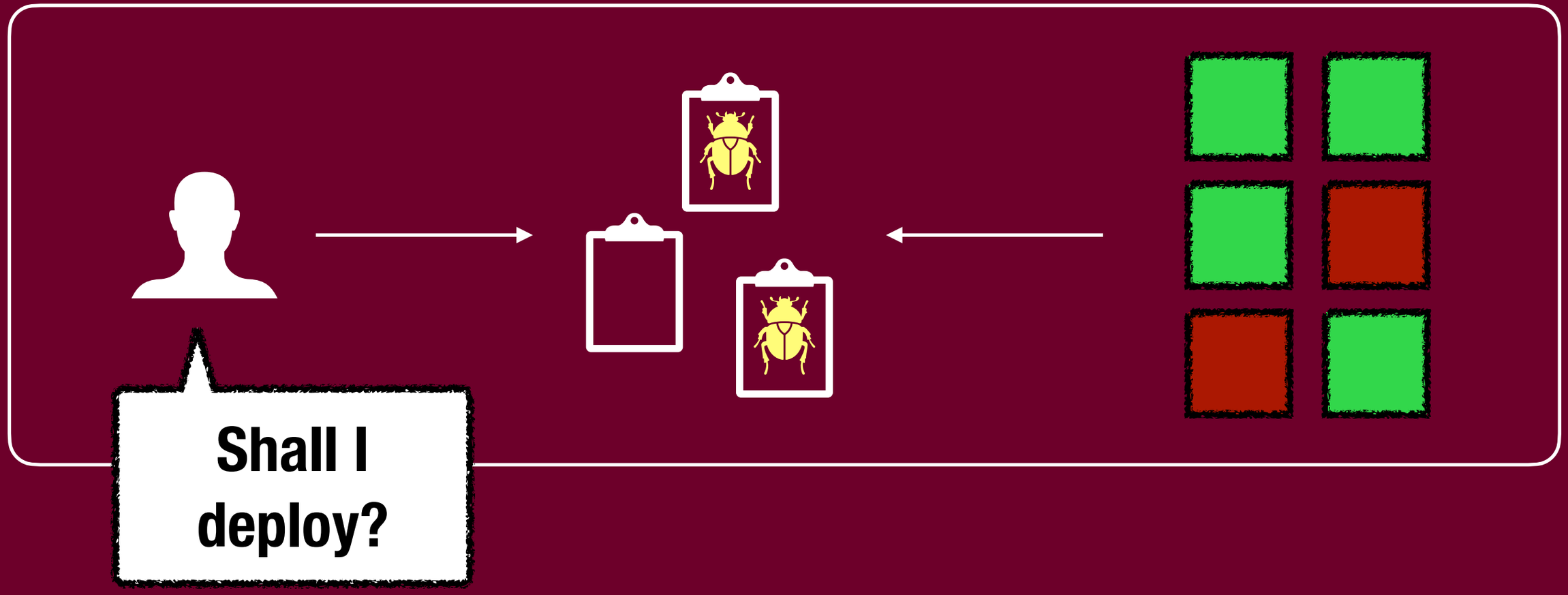
Test cases form the first line of defence against the introduction of software faults, especially when testing for regressions



