Testing Web Enabled Simulation at Scale Using Metamorphic Testing

John Ahlgren, Maria Eugenia Berezin, Kinga Bojarczuk, Elena Dulskyte, Inna Dvortsova, Johann George, Natalija Gucevska, Mark Harman, Maria Lomeli, Erik Meijer, Silvia Sapora, and Justin Spahr-Summers
FACEBOOK Inc.

***Abstract*—We report on Facebook's deployment of MIA (Metamorphic Interaction Automaton). MIA is used to test Facebook's Web Enabled Simulation, built on a web infrastructure of hundreds of millions of lines of code. MIA tackles the twin problems of test flakiness and the unknowable oracle problem. It uses metamorphic testing to automate continuous integration and regression test execution. MIA also plays the role of a test bot, automatically commenting on all relevant changes submitted for code review. It currently uses a suite of over 40 metamorphic test cases. Even at this extreme scale, a non–trivial metamorphic test suite subset yields outcomes within 20 minutes (sufficient for continuous integration and review processes). Furthermore, our offline mode simulation reduces test flakiness from approximately 50% (of all online tests) to 0% (offline). Metamorphic testing has been widely-studied for 22 years. This paper is the first reported deployment into an industrial continuous integration system.**

***Index Terms*—Metamorphic Testing, Oracle Problem, Scalability, Testing, Test Flakiness, Web-Enabled Simulation.**

I. Introduction

We describe Facebook's metamorphic testing system MIA, which is used to test WW, Facebook's simulation of its own platforms [1]. WW incorporates hundreds of millions of lines of code, simulating user interactions with bots trained by machine learning. WW is a multi agent simulation in which each agent is essentially a bot, isolated from production users, that simulates classes of user behaviour.

Not only is WW a simulation at scale, it is potentially *highly* non-deterministic. This forces us to tackle the issue of test flakiness [2], [3]. Furthermore, WW is a simulation of *unanticipated* human behaviours, thereby making the test oracle problem particularly pernicious; not only is the oracle unknown, it is inherently *unknowable*, making it the epitome of the most challenging of test oracle problems [4].

Simulation correctness is of profound importance, due to the far-reaching impact of simulation-based predictions. Such simulation–based predictions lie at the very heart of many of the critical decisions humanity must make; decisions that fundamentally impact all species on the planet. For example, simulation applications include economics [5], climate change and weather prediction [6], traffic safety [7], and the spread of diseases, where simulation determined governmental responses to the COVID-19 pandemic [8].

Since the decisions made from the results of these simulations are so fundamental, their predictions and the code on which they are based acquire paramount importance. The consequences of (perhaps subtle, yet pivotal) bugs are clearly among the most impactful facing any software engineer. In order to improve testing of simulation-based systems, more work is needed to address the problems of test flakiness and the oracle problem.

Based on metamorphic testing, we have been deploying an automated test infrastructure for our web-enabled simulation, WW, at Facebook, that tackles these twin problems of oracles and flakiness. In this paper we report on the development and deployment of our metamorphic testing system, MIA: Metamorphic Interaction Automaton. MIA is a system for end–to–end automated metamorphic testing, at scale. MIA was first deployed in 2020 as part of Facebook's continuous integration system. Developers working on WW primarily see its results in the form of an automated bot commenting on relevant code changes in code review at diff time (at Facebook a code change request is called a 'diff').

Tackling the Oracle Problem: The oracle problem is a well-known problem in software testing: we typically do not have an 'oracle' that determines the expected output for a given input [9]. Metamorphic testing [10], [11] is one widely-studied way to ameliorate the difficulties posed by the oracle problem. A metamorphic test relates two or more input and output pairs, thereby requiring only a partial specification [12], sufficient to capture the metamorphic relation. Metamorphic relations imbue testing with *some* degree of specification, when a *full* specification is unknown (or even, as in our case, unknowable).

There has been previous work on the application of metamorphic testing techniques to real-world software systems. For example, Zhou et al. [13] used metamorphic testing to test search engines such as Google and Yahoo, while Kuo et al. [14] reported on a case study application to an embedded system. Donaldson [15] reported that Google is also using metamorphic testing to uncover bugs in third graphics systems. However, although metamorphic testing has been a research topic of much interest [11] for over two decades [16], there have been few other previous industrial deployments reported in the literature and none that have been deployed into a continuous integration system. MIA is therefore the first deployment of Metamorphic Testing into a full–scale industrial continuous integration system.

