Assured LLM-Based Software Engineering

ECSS 2024 keynote

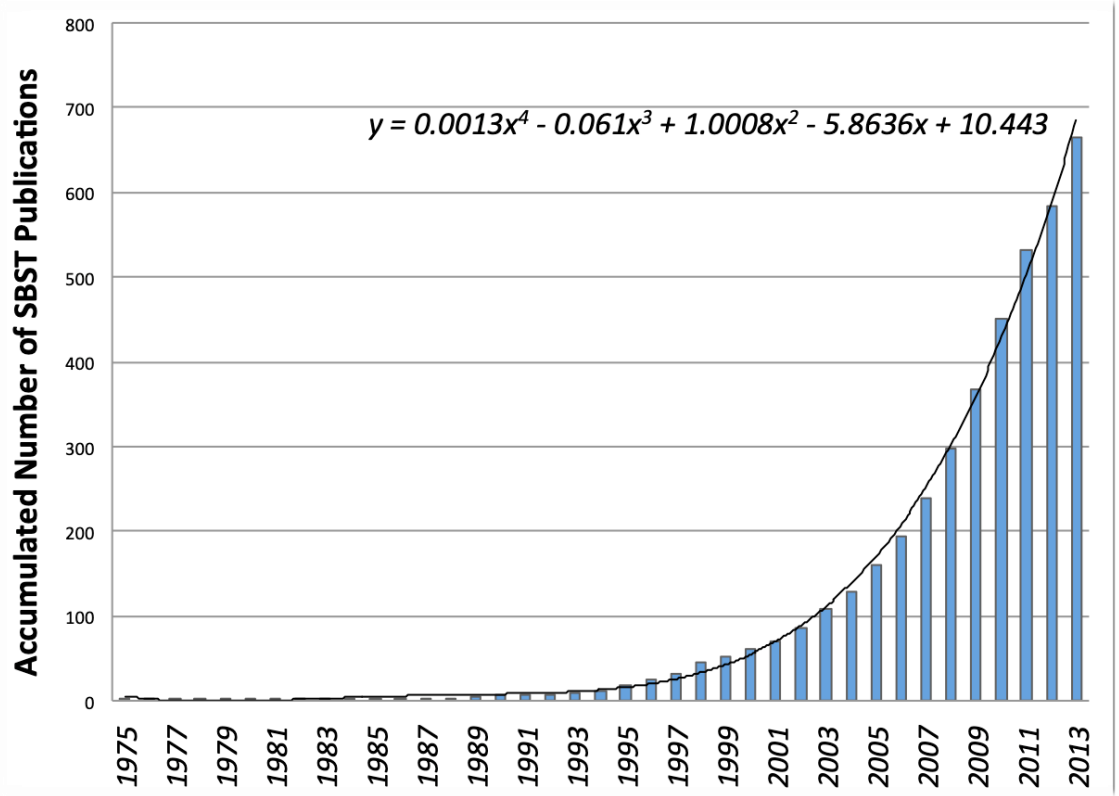
Mark Harman
29th October 2024

Joint work with

Nadia Alshahwan, Andrea Aquino, Jubin Chheda, Anastasia Finegenova, Inna Harper, Mitya Lyubarskiy, Neil Maiden, Alexander Mols, Shubho Sengupta, Alexandru Marginean, and Eddy Wang.

A brief note about how I got here

Search Based Software Engineering



Polynomial yearly rise in the number of papers Search Based Software Testing

Search Based Software Engineering

Achievements, open problems and challenges for search based software testing

Mark Harman, Yue Jia and Yuanyuan Zhang
University College London, CREST Centre, London, UK

***Abstract*—Search Based Software Testing (SBST) formulates testing as an optimisation problem, which can be attacked using computational search techniques from the field of Search Based Software Engineering (SBSE). We present an analysis of the SBST research agenda[1], focusing on the open problems and challenges of testing non-functional properties, in particular a topic we call 'Search Based Energy Testing' (SBET), Multi-objective SBST and SBST for Test Strategy Identification. We conclude with a vision of FIFIVERIFY tools, which would automatically find faults, fix them and verify the fixes. We explain why we think such FIFIVERIFY tools constitute an exciting challenge for the SBSE community that already could be within its reach.**

I. Introduction

Search Based Software Testing (SBST) is the sub-area of Search Based Software Engineering (SBSE) concerned with software testing [2], [85]. SBSE uses computational search techniques to tackle software engineering problems (testing problems in the case of SBST), typified by large complex search spaces [58]. Test objectives find natural counterparts as the fitness functions used by SBSE to guide automated search, thereby facilitating SBSE formulations of many (and diverse) testing problems. As a result, SBST has proved to be a widely applicable and effective way of generating test data, and optimising the testing process. However, there are many exciting challenges and opportunities that remain open for further research and development, as we will show in this paper.

It is widely believed that approximately half the budget spent on software projects is spent on software testing, and therefore, it is not surprising that perhaps a similar proportion of papers in the software engineering literature are concerned with software testing. We report an updated literature analysis from which we observe that approximately half of all SBSE papers are SBST papers, a figure little changed since the last thorough publication audit (for papers up to 2009), which found 54% of SBSE papers concerned SBST [56]. Many excellent and detailed surveys of the SBST literature can be found elsewhere [2], [4], [55], [85], [126]. Therefore, rather than attempting another survey, we provide an analysis of SBST research trends, focusing on open challenges and areas for future work and development.

[1]This keynote was given by Mark Harman at the 8th IEEE International Conference on Software Testing, Verification and Validation (ICST 2015), but this paper, on which the keynote was based, is the work of all three authors.

II. A Brief History of SBST

Since the first paper on SBST is also likely to be the first paper on SBSE, the early history of SBST is also the early history of SBSE. SBSE is a sub-area of software engineering with origins stretching back to the 1970s but not formally established as a field of study in its own right until 2001 [51], and which only achieved more widespread acceptance and uptake many years later [38], [43], [100].

The first mention of *software optimisation* (of any kind) is almost certainly due to Ada Augusta Lovelace in 1842. Her English language translation of the article (written in Italian by Menabrae), 'Sketch of the Analytical Engine Invented by Charles Babbage' includes seven entries, labelled 'Note A' to 'Note G' and initialed 'A.A.L'. Her notes constituted an article themselves (and occupied three quarters of the whole document). In these notes we can see perhaps the first recognition of the need for software optimisation and source code analysis and manipulation (a point argued in more detail elsewhere [44]):

> "*In almost every computation a great variety of arrangements for the succession of the processes is possible, and various considerations must influence the selection amongst them for the purposes of a Calculating Engine. One essential object is to choose that arrangement which shall tend to reduce to a minimum the time necessary for completing the calculation.*" Extract from 'Note D'.

The introduction of the idea of software testing is probably due to Turing [115], who suggested the use of manually constructed assertions. In his short paper, we can find the origins of both *software testing* and *software verification*. The first use of *optimisation techniques* in software testing and verification probably dates back to the seminal PhD thesis by James King [67], who used automated symbolic execution to capture path conditions, solved using linear programming. The first formulation of the test input space as a *search space* probably dates back seven years earlier to 1962, when a Cobol test data generation tool was introduced by Sauder [103]. Sauder formulates the test generation problem as one of finding test inputs from a search space, though the search algorithm is *random search*, making this likely to be the first paper on Random Test Data Generation. Sauder's work is also significant because it introduces the idea of constraints to capture path conditions, although these constraints are manually defined and not automatically constructed.

Testing is a search process

Searching for test cases
Searching for test application orders
Searching for patches

Search Based Software Engineering

Sapienz: Multi-objective Automated Testing for Android Applications

Ke Mao Mark Harman Yue Jia
CREST Centre, University College London, Malet Place, London, WC1E 6BT, UK
k.mao@cs.ucl.ac.uk, mark.harman@ucl.ac.uk, yue.jia@ucl.ac.uk

ABSTRACT

We introduce SAPIENZ, an approach to Android testing that uses multi-objective search-based testing to automatically explore and optimise test sequences, minimising length, while simultaneously maximising coverage and fault revelation. SAPIENZ combines random fuzzing, systematic and search-based exploration, exploiting seeding and multi-level instrumentation. SAPIENZ significantly outperforms (with large effect size) both the state-of-the-art technique Dynodroid and the widely-used tool, Android Monkey, in 7/10 experiments for coverage, 7/10 for fault detection and 10/10 for fault-revealing sequence length. When applied to the top 1,000 Google Play apps, SAPIENZ found 558 unique, previously unknown crashes. So far we have managed to make contact with the developers of 27 crashing apps. Of these, 14 have confirmed that the crashes are caused by real faults. Of those 14, six already have developer-confirmed fixes.