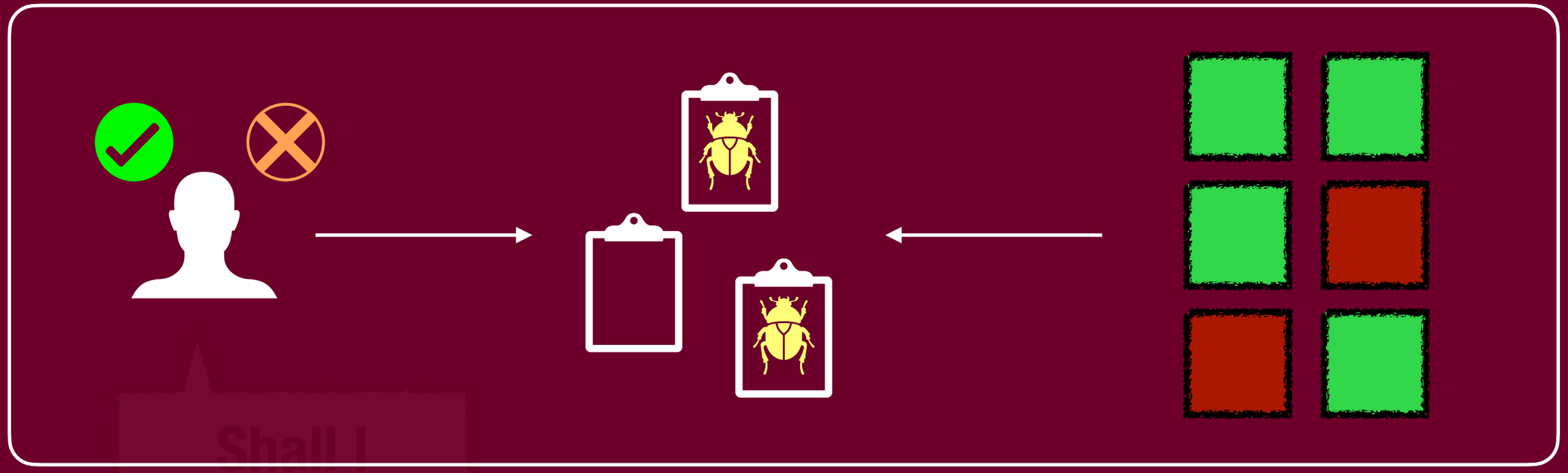
Test cases form the first line of defence against the introduction of software faults, especially when testing for regressions



Test cases form the first line of defence against the introduction of software faults, especially when testing for regressions

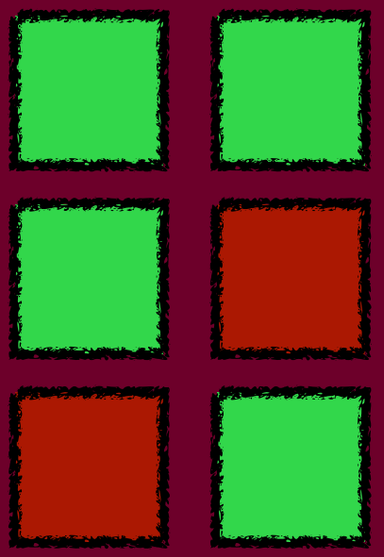


Test cases form the first line of defence against the introduction of software faults, especially when testing for regressions



Test cases form the first line of defence against the introduction of software faults, especially when testing for regressions

The scenario makes sense if the test suite is reliable!



Shall I deploy?

Even angels eat beans

Test code can be defective as well

Vahabzadeh et al.
“An Empirical Study of Bugs in Test Code”
ICSME 2015

Even angels eat beans

Test code can be defective as well

50% of software systems are affected by test code defects

Vahabzadeh et al.
“An Empirical Study of Bugs in Test Code”
ICSME 2015

Even angels eat beans

Test code can be defective as well

50% of software systems are affected by test code defects

Besides functional defects, a typical problem is called “flakiness”

Vahabzadeh et al.
“An Empirical Study of Bugs in Test Code”
ICSME 2015

Flaky Tests

"Test cases that exhibit both
a passing and failing result
with the same code."

Luo et al.
"An Empirical Analysis of Flaky Tests"
FSE 2014

Flaky Tests

"Test cases that exhibit both a passing and failing result with the same code."