Mark Harman is part supported by European Research Council Advanced Fellowship grant number 741278.

Tackling the Test Flakiness Problem: MIA also performs regression testing [17] for the WW system. We show that regression testing can be formulated as a special case of metamorphic testing. The task of regression testing a simulation system is challenging because of the test flakiness problem [2], [3]. Test cases tend to be highly flaky [1], due to the simulation resting on an infrastructure that contains, in abundance, all the previously identified technical features known to cause test flakiness [3], [18]–[20].

We developed an iterative capture–replay [17], and an offline mode to tackle flakiness. We then formulate the problem of regression testing as the testing of a metamorphic relation between different versions of the system using either capture–replay and/or offline mode. Iterative capture–replay avoids the test flakiness problem by continually updating the metamorphic relation to capture the previous version's outcomes and test against the next version. Offline mode helps us to tackle both the scalability and test flakiness problems.

In testing terms, capture–replay is an 'on-demand' deflakiness approach that can, in theory, tackle flakiness in any simulation, as it executes. By contrast, offline mode is an anticipatory de-flakiness approach, that determines, prior to simulation, the computation needed to be pre-computed to avoid flakiness.

The primary contributions of this paper as follows:

1) The first industrial application of metamorphic testing.
2) A new approach to testing simulation-based systems using metamorphic testing, introducing novel metamorphic relations for simulation testing.
3) Experience on running metamorphic testing and regression testing on a large scale system (hundreds of millions of lines of code), with open problems and research challenges identified based on this experience.

II. Web Enabled Simulation

A Web Enabled Simulation (WES) is a simulator of the behaviour of a community of users on a software platform. By contrast with more traditional simulation systems [21], [22], a WES can use a (typically web-enabled) software platform to simulate real-user interactions, thereby allowing it to simulate users' behaviour on the real platform infrastructure, isolated from production users [1]. A WES system is thus a multi agent system in which each agent is essentially a bot that simulates user behaviour. At Facebook we have built a WES called WW [1] that simulates the Facebook WWW backend by wrapping the entire real WWW platform infrastructure to create a WES for social interaction simulation.

A. Mechanism Design

Partly inspired by economic game theory [23] and our previous work on automated repair [24], we are building a transformation 'layer' we call 'Automated Mechanism Design' [1]. Mechanism design is a form of genetic improvement. Genetic improvement consists of automatically searching for code modifications that improve functional and non-functional behaviour of the system, guided by testing [25].

Mechanism design offers a scalable 'light touch' approach to genetic improvement, in which the mechanism through which bots interact with the underlying platform is transformed. These transformations simulate proposed software improvements, without the need to rebuild the entire backend system. Automation can be provided Search Based Software Engineering (SBSE) [26], [27]. Our approach to automated mechanism design was motivated by the need for scalability: given the scale of the backend systems and Facebook platform, build times can take several minutes [28], [29]. This build time issue would be a barrier to scalability were we to seek to perform genetic improvement directly on the code itself.

Fortunately, automated mechanism design circumvents the need to change the code directly, by building a mechanism layer between the bots and the underlying infrastructure. The mechanism can be tuned and reconfigured to simulate arbitrary possible changes to the code. We believe that this mechanism design approach may also benefit the many other applications of genetic improvement [25], where build times pose a challenge to scalability.

B. Bot Traversals of the Social Graph

A key component to WW is that bots traverse the social graph in order to simulate real users' journeys in their sessions with Facebook products. We use a combination of machine learning techniques, including imitation learning and rule-based descriptions of previously-witnessed behaviours in order to capture realistic traversals.

Due to page limit constraints, a detailed overview of the bot training process is omitted herein. It is sufficient to know that multiple bots traverse the WW platform simulating harmful and non-harmful behaviours. The focus of this paper is on testing the overall infrastructure and the mechanisms that constrain the way in which bots can interact with the platform. The testing approach is independent of the bot training method. Therefore, our metamorphic testing approach and regression testing techniques generalise to simulations that involve any kind of bot training, and any kind of constraints that restrict the way the bots can interact with an underlying platform.