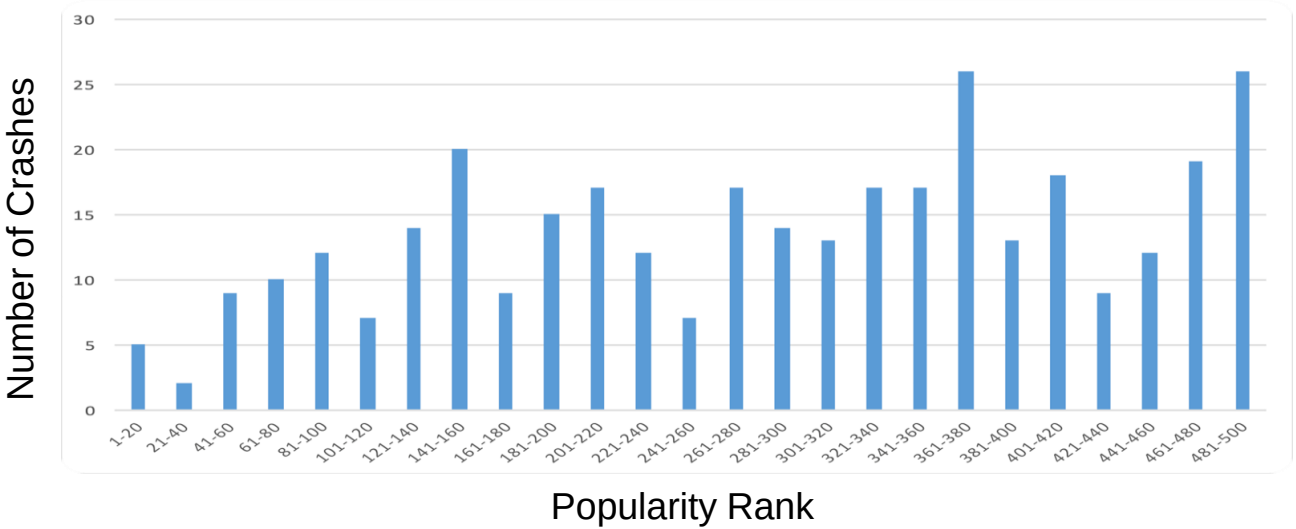
Where test automation does occur, it typically uses Google's Android Monkey tool [36], which is currently integrated with the Android system. Since this tool is so widely available and distributed, it is regarded as the current state-of-practice for automated software testing [53]. Although Monkey automates testing, it does so in a relatively unintelligent manner: generating sequences of events at random in the hope of exploring the app under test and revealing failures. It uses a standard, simple-but-effective, default test oracle [22] that regards any input that reveals a crash to be a fault-revealing test sequence.

Automated testing clearly needs to find such faults, but it is no good if it does so with exceptionally long test sequences. Developers may reject longer sequences as being impractical for debugging and also unlikely to occur in practice; the longer the generated test sequence, the less likely it is to occur in practice. Therefore, a critical goal for automated testing is to find faults with the *shortest possible* test

Search Based Software Engineering

Sapienz: Multi-objective Automated Testing for Android Applications

Ke Mao Mark Harman Yue Jia
CREST Centre, University College London, Malet Place, London, WC1E 6BT, UK
k.mao@cs.ucl.ac.uk, mark.harman@ucl.ac.uk, yue.jia@ucl.ac.uk

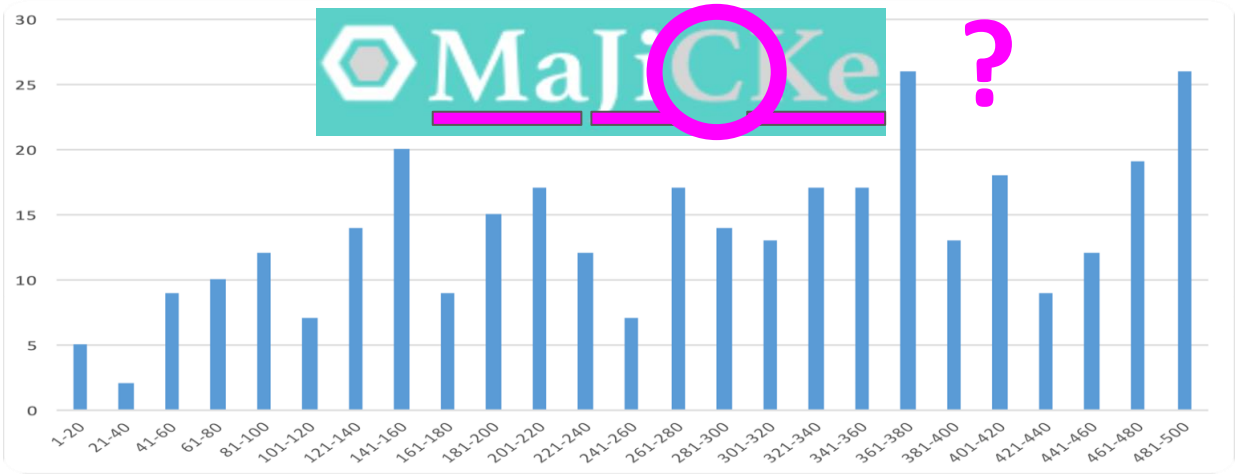


Search Based Software Engineering

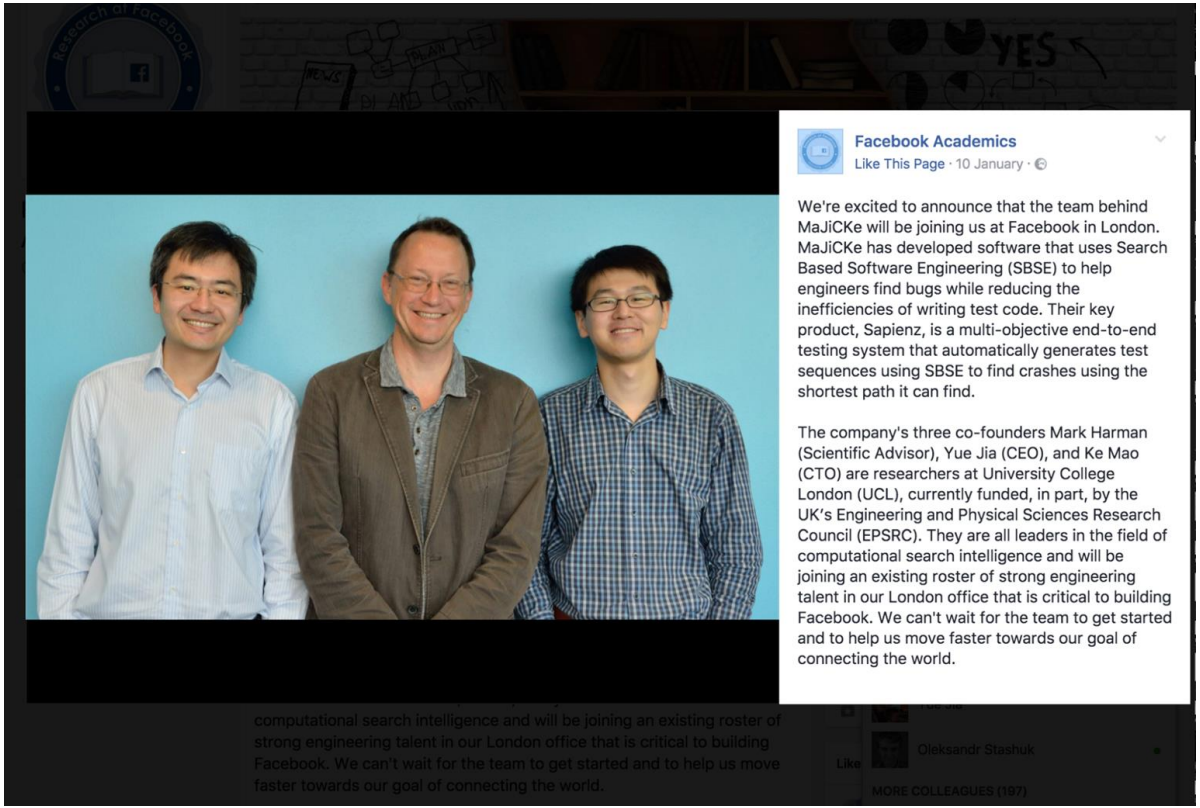
Sapienz: Multi-objective Automated Testing for Android Applications

Ke Mao Mark Harman Yue Jia
CREST Centre, University College London, Malet Place, London, WC1E 6BT, UK
k.mao@cs.ucl.ac.uk, mark.harman@ucl.ac.uk, yue.jia@ucl.ac.uk

MaJiCKe ?



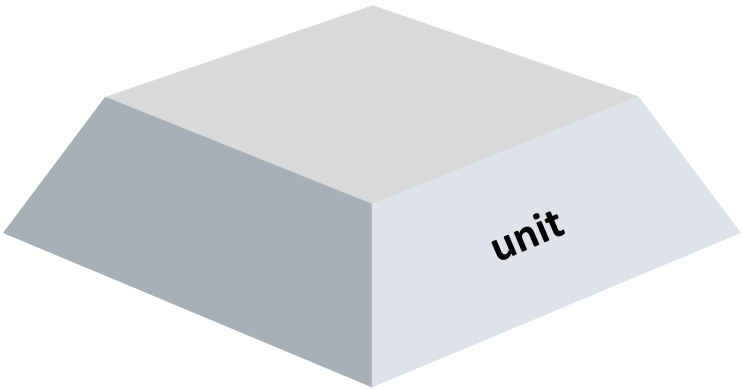
Search Based Software Engineering



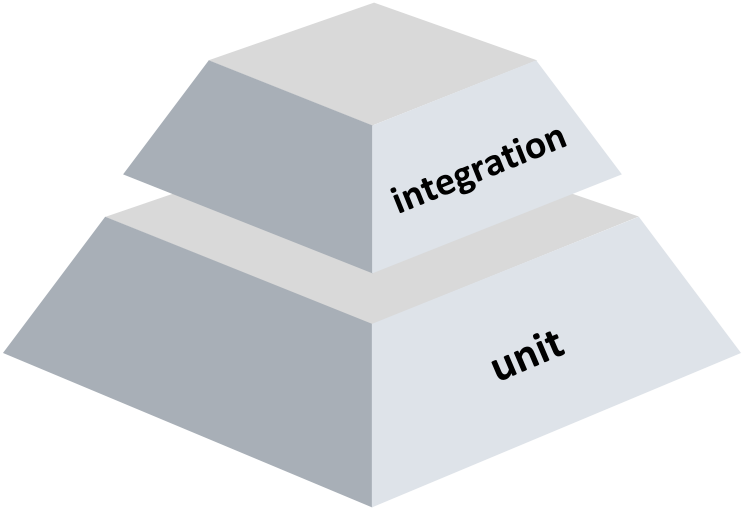
Challenges in Generating unit and integration tests