They might hide real bugs, increase maintenance costs, and reduce developers' confidence

Luo et al.
"An Empirical Analysis of Flaky Tests"
FSE 2014

Empirical Studies

Investigations aimed at understanding specific causes leading to test flakiness (e.g., concurrency)

Luo et al., “An Empirical Analysis of Flaky Tests” - FSE 2014

Palomba and Zaidman - “The Smell of the Fear: On the Relation Between Test Smells and Flaky Tests” - EMSE 2019

Empirical Studies

Investigations aimed at understanding specific causes leading to test flakiness (e.g., concurrency)

Luo et al., “An Empirical Analysis of Flaky Tests” - FSE 2014

Palomba and Zaidman - “The Smell of the Fear: On the Relation Between Test Smells and Flaky Tests” - EMSE 2019

Bell et al., “DeFlaker: Automatically Detecting Flaky Tests” - ICSE 2018

Bell and Kaiser, “Unit Test Virtualization with VMVN” - ICSE 2014

Proposing solutions to automatically identify and fix flaky tests

Flaky Test Identification and Repair

Understanding Flaky Tests: The Developer's Perspective

Moritz Eck,* **Fabio Palomba**,* Marco Castelluccio,¶ Alberto Bacchelli*

*University of Zurich, Switzerland — ¶Mozilla Software Foundation, UK

Flaky Tests

Flaky tests seems to be a relevant issue and is highly discussed by practitioners on the web

Flaky Tests

Flaky tests seems to be a relevant issue and is highly discussed by practitioners on the web.

An improved understanding of the developer's perception w.r.t. **nature, relevance, and challenges of flaky tests** can inform SE approaches, tools, and research directions.

Research Questions

RQ_1	How do professional developers **categorise** and **fix** flaky tests?

Research Questions

RQ_1	How do professional developers **categorise** and **fix** flaky tests?
RQ_2	How **prominent** is test flakiness and how **problematic** is it as perceived by developers?

Research Questions

RQ_1	How do professional developers **categorise** and **fix** flaky tests?
RQ_2	How **prominent** is test flakiness and how **problematic** is it as perceived by developers?
RQ_3	What are the main **challenges** that developers face when dealing with flaky tests?