C. Bot Personas

WW imbues a bot with a persona that captures a particular sub–population of the overall user population, based on coarse-grained statistical demographic information. For example, age range and geography are both captured by personas. For bots that simulate harmful behaviours, a predilection towards the particular kind of behaviour is also part of the persona. For each specific persona, we sample from the set of all possible bots that share this persona to produce a population of bots, each of which implements an instance of the persona. In this way, we can perform simulations on classes of users that exhibit particular kinds of behaviour, using the associated persona, although we do *not* simulate *any* specific user. In software testing terms, a persona can be thought of as part of the initial state of the test case.

The behaviour of different bots with different personas is expected to exhibit certain characteristics and these can be captured in metamorphic tests. That is, we cannot determine *exactly* how a particular bot will behave, because its behaviour is inherently unknowable (this is why we need simulation after all). Nevertheless, we *can* capture salient properties of particular personas and the relationships between behaviours of bots with different categories of persona. These behavioural differences are relationships on which to base metamorphic relations, and thereby facilitate metamorphic testing.

III. Metamorphic Testing

For a System Under Test (SUT) p that implements the function f, a *metamorphic relation* is a relation over applications of f that we expect to hold across *multiple* executions of p. For simple mathematical functions f, there are a number of obvious (readily definable) metamorphic relations. For instance, suppose $f(x) = e^x$, then we know that $e^a e^{-a} = 1$ ought to hold, and this thereby defines a metamorphic relation that ought to be respected by any SUT that implements f [30].

Metamorphic testing is the process of defining and applying metamorphic relations to generate partial test oracles that apply between two (or more) executions of a SUT [31]. We implemented metamorphic testing in terms of the sequence of test observations with a reliable reset. This is easily defined using the terminology and notation from the recent survey on the Test Oracle Problem [32]. In that survey, testing was couched in terms of 'stimuli' and 'observations', denoted $\overline{x}$ when x is a member of the set of observations (such as outputs) and $\underline{x}$ when x is a member of the set of stimuli (such as inputs).

Formally, this means that a metamorphic relation is defined [32] as follows: There is a reliable reset, which is a stimulus, $\underline{R}$, that allows us to interpose a reset between a previous test case, $\langle \underline{x_1}, \overline{y_1} \rangle$, and a subsequent test case, $\langle \underline{x_2}, \overline{y_2} \rangle$, to form a five-element test sequence $\langle \underline{x_1}, \overline{y_1}, \underline{R}, \underline{x_2}, \overline{y_2} \rangle$, for which we can define a relationship, the "metamorphic testing relationship", π, between x_1, y_1, x_2 and y_2.

According to the formulation of metamorphic testing in the current literature π is a 4-ary predicate, relating the first input-output pair to the second and our metamorphic oracle, $\mathbb{D}$ consists of running a sequence of test executions with input stimuli, observing the output, and then applying a reliable reset between the two test cases. The reliable reset simply initialises the simulation to an initial state, thereby ensuring that both test executions that participate in the metamorphic relation are executed in the same initial state. The metamorphic oracle $\mathbb{D}$ is defined as follows:

$$\mathbb{D}\langle \underline{x_1}, \overline{y_1}, \underline{R}, \underline{x_2}, \overline{y_2} \rangle \text{ if } \pi(x_1, y_1, x_2, y_2)$$

MIA allows relationships between arbitrary numbers of runs, so we generalise this to a test sequence $\langle \underline{x_1}, \overline{y_1}, \underline{R}, \ldots, \underline{R}, \underline{x_n}, \overline{y_n} \rangle$ for $n \geq 1$, where the metamorphic oracle, $\mathbb{D}$, is defined as follows:

$$\mathbb{D}\langle \underline{x_1}, \overline{y_1}, \underline{R}, \ldots, \underline{R}, \underline{x_n}, \overline{y_n} \rangle \text{ if } \pi(x_1, y_1, \ldots, x_n, y_n) \\ n \geq 1$$

When $n = 1$ this metamorphic oracle degenerates to the traditional (end-to-end) testing oracle in which the predicate π is a predicate on a single input–output pair (a test case). For this degenerate case, MIA falls back on the implicit oracle [9], that the test input $\underline{x_1}$ should not 'break' execution; if the test causes a crash then it is deemed to fail, otherwise it passes, so $\pi = \lambda x, y.y \neq \mathtt{crash}$.