Software Testing 101

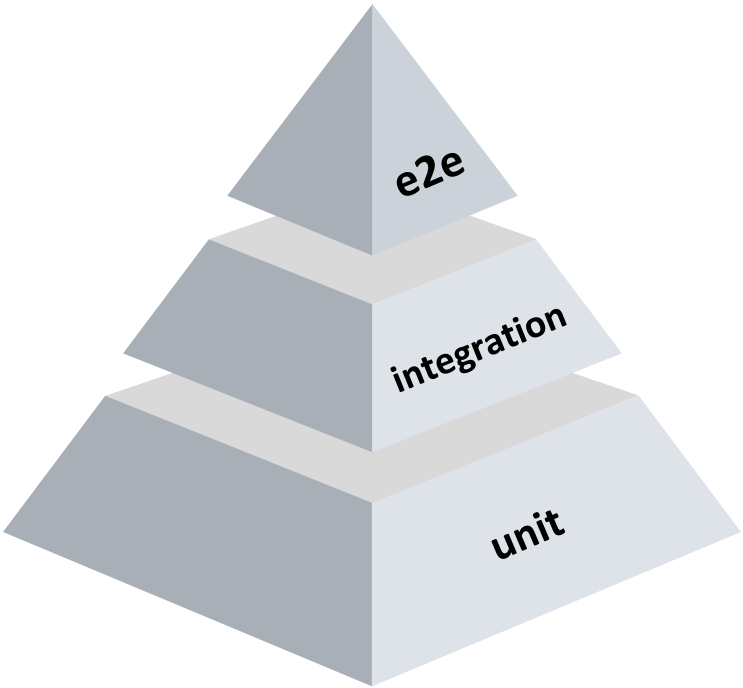
Traditional testing pyramid



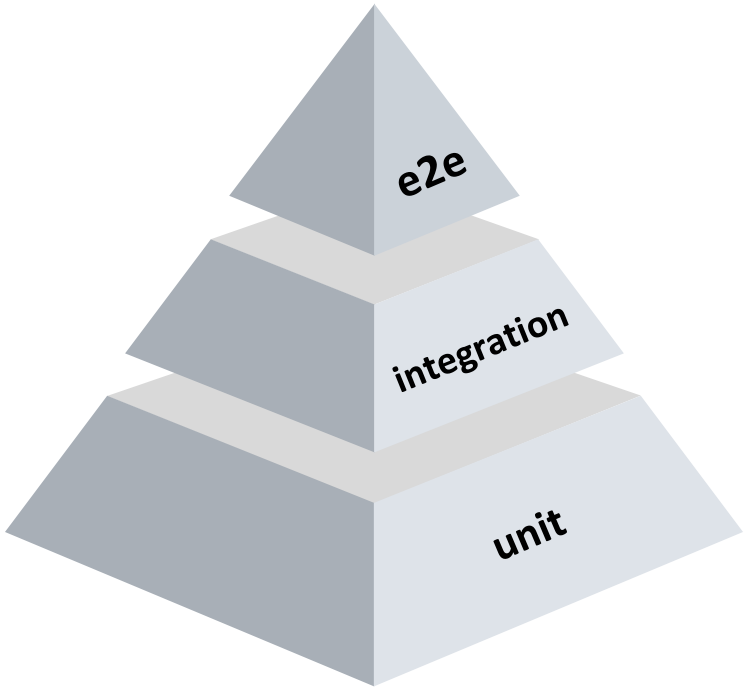
Traditional testing pyramid



Traditional testing pyramid

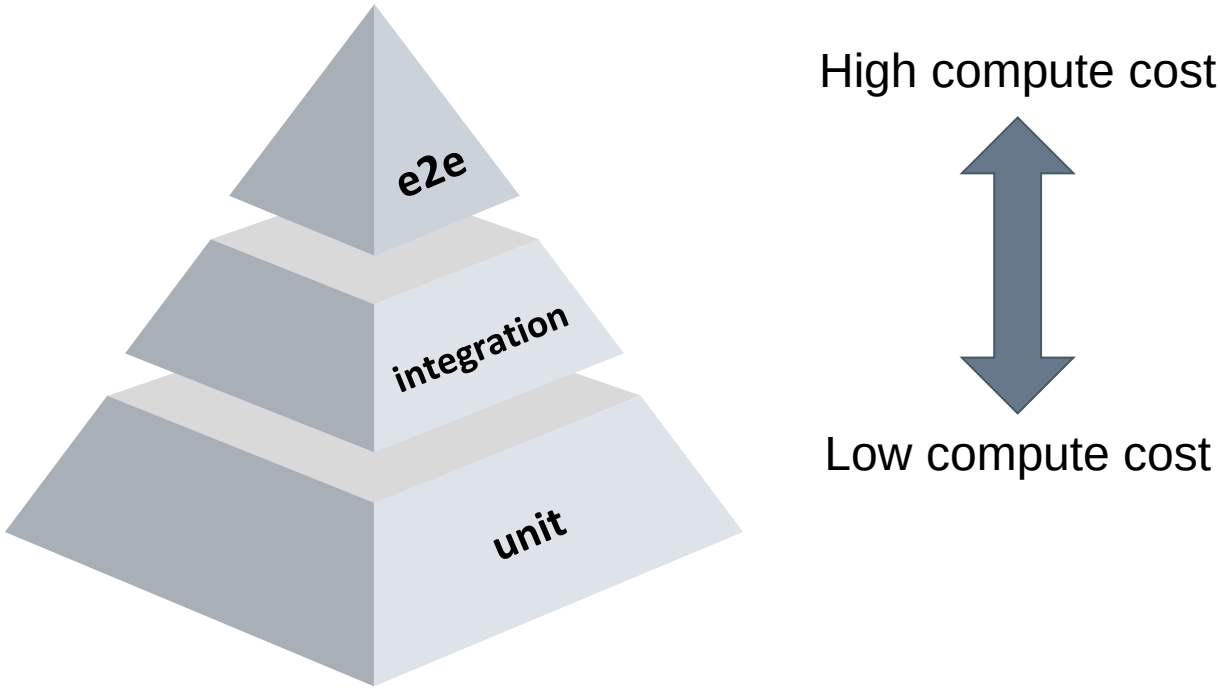


Traditional testing pyramid

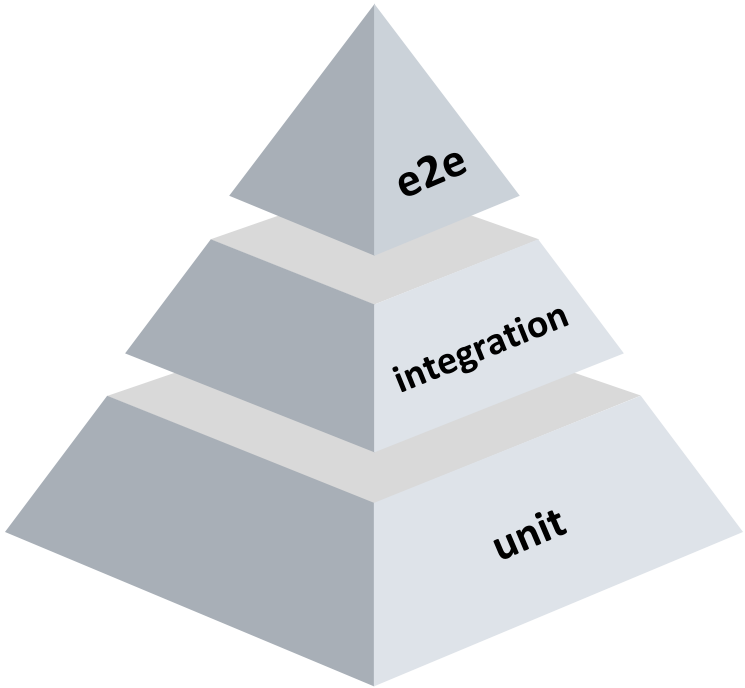


Unit and integration distinction is blurry

Traditional testing pyramid



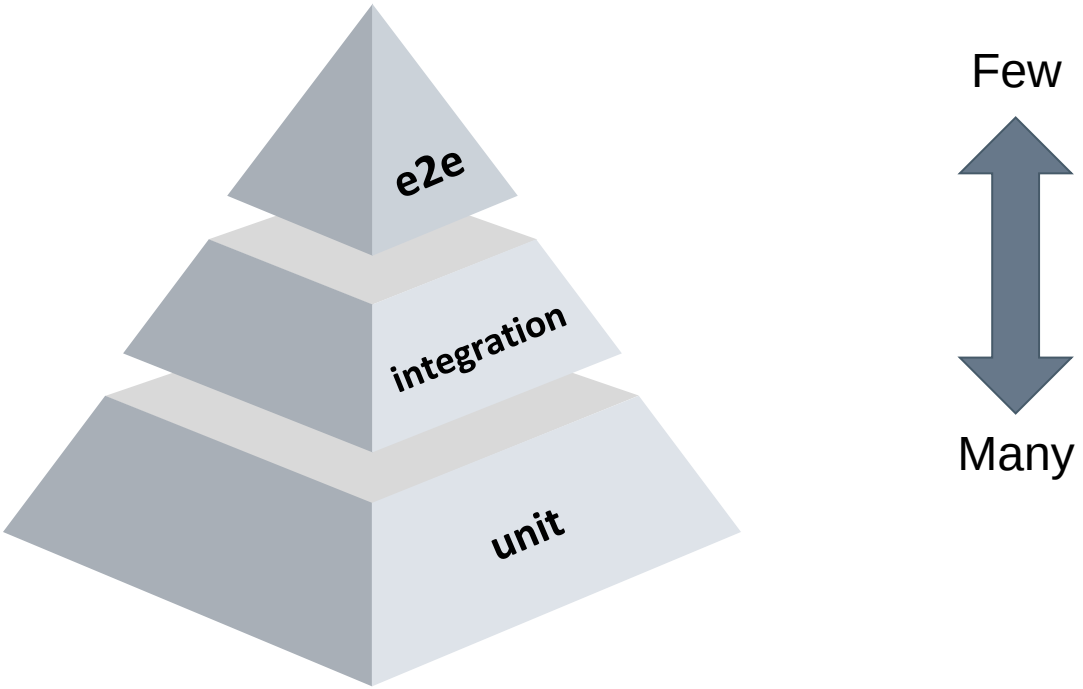
Traditional testing pyramid



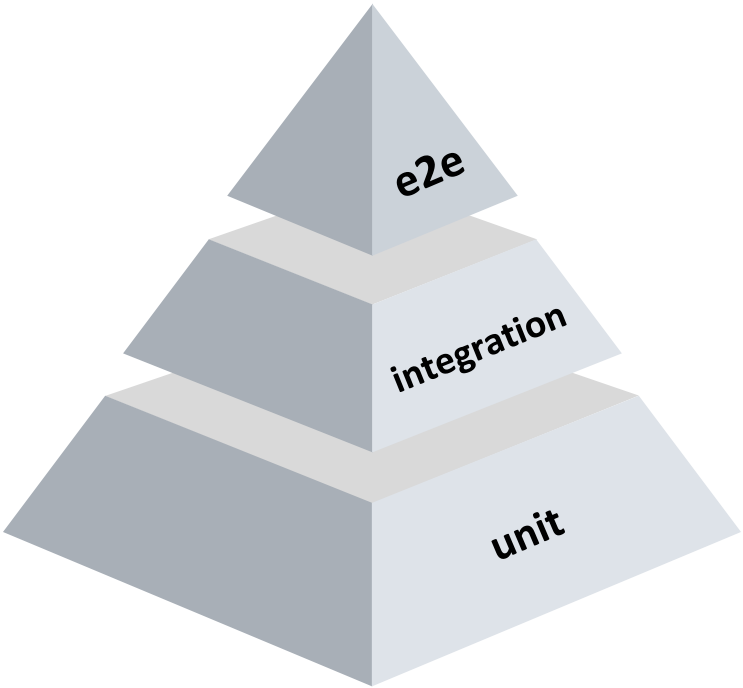
High human effort

Low human effort

Traditional testing pyramid - in theory



Traditional testing pyramid - in practice

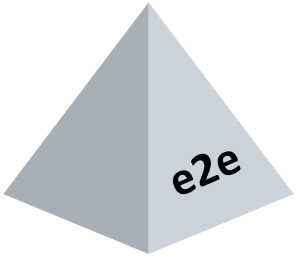


Some tests.
Non trivial human effort

Some tests.
Sometimes
unexpectedly
high
human effort

Let’s auto generate unit and integration tests

Search Based Test Generation



Where to start?

sapienz

Deploying Search Based Software Engineering with Sapienz at Facebook