RQ_1 - Categorizing Test Flakiness

Mozilla
Software Foundation

RQ_1 - Categorizing Test Flakiness

05/2017

04/2018

Mozilla
Software Foundation

869 fixed or verified
flaky tests

RQ_1 - Categorizing Test Flakiness

05/2017

04/2018

Mozilla Software Foundation

869 fixed or verified flaky tests

304 flaky tests solved by experts

RQ_1 - Categorizing Test Flakiness

Mozilla
Software Foundation

869 fixed or verified
flaky tests

05/2017

04/2018

304 flaky tests
solved by experts

29 Mozilla
developers invited

RQ_1 - Categorizing Test Flakiness

Mozilla
Software Foundation

869 fixed or verified
flaky tests

05/2017

04/2018

304 flaky tests
solved by experts

29 Mozilla
developers invited

21 Mozilla developers
for 248 flaky tests

RQ_1 - Categorizing Test Flakiness

Mozilla
Software Foundation

869 fixed or verified
flaky tests

05/2017

04/2018

304 flaky tests
solved by experts

29 Mozilla
developers invited

21 Mozilla developers
for 248 flaky tests

Category
Origin
Effort

Classification coming
from developers

RQ_1 - Categorizing Test Flakiness

26%

Concurrency

22%

Async Wait

11%

Test Order Dependency

RQ_1 - Categorizing Test Flakiness

26%

Concurrency

46%

Add Await

22%

Async Wait

86%

Add Await

11%

Test Order Dependency

86%

Remove Dependency

RQ_1 - Categorizing Test Flakiness

10%

Too Restrictive Range

7%

Test Case Timeout

4%

Platform Dependency

2%

Test Suite Timeout

lency

RQ_1 - Categorizing Test Flakiness

10%

Too Restrictive Range

45%

Fix Assertion

7%

Test Case Timeout

85%

Increase Timeout

4%

Platform Dependency

60%

Run Additional Tests

2%

Test Suite Timeout

75%

Skip Non-Initialized Part

RQ_1 - Categorizing Test Flakiness

10%

Too Restrictive Range

45%

Fix Assertion

7%

Test Case Timeout

85%

Increase Timeout

Prod. Code

High Effort

4%

Platform Dependency

60%

Run Additional Tests

Prod. Code

High Effort

2%

Test Suite Timeout

75%

Skip Non-Initialized Part

Medium Effort

RQ_1 - Categorizing Test Flakiness

Up to 25%

Of flaky tests are just skipped or disabled

RQ_2-RQ_3 - Relevance and Challenges of Flaky Tests

1. How many times flaky tests occur
2. How problematic they are
3. What are the top 3 to 5 problems caused by test flakiness

RQ_2-RQ_3 - Relevance and Challenges of Flaky Tests

1. How many times flaky tests occur
2. How problematic they are
3. What are the top 3 to 5 problems caused by test flakiness

4. What are the challenges of dealing with flaky tests

RQ_2-RQ_3 - Relevance and Challenges of Flaky Tests

1. How many times flaky tests occur
2. How problematic they are
3. What are the top 3 to 5 problems caused by test flakiness

4. What are the challenges of dealing with flaky tests

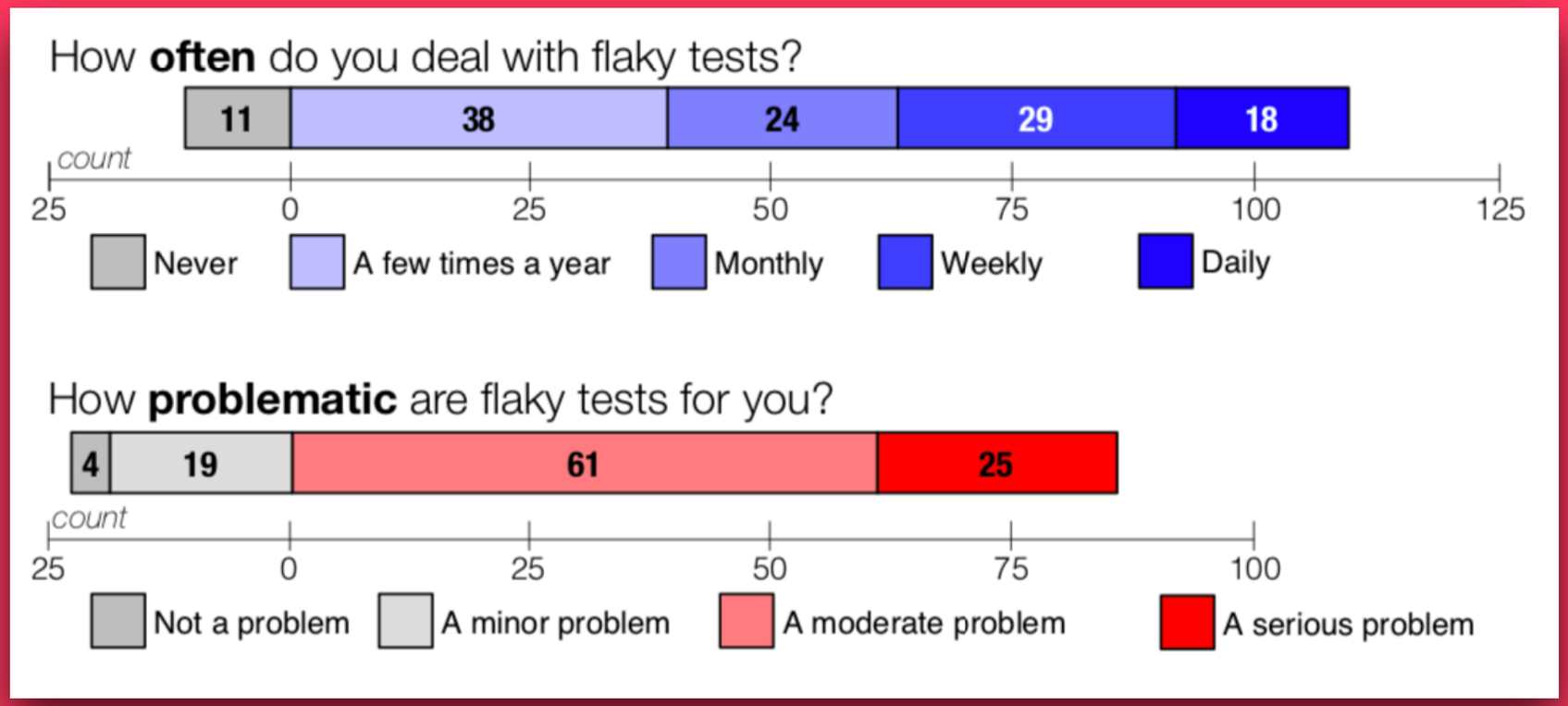
RQ_2-RQ_3 - Relevance and Challenges of Flaky Tests