For traditional metamorphic testing $n = 2$. For regression testing, $n = 2$, just as for traditional metamorphic testing, but there are further constraints: $\overline{y_1}$ is the result of executing a previous version of the system on $\underline{x_1}$ (in offline mode), while $\overline{y_2}$ is the result of executing the latest version of the system on $\underline{x_2}$ (also in offline mode), where $x_1 = x_2$ and $\pi = \lambda x, y.x = y$. As this formalisation reveals, for MIA, regression testing is simply as special (constrained) instance of traditional metamorphic testing. This connection between metamorphic testing and regression testing was first observed (although not exploited further) in the technical report version [32] of the subsequently peer-reviewed [9] survey on the Test Oracle Problem. MIA also supports more general metamorphic testing, where $n > 2$.

A. Test Order Dependence and Asynchronous IO

Asynchronous IO can help scale testing, particularly where (as in our case) the overall computation is heavily reliant on IO. However, we need to tackle the test order dependence problem [17], [33]. We can only asynchronously execute those tests that are independent and some are inherently order dependent. For example, suppose we wish to observe a particular bot trajectory for a given input, and then select some arbitrary point in that trajectory and use this as the initial state for a subsequent follow-up run. MIA automatically detects test order dependence and runs order dependent tests sequentially, while order independent tests are run asynchronously, thereby contributing to the scalability of the overall approach.

B. Statistical Metamorphic Testing

In our work, reported here, we have used a limited form of statistical metamorphic relation. We have chosen relations where stochastic simulation properties do not impinge on our ability to test the metamorphic relation. For example, whenever we test a metamorphic relationship between different bot personas, we choose situations where the statistical effect size is sufficiently large that it is highly likely to manifest in *any* pair of executions.

More subtle stochastic properties may require a more sophisticated statistical analysis. For example, many mechanisms that denote rate limits; limits on the rate at which some particular actions or observations can be performed. Rate limits tend to have an overall monotonic effect: the more we tighten the rate limit, the more the behaviour becomes constrained, with the result that observable properties tend to rise or fall *roughly* monotonically with the increasing rate limit. The problem is that this monotonicity property is, indeed, a "rough trend" rather than a guaranteed behaviour.

It is known that such statistical metamorphic relations pose additional challenges [34], [35]. There are approaches reported in the literature on statistical metamorphic testing to handle such cases. For example, Guderlei and Mayer [36] introduced statistical metamorphic testing, which has been applied to metamorphic testing to stochastic optimisation algorithms [37]. Nevertheless, the statistical analyses required to augment traditional metamorphic testing with the ability to detect small yet important effect sizes remain to be fully developed.

IV. The MIA End to End Metamorphic Testing System

WW has three simulation modes (online, offline and synthetic), while MIA has two deployment modes (continuous and per-diff).

A. *WW Simulation Modes and Handling Test Flakiness*

We uses three simulation modes:

Online Mode: This is the most computationally expensive, but also arguably the most realistic simulation; it uses the real infrastructure and graph, although bots traverse this graph in read only mode [1] and therefore cannot interact.

Offline Mode: This is less computationally expensive because a near strongly connected component sub–graph is extracted and pre–loaded (incurring a one–off computational cost).

Synthetic Graph Mode: This mode does not use a real social graph, but replaces it with a graph generated to share some salient statistical properties with the real social graph.

For online mode we can also capture–replay traversals. Each mode has different applications and use cases. In this paper we are primarily concerned with the way in which the modes assist testing. In particular, capture–replay and offline modes tackle test flakiness. That is, most nontrivial web–based systems are also highly non-deterministic. Non-determinism leads to test flakiness. Non-determinism is particularly prevalent in web enabled systems, so WW denotes the archetype of software testing flakiness challenge.