Nadia Alshahwan, Xinbo Gao, Mark Harman(✉), Yue Jia, Ke Mao, Alexander Mols, Taijin Tei, and Ilya Zorin

Facebook, London, UK
{markharman,kemao}@fb.com

Abstract. We describe the deployment of the Sapienz Search Based Software Engineering (SBSE) testing system. Sapienz has been deployed in production at Facebook since September 2017 to design test cases, localise and triage crashes to developers and to monitor their fixes. Since then, running in fully continuous integration within Facebook's production development process, Sapienz has been testing Facebook's Android app, which consists of millions of lines of code and is used daily by hundreds of millions of people around the globe.

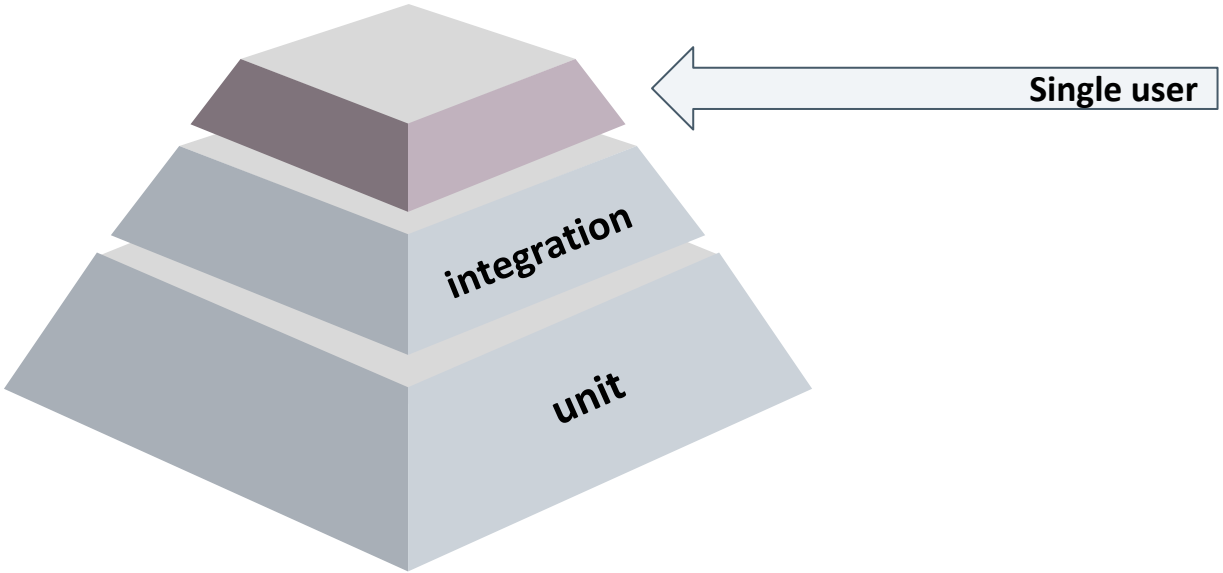
We continue to build on the Sapienz infrastructure, extending it to provide other software engineering services, applying it to other apps and platforms, and hope this will yield further industrial interest in and uptake of SBSE (and hybridisations of SBSE) as a result.