1. How many times flaky tests occur
2. How problematic they are
3. What are the top 3 to 5 problems caused by test flakiness

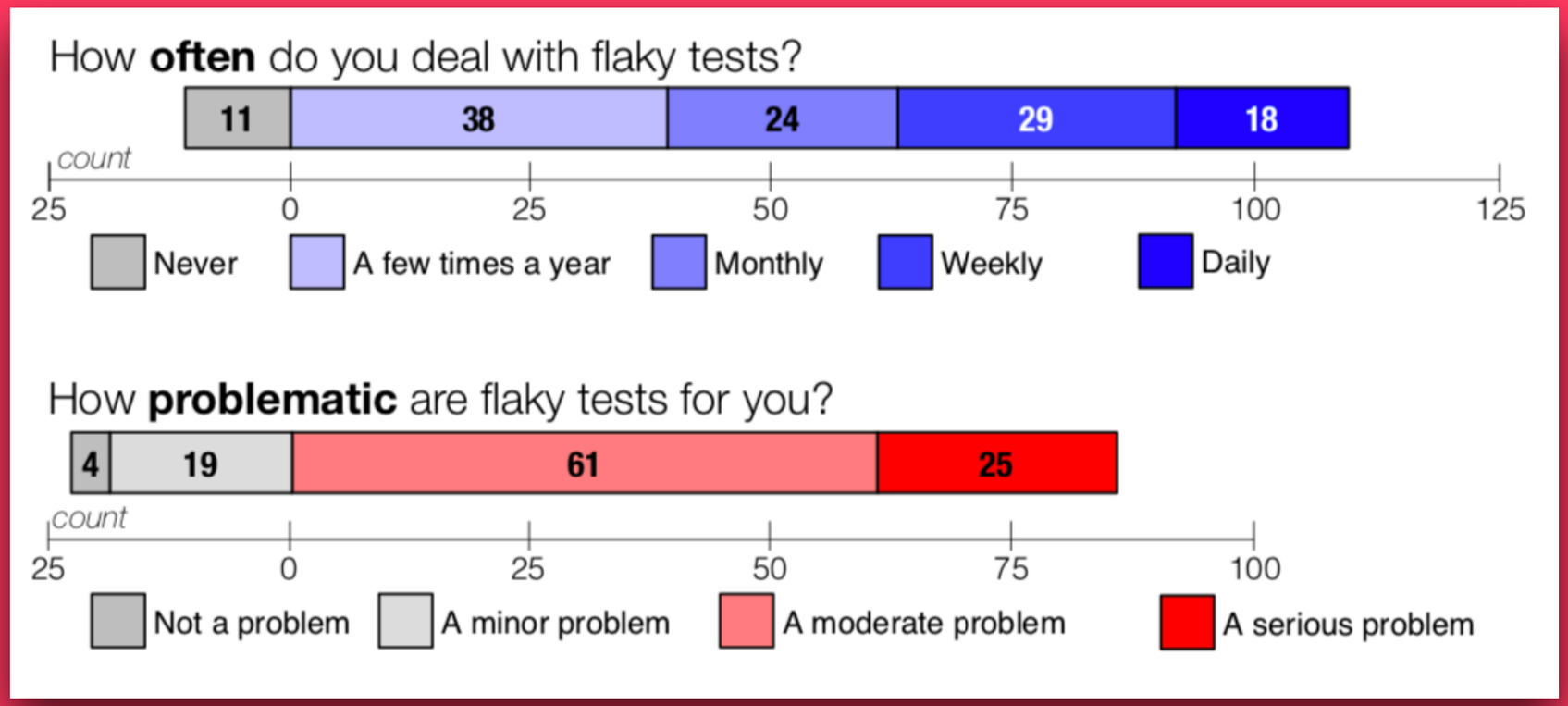
4. What are the challenges of dealing with flaky tests