The problem of test flakiness has been widely reported in the literature [2], [3], [38]. It is one of the most challenging problems for production testing and verification systems [2], [39].

We have found three causes of test flakiness, each of which is tackled by the MIA as follows:

Real World State Change: WES systems model the real world, which is subject to constant state change. These changes are reflected in WES system state. As it has previously been observed [40], real world state change means that the test coverage can be affected by almost *anything*, including the prevailing weather conditions. In the case of WW, the Facebook social graph continually changes, as over 2 billion users interact with it, updating state as they do so. Fortunately, we can use offline execution to control for any consequent flakiness. Offline mode uses a pre–computed subgraph; a snapshot of the real world at a point in time and therefore it is immune to state changes.

Inherent Infrastructure Non-Determinism: The backend Facebook system optimisation involves all of the software engineering features known to engender test flakiness, such as asynchronous IO, context sensitive optimisation, and order dependencies [3]. However, using the capture–replay approach (online) and/or offline mode, we not only control flakiness due to changes in the real world state (e.g. users' updates to the social graph), but also the inherent non-determinism of the infrastructure that supports social media interactions.

Simulation Sampling: Differences in bot behaviour are also possible due to different bot personas. We therefore require that the bot persona is determined entirely by the simulation run's pseudo random number generation seed. Essentially, the seed denotes an element of a sample of all possible bot behaviours simulated in WW. With iterative capture–replay and offline mode, it is ensured that all the variance in bot behaviours is due to the bot's persona, which is, in turn, defined solely by the choice of random seed. This approach facilitates simple statistical random sampling.

B. *MIA Deployment Modes*

MIA is deployed in two modes, as described in this section.

Continuous Mode: Figure 1 shows an example of dashboard monitoring of the continuous deployment of metamorphic and regression testing. The dashboard also incorporates automated alerts, which fire when any of the existing test cases break in production, which alerts the team member who is 'on-call' for the week. This feeds into a standard Facebook DevOps process, part of which requires that each team member takes a one-week duty as the on-call member of the team for the week.

We also use a 'Chaos Monkey' style approach [41] to simulate infrastructural failures, in order to test MIA's resilience to unexpected and expected infrastructural failures. Expected infrastructure failure can occur, for example, because of test timeouts, particularly with online mode, where it is challenging to predict test execution time. During periods of high demand on the overall elastic computing infrastructure, there can also be test failures due to lack of availability of virtual machines on which to perform the tests. In such cases, MIA uses standard engineering techniques such as retries and ultimately marks the test as broken so that this signal is not confused with failure. The Chaos Monkey approach also allows us to test this infrastructural resilience itself.

As can be seen from Figure 1 there is a cyclic system demand pattern, with reduced activity at weekends, which is a phenomenon observable throughout the company and reflects a commitment to work-life balance [42]. The company culture requires engineers to be accountable for breakages, hence the on-call process. However, balanced against this, is a strong commitment to work life balance, and therefore it is not expected that engineers will typically be working at weekends, nor after hours. Where possible, the company strives to deploy Software Engineering processes that avoid unplanned overtime and the associated harmful effects (on morale and productivity) this can engender in the engineering culture [43].

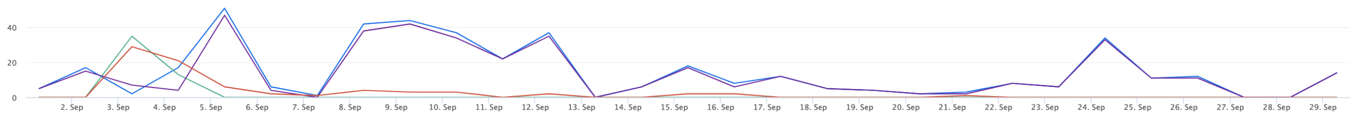
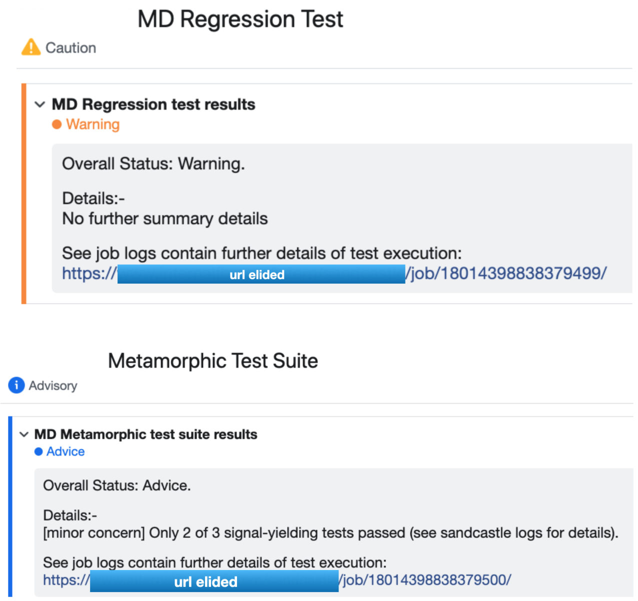