75% Fix rate

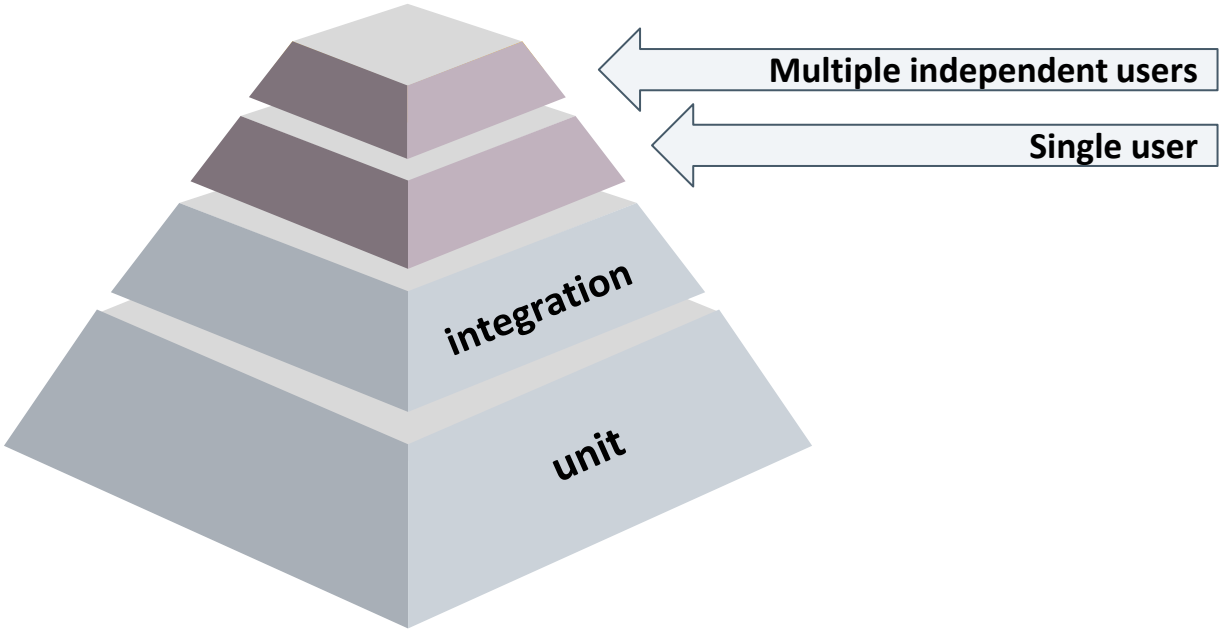
SIMULATION-BASED TESTING

Industry feeding back to scientific base …

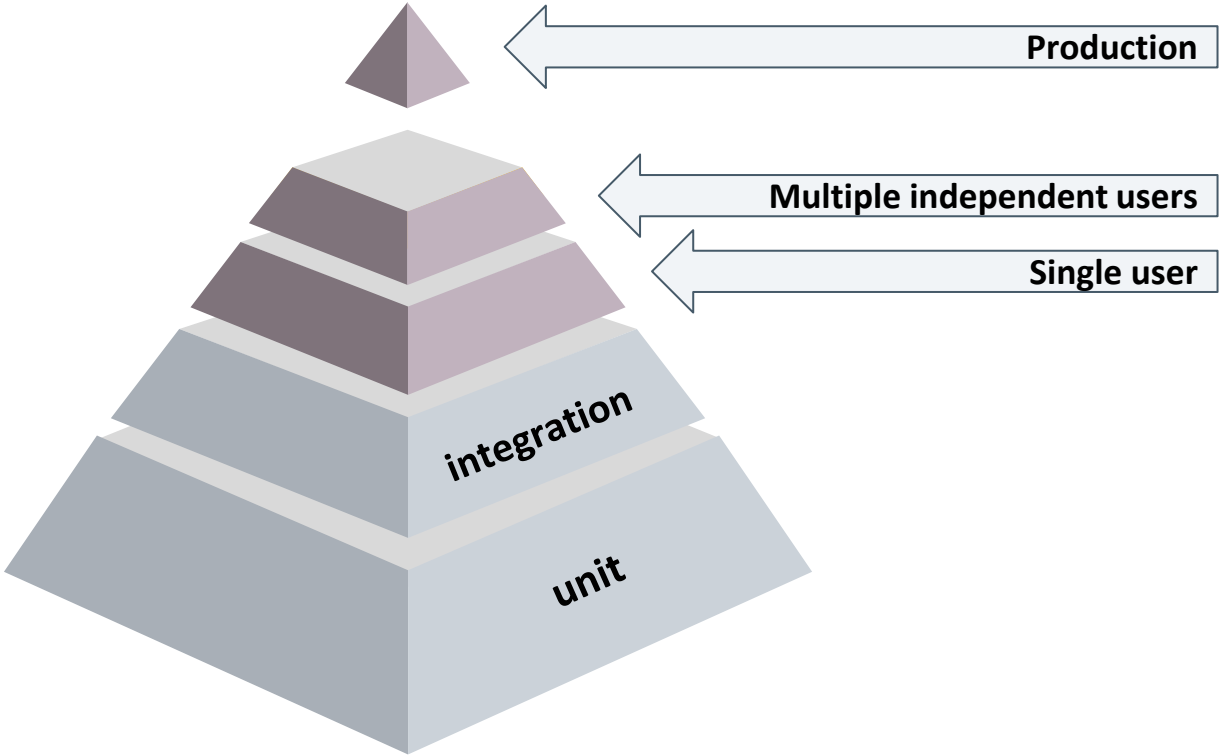
Exploring e2e in more detail



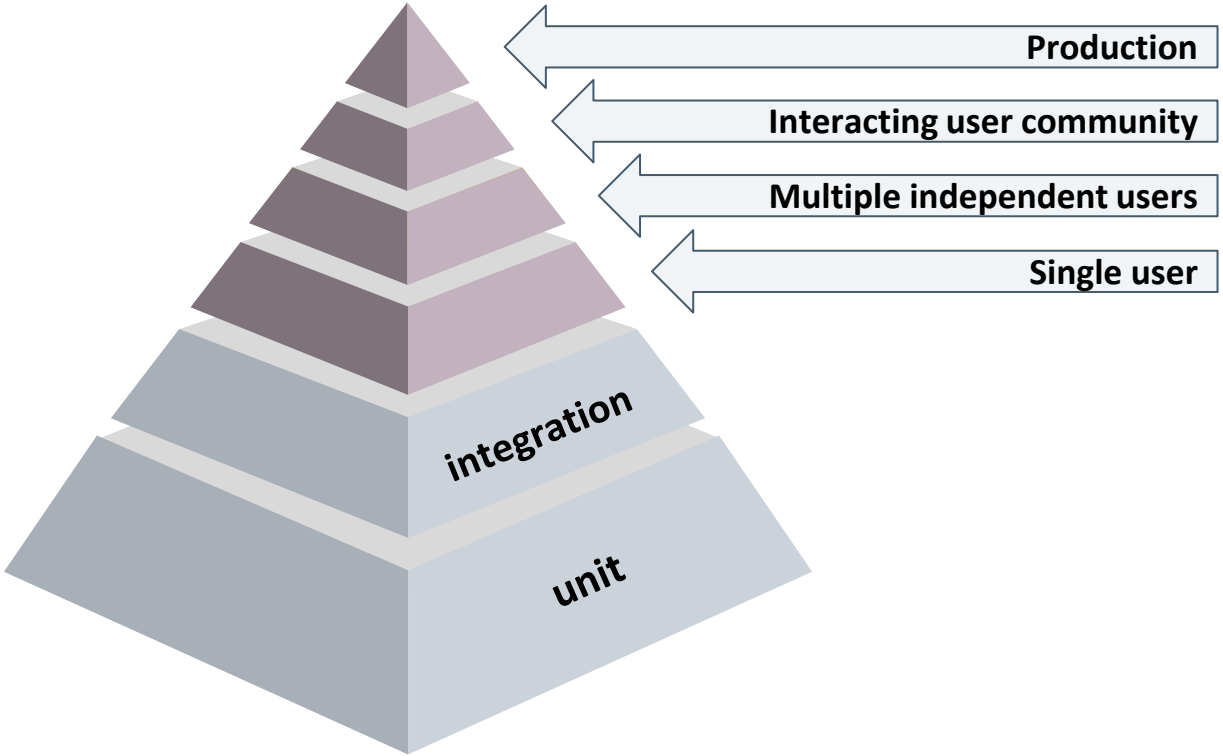
Exploring e2e in more detail



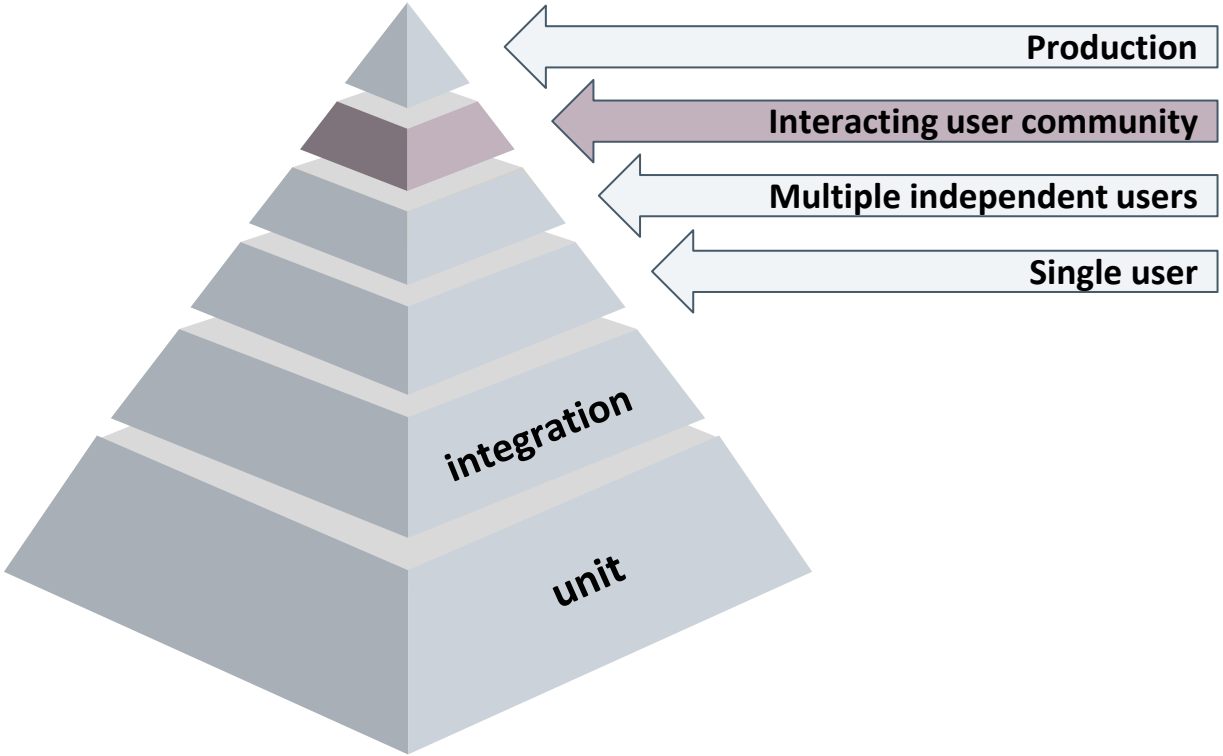
Exploring e2e in more detail



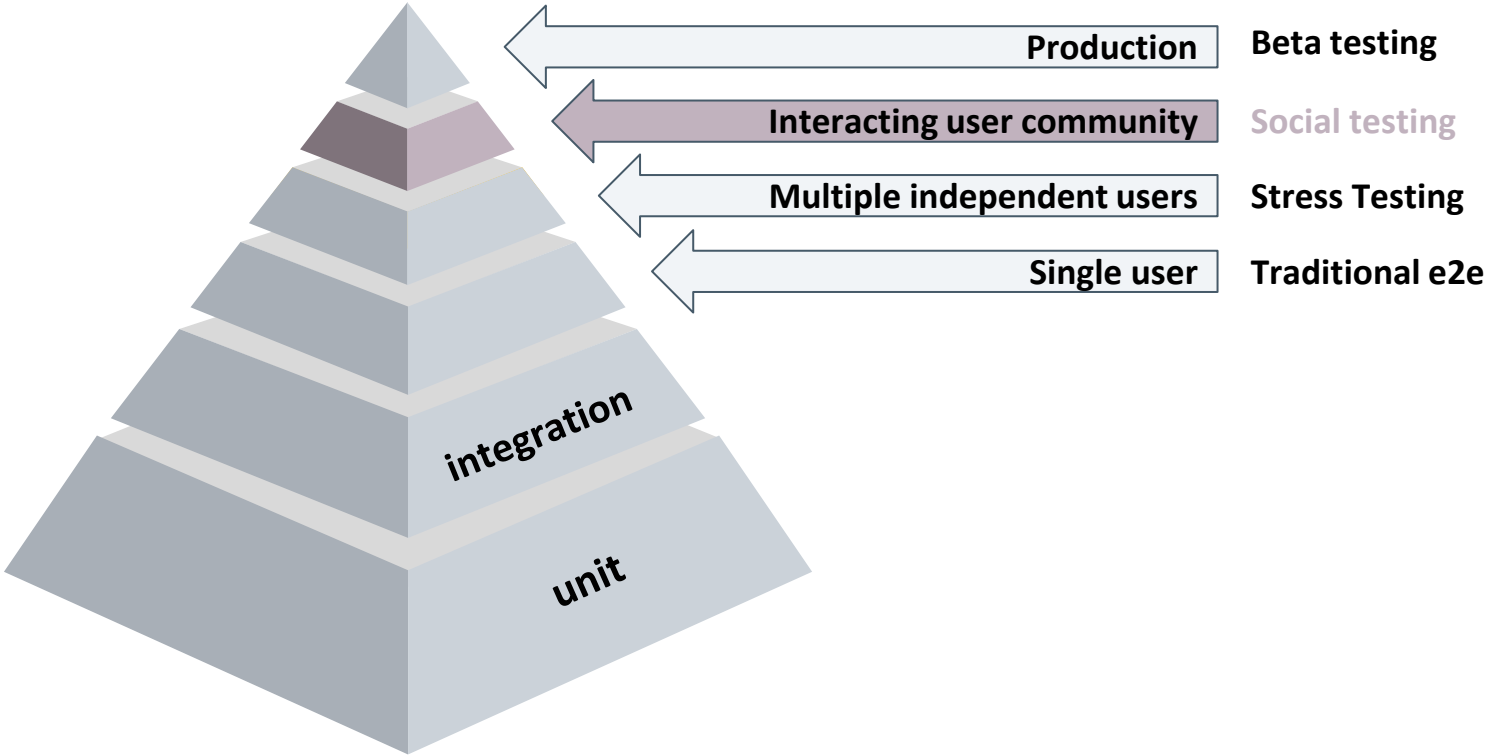
Exploring e2e in more detail



Exploring e2e in more detail: New social testing



Exploring e2e in more detail: New social testing



Simulation Based Testing at Meta timeline

2016 — Sapienz paper

2017 — Acquired and Founded Sapienz

2019 — WES project

2020 — WES Paper

2020 — Virtual Alpha

2021 — MIA

2022 — Simulation-based testing

sapienz