188 responses (121 full responses)

RQ_2 - Relevance of Flaky Tests



RQ_2 - Relevance of Flaky Tests



Flaky tests are **rather frequent** and
a **non-negligible problem**

RQ_2 - Relevance of Flaky Tests

Flaky tests usually have a **lower priority than permanent defects**, and thus their **scheduling can be postponed** till the moment they will never fixed

RQ_2 - Relevance of Flaky Tests

Flaky tests usually have a **lower priority than permanent defects**, and thus their **scheduling can be postponed** till the moment they will never fixed

After finding a flaky test, the entire test suite is **no longer reliable**. Therefore, developers **may start disregarding it**, potentially leading to ignoring an actual failure.

RQ_2 - Relevance of Flaky Tests

Flaky tests usually have a **lower priority than permanent defects**, and thus their **scheduling can be postponed** till the moment they will never fixed

After finding a flaky test, the entire test suite is **no longer reliable**. Therefore, developers **may start disregarding it**, potentially leading to ignoring an actual failure.

Flaky tests **require a certain level of knowledge** to be able to fix them and, thus, intermittent tests may lead to problems in the **allocation of the available resources**.

RQ_3 - Challenges of Flaky Tests

The **context leading to failure** is the most challenging aspect when dealing with flakiness

RQ_3 - Challenges of Flaky Tests

The **context leading to failure** is the most challenging aspect when dealing with flakiness

The timely identification of the **nature of the flakiness** is key for developers

RQ_3 - Challenges of Flaky Tests

The **context leading to failure** is the most challenging aspect when dealing with flakiness

The timely identification of the **nature of the flakiness** is key for developers

Getting the **origin of test flakiness** can substantially reduce the fixing time

RQ_3 - Challenges of Flaky Tests

The **context leading to failure** is the most challenging aspect when dealing with flakiness

The timely identification of the **nature of the flakiness** is key for developers

Getting the **origin of test flakiness** can substantially reduce the fixing time

RQ_3 - Challenges of Flaky Tests

The **context leading to failure** is the most challenging aspect when dealing with flakiness

The timely identification of the **nature of the flakiness** is key for developers

Getting the **origin of test flakiness** can substantially reduce the fixing time

Design for testability! Definition of best practices for mocking dependencies and keeping tests decoupled

RQ_3 - Challenges of Flaky Tests

The **context leading to failure** is the most challenging aspect when dealing with flakiness

The timely identification of the **nature of the flakiness** is key for developers

Getting the **origin of test flakiness** can substantially reduce the fixing time

Design for testability! Definition of best practices for mocking dependencies and keeping tests decoupled

Assessing the Fixing Process. Measuring uncertainty of changes fixing the test and empowering log files with flakiness information

To sum up: What is needed

Flaky tests are relevant for developers. More research on their **root causes**!