Find issues early in development process testing diffs and master builds

VIRTUAL ALPHA

Identify high impact issues by simulating production and prevent their release

Automated Test Generation: next … unit tests!

Forthcoming papers

… three come along at once

Published papers

Assured LLM-Based Software Engineering

arXiv and ICSE 2024 InteNSE workshop keynote

Automated Unit Test Improvement using Large Language Models at Meta

arXiv and submitted to FSE 2024 Industry track

Observation-based unit test generation at Meta

arXiv and submitted to FSE 2024 Industry track

Published papers

Assured LLM-Based Software Engineering

arXiv and ICSE 2024 InteNSE workshop keynote

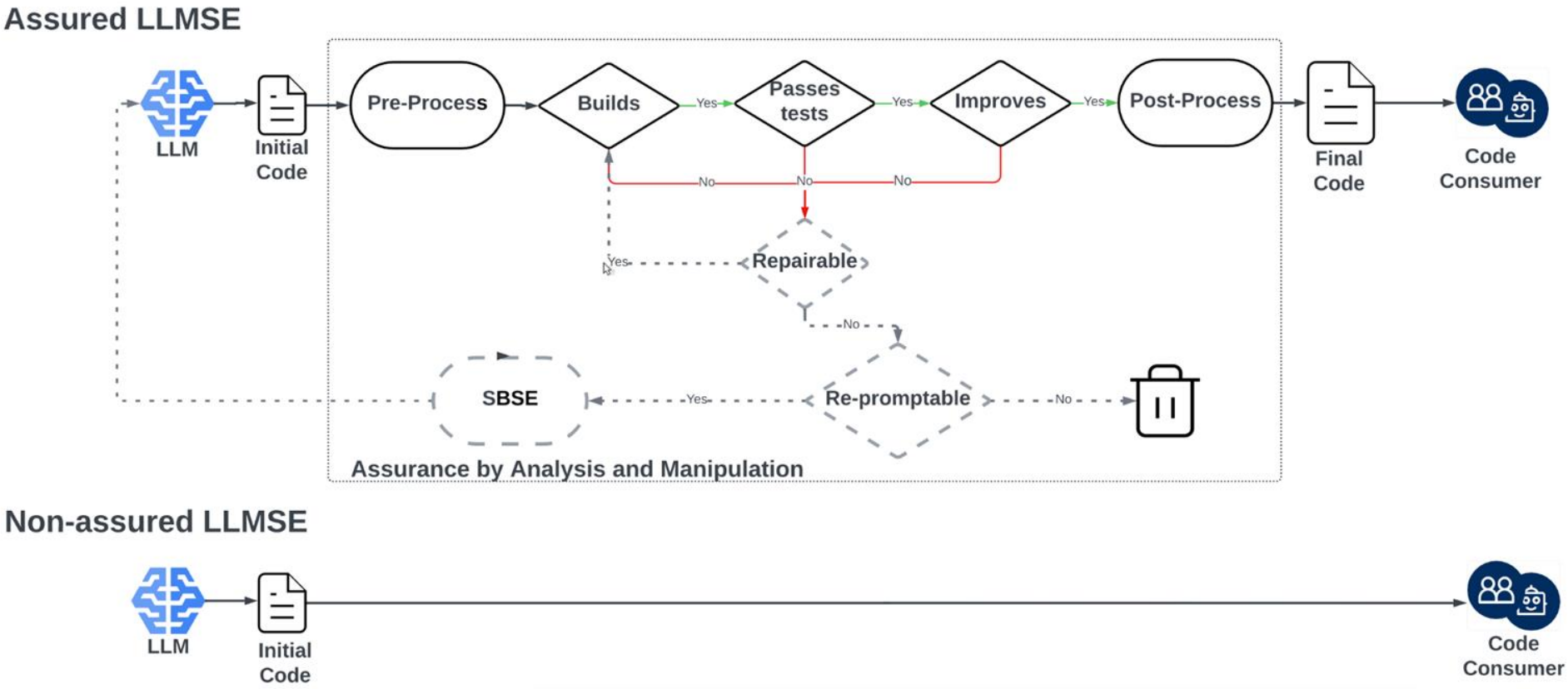
Automated Unit Test Improvement using Large Language Models at Meta