To sum up: What is needed

Flaky tests are relevant for developers. More research on their **root causes**!

The need of **an organisational view of flaky tests**. More techniques for flaky test triage!

To sum up: What is needed

Flaky tests are relevant for developers. More research on their **root causes**!

The need of **an organisational view of flaky tests**. More techniques for flaky test triage!

The definition of **best and bad practices to deal with flakiness**.

To sum up: What is needed

Flaky tests are relevant for developers. More research on their **root causes**!

The need of **an organisational view of flaky tests**. More techniques for flaky test triage!

The definition of **best and bad practices to deal with flakiness**.

Methods able to provide developers with **actionable information on flaky tests.**

Understanding Flaky Tests: The Developer's Perspective

Moritz Eck,* **Fabio Palomba**,* Marco Castelluccio,¶ Alberto Bacchelli*

*University of Zurich, Switzerland — ¶Mozilla Software Foundation, UK

Causes of flakiness

Nature: Concurrency	Cases: 61 | Avg. Fixing Effort: 4.0 | Origin: 66% Test Code	
Fixing Strategies:		
Name	Description	Frequency
Wait For Addition	An `await` or `waitFor` promise is added to wait for the required event.	46%
Change Concurrency Guard Conditions	Modification of the constraints that may block the applicability of the clause under some conditions.	26%
Adding Lock	New locks are introduced to ensure mutual exclusion between threads.	21%
Make Code Deterministic	The concurrency is removed.	5%
Test Replacement	The flaky test is replaced with a brand new test.	1%
Disable Test	The flaky test is disabled and no longer executed.	1%

Nature: Async Wait	Cases: 52 | Avg. Fixing Effort: 3.0 | Origin: 100% Test Code	
Wait For Addition	An `await` or `waitFor` promise is added to wait for the required event.	86%
Reordering Threads Execution	The source code is reorganized or refactored to make the execution deterministic or less async.	13%
Disable Test	The flaky test is disabled and no longer executed.	1%

Nature:	Cases: 40 | Avg. Fixing Effort: 1.0 | Origin: 100% Test Code	
Fix Assertion	The cause leading to the failure is diagnosed and fixed.	45%
Adjust Fuzzing	The `assert` statement is modified so that it provides less extreme values or allow larger difference when comparing expected and actual outcome.	20%
Disable Test	The flaky test is disabled and no longer executed.	16%
Missing Code Added	The code needed to check whether a certain condition is met is added.	1%

Nature: Test Order Dependency	Cases: 22 | Avg. Fixing Effort: 2.0 | Origin: 100% Test Code	
Remove Dependency	Changes aimed at (i) modifying the output directory for the test to use a separate directory or (ii) checking instance variables before that the test is accessed and executed.	100%

Nature: *Test Case Timeout	Cases: 18 | Avg. Fixing Effort: 4.0 | Origin: 100% Test Code	
Increase Timeout	The timeout time is increased to avoid the flakiness behavior.	85%
Skip Non-Initialized Part	Code added to skip non-initialized parts to make the test run faster and not timeout.	5%
Split Test	Split up all tests (run in parallel) into more groups to reduce risk of a timeout.	5%
Disable Test	The flaky test is disabled and no longer executed.	5%

Causes of flakiness

Nature: Resource Leak	Cases: 14 | Avg. Fixing Effort: 3.0 | Origin: 85% Test Code	
Destroy Object	New code is added so that the conflicting object is destroyed before continuing the execution.	50%
Test Replacement	The flaky test is replaced with a brand new test.	36%
Disable Test	The flaky test is disabled and no longer executed.	14%

Nature: *Platform Dependency	Cases: 10 | Avg. Fixing Effort: 4.0 | Origin: 90% Test Code	
Run Additional Tests	New platform-specific tests are ran instead of the flaky one.	60%
Correct Directories	The test suite is modified so that it includes the correct platforms to run the tests on or to fix a directory issue.	40%