arXiv and submitted to FSE 2024 Industry track

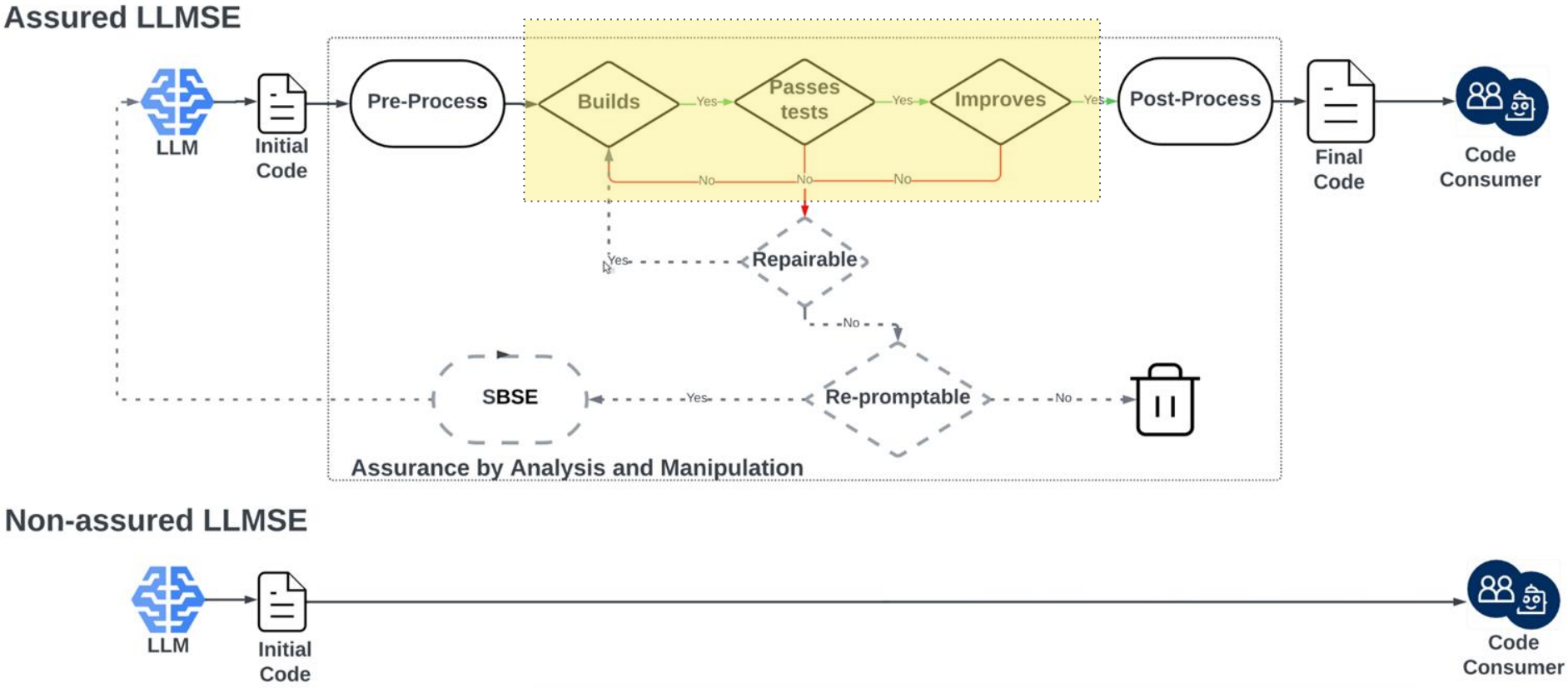
Observation-based unit test generation at Meta

arXiv and submitted to FSE 2024 Industry track

Assured LLM-Based Software Engineering: ICSE workshop keynote



Assured LLM-Based Software Engineering: ICSE workshop keynote



Filters are fitness functions too

Imagine a real valued “threshold” filter

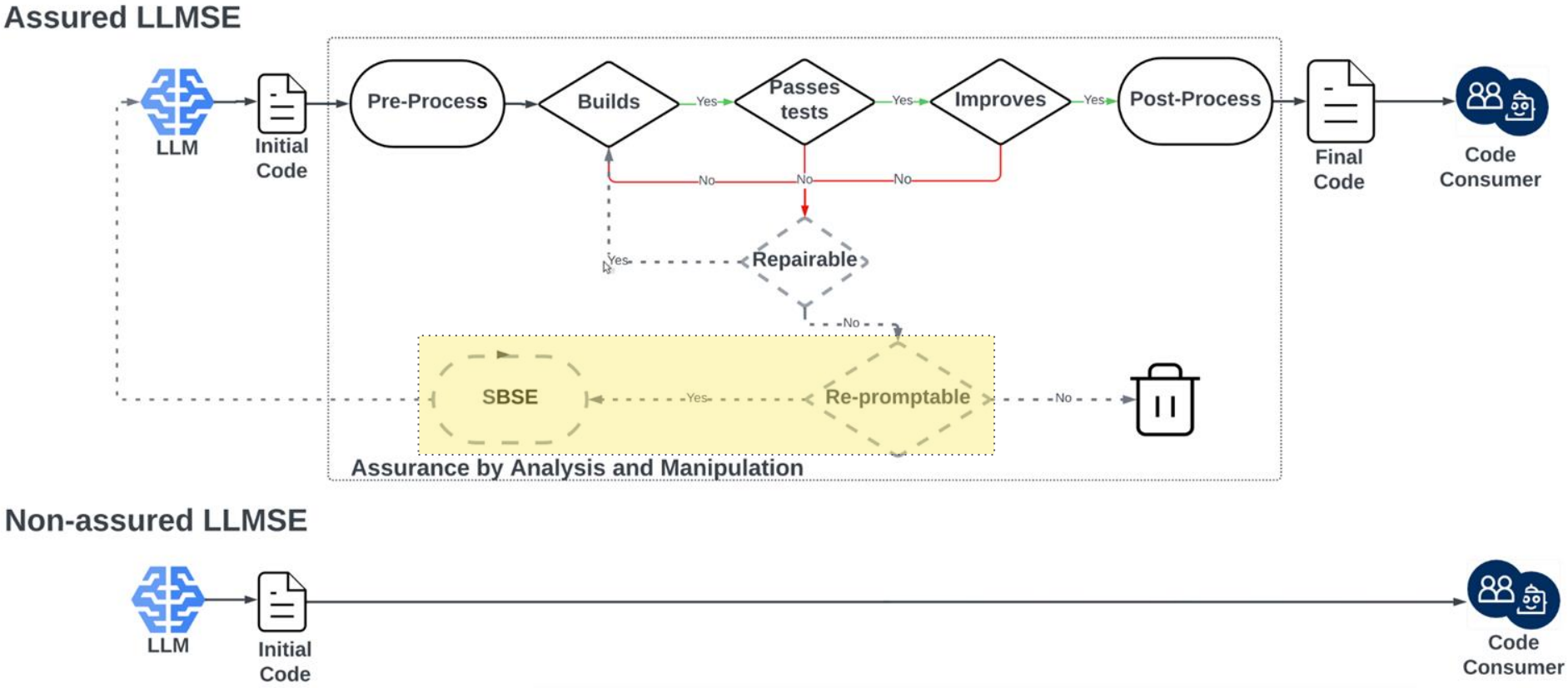
This can be a **fitness function**

Fitness functions are metrics

“Metrics are fitness functions too”: Harman et al., 10th International Symposium on Software Metrics, 2004. [ref]

Details in *Assured LLM-Based Software Engineering* paper

Assured LLM-Based Software Engineering: ICSE workshop keynote



DSL promoting language

Imagine a Turing-complete **prompting language**

- Chain of Thought (CoT) is simply **sequencing** alone
- Add **selection** and **repetition**

Use **SBSE** to optimise programs in this language

Now we have a **self-optimising prompting strategy**

In time, maybe also **use LLMs** to suggest programs in the language

The fitness functions are metrics that guide the whole process

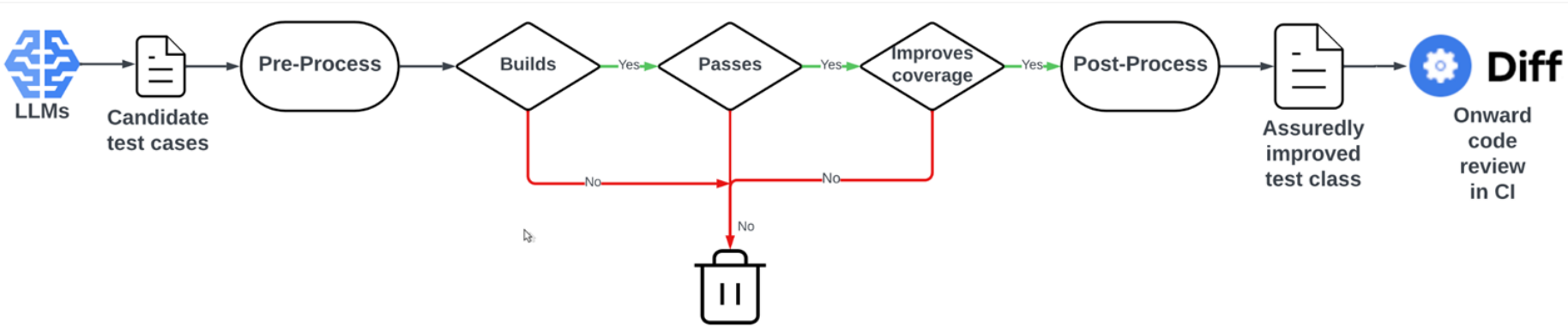
Like in Genetic Improvement [ref]

Assured LLMSE for Test class improvement

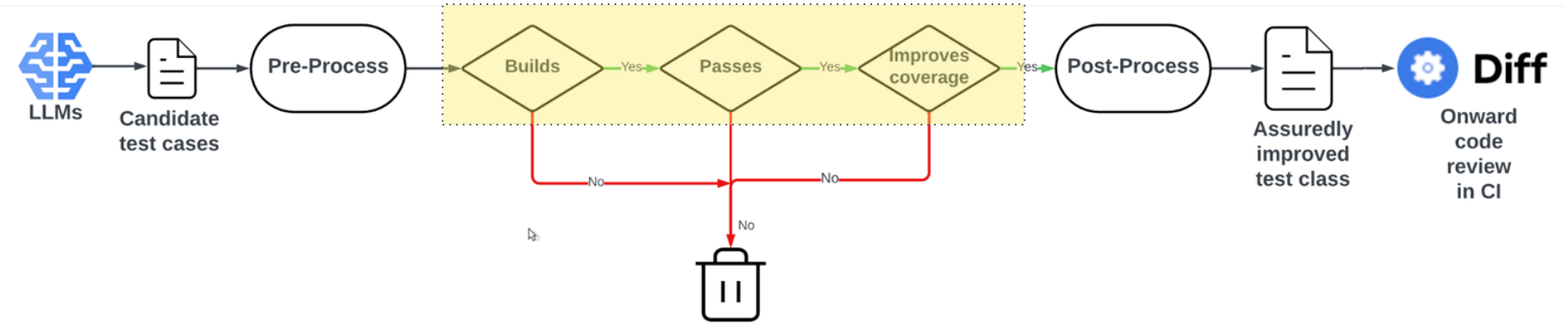
Our first steps towards general LLMSE

We considered the special case of test class improvement

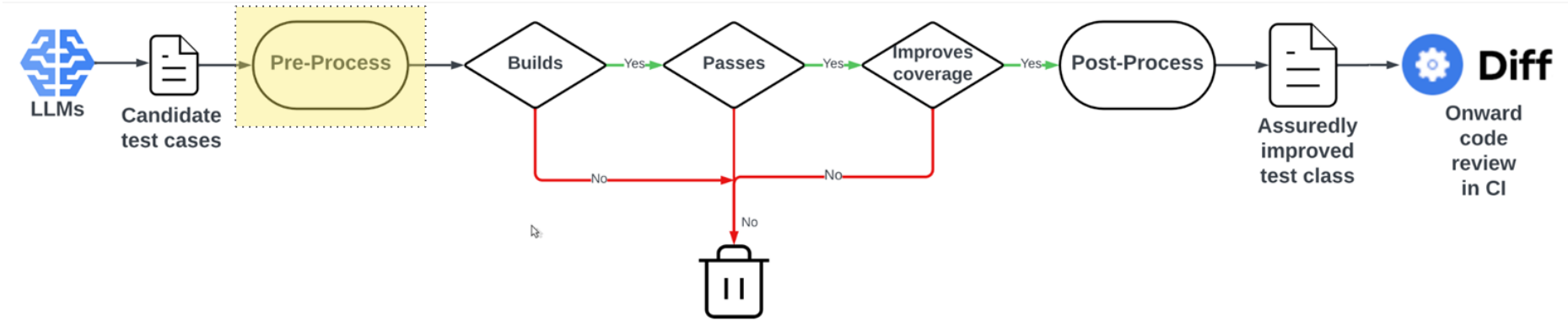
Automated Unit Test Improvement using Large Language Models at Meta



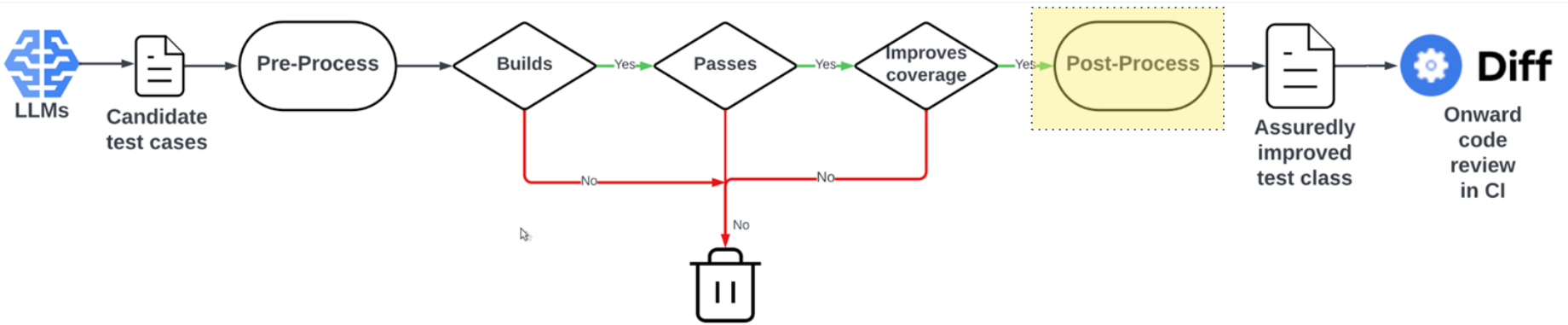
Automated Unit Test Improvement using Large Language Models at Meta



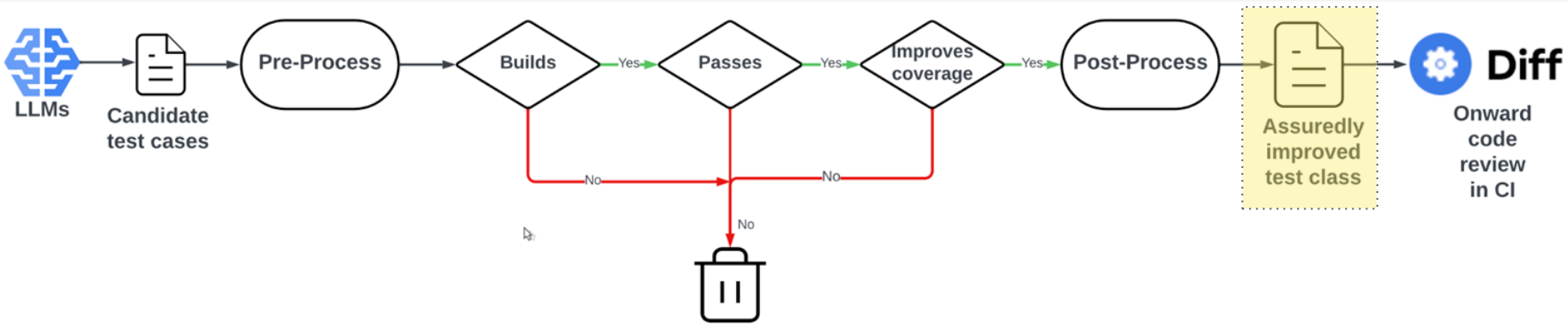
Automated Unit Test Improvement using Large Language Models at Meta



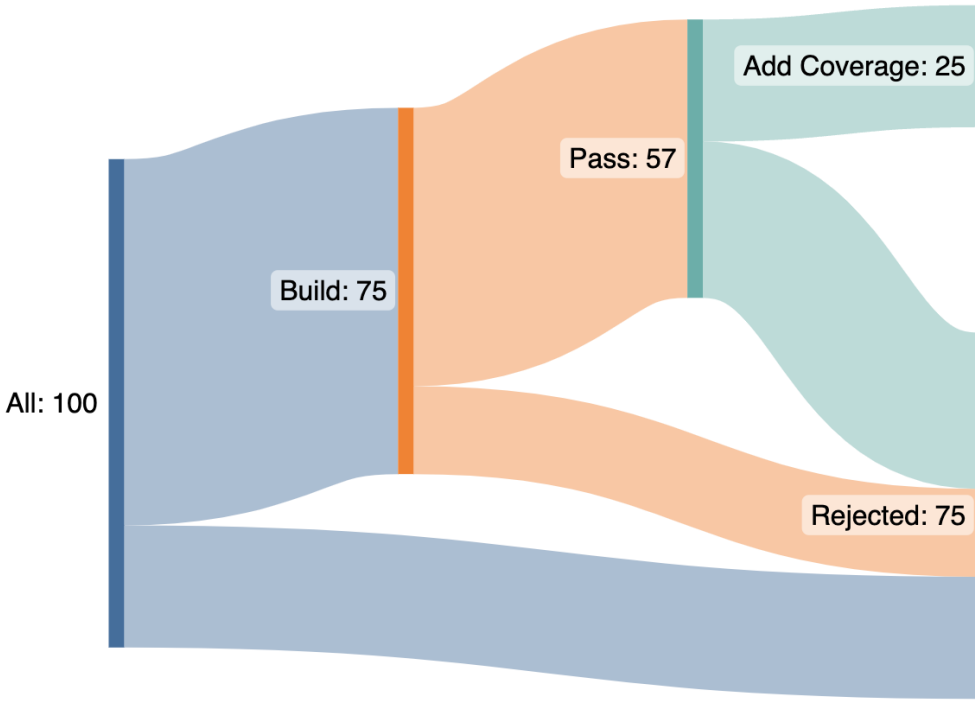
Automated Unit Test Improvement using Large Language Models at Meta



Automated Unit Test Improvement using Large Language Models at Meta



Automated Unit Test Improvement using Large Language Models at Meta: results



rank	test author	No, of tests	lines covered	diffs
1.	Threads Engineer	40	1,047	8
2.	Home Engineer	34	650	6
3.	Business Engineer	34	443	3
4.	Sharing Engineer	33	816	8
5.	Messaging Engineer	18	157	2
6.	**TestGen-LLM**	17	1,460	17
7.	Friend Engineer	12	143	2
8.	Home Engineer	10	273	2
9.	Creators Engineer	10	198	3
10.	Friends Engineer	10	196	5

Table 1: Results from the First Instagram Test-a-thon, Conducted in November 2023. The TestGen-LLM tool landed in sixth place overall, demonstrating its human competitive added value as a virtual member of the test improvement team during the test-a-thon.

Conclusions

Assured LLMSE is the new Genetic Improvement

LLMs are the new GP

So much exciting research to do

Next steps

Use high quality regression tests as the filter for LLMSE

Details in *Assured LLM-Based Software Engineering* paper