Nature: Float Precision	Cases: 6 | Avg. Fixing Effort: 4.0 | Origin: 100% Test Code	
Subtract Bytecode Offset	The floating point is corrected by subtracting the bytecode offset so that the precision is preserved.	100%

Nature: *Test Suite Timeout	Cases: 4 | Avg. Fixing Effort: 3.5 | Origin: 100% Test Code	
Skip Non-Initialized Part	Code added to skip non-initialized parts to make the test run faster and not timeout.	75%
Split Test Suite	Split up all tests (run in parallel) into more groups to reduce risk of a timeout.	25%

Nature: Time	Cases: 4 | Avg. Fixing Effort: 1.0 | Origin: 100% Test Code	
Disable Test	The flaky test is disabled and no longer executed.	75%
Preserve Time Precision	The test case is modified so that the time precision is preserved.	25%

Nature: Randomness	Cases: 3 | Avg. Fixing Effort: 1.0 | Origin: 100% Test Code	
Replaced Math.random	The `Math.random` call is replaced with more reliable random number generator.	100%

Causes of flakiness

*__Test Suite Timeout.__ Flaky tests originating because of the test suite non-deterministically timing out are classified as Test Suite Timeout flakiness. Test suites grow over time and the max runtime value is not always adjusted accordingly, leading to the test suites passing or failing depending on different random variables (e.g., the network congestion, the execution speed of the platform or the type of build). Note that in a `Test Suite Timeout`, no single test is the cause of flakiness, rather the whole execution; this makes this cause different from `Test Case Timeout`, whose cause is a specific test. This flakiness cause appears very rarely (only 4 cases in our dataset) and the developers assigned a median effort score of 3.5. As this is test issue-related problem, its origin is within test files. To fix the flakiness, developers frequently decide to take actions able to fasten the execution of the test suite (75% of the cases), while the remaining 25% of times they perform an Extract Class refactoring [17] to make the resulting suites more independent and faster.

*__Test Case Timeout.__ Flaky tests experiencing non-deterministic timeouts related to a single test belong to this category. It is comparable to the Test Suite Timeout (reported later), with the difference that the size of a single test grew over time without adjusting the max runtime value. Various reasons (e.g., failing to download prerequisites or a test not producing output for a fixed amount of time, which then is killed by the execution system assuming that the test stalled) can lead to the non-deterministic outcome. In our dataset, this flakiness type appears in 18 tests (8%). Moreover, it is associated with a high effort to fix: despite it might seem that this problem would be only concerned with the increasing of the timeout, practitioners explained that finding the right timeout is unexpectedly hard. The increase of the timeout time is, obviously, the most frequent solution to this type of flakiness (85%), however other fixing strategies such as (i) the addition of code that make the test faster or (ii) the extraction of a new test to reduce the risk of timeout are also sometimes taken into account.

Complete list of challenges

Information on the flaky test	Importance for fixing	Difficulty in obtaining
C$_f$ - The context leading to failure	2.69	1.65
N - The nature of the flakiness	2.40	1.71
O - The origin of the flakiness	2.22	1.17
E - The involved code elements	2.21	1.13
C - The changes to perform the fix	2.08	1.59
C$_p$ - The context leading to passing	1.95	1.20
Co - The commit introducing the flakiness	1.89	0.75
H - The history of this test's flakiness (previous causes and fixes)	1.79	0.83

Complete list of challenges

Category	Challenge	# of mentions
Design	Mocking Dependencies	6
	Keeping Tests Decoupled	6
	Reliance on External Dependencies	3
	Too Much Setup Code	3
	Common Coding Style / Etiquette	1
Fixing	Uncertainty of Changes Fixing The Test	7
	Not Detailed Enough Log	4
	Lack of Insight Into The System	4
	Restructuring The Tests	2