Fig. 1. Continuous test monitoring; part of the MIA Dashboard. This dashboard shows the executions of tests in September 2020. The upper two lines show successful runs for regression and metamorphic tests, while the (generally) lower two lines show the tests that failed to execute. The periods of minimal activity correspond to weekends.


MD Regression Test

⚠ Caution

MD Regression test results

● Warning

Overall Status: Warning.

Details:-
No further summary details

See job logs contain further details of test execution:
https://url elided/job/18014398838379499/

Metamorphic Test Suite

ⓘ Advisory

MD Metamorphic test suite results

● Advice

Overall Status: Advice.

Details:-
[minor concern] Only 2 of 3 signal-yielding tests passed (see sandcastle logs for details).

See job logs contain further details of test execution:
https://url elided/job/18014398838379500/

Fig. 2. Example of submit-time comments by the MIA bot in the Continuous Integration System. MD stands for 'Mechanism Design'. These are faithful screenshots of the signal given to engineers by the MIA bot, that have been amended only to occlude confidential proprietary information.

Per–Diff Mode: Figure 2 shows an example of the comments that MIA makes on all relevant diffs submitted to the code review system at Facebook. Each time a developer submits a diff for code review into the continuous integration system, the MIA system is automatically invoked to execute the changed code, and produces a test outcome signal to the developer (and human reviewer) in the form of regression tests and diff time metamorphic tests.

There is a standard "signal box" that can be incorporated in the continuous integration system to give such feedback. Using the signal box, the MIA system comments appear as part of the review process. Effectively, MIA acts as an automated bot commenting on diffs with test signal. The regression test signal appears in one signal box, while the metamorphic test outcomes appear in another box. As can be seen from Figure 2, the initial signal is a summary, but more information could be found by following up test logs. The entire integration with continuous deployment uses a simple web interface to allow developers to navigate to get outcomes and test signals provided.

V. The Metamorphic Test Suite

There are techniques in the literature for automated discovery of mathematical [31], [44], [45] and combinatorial [34], [35], [37], [46] metamorphic relations. However, the current state of the art is only, at best, partially automated. In the research literature, and in practice, since no metamorphic testing system has hitherto been deployed in industry, we use a manual definition of metamorphic relations for this first industrial deployment of metamorphic testing at scale.

For simulation systems, the interplay between different modes of simulation, and the interplay between these simulation modes and the application of mechanisms (through mechanism design) has led to a rich space of natural metamorphic relations. Space does not permit us to present the full test suite in detail. At the time of writing we have over 40 metamorphic tests, and the number is currently growing at a rate of between two and three per week.

Table I provides a top-level description of a key subset of MIA's current metamorphic test suite. This subset serves as an illustration of the rich diversity of possibilities, and also a basis for empirical results on flakiness. Section V-A, below, provides detailed descriptions of each. We continue to develop metamorphic tests and also end-to-end tests, which also use MIA, thereby practically exploiting the theoretical observation (from Section III) that an end-to-end test is merely a special case of metamorphic test where $N = 1$.

A. Illustrative Examples of Metamorphic Tests

Basic Replicability Online: Some of our metamorphic testing scenarios rely on replicability (non–flakiness). Therefore, we have a metamorphic relation to check this basic replicability. That is, we make two executions to create two traversals, each starting from the same starting point in the social graph and check that they do, indeed, return identical results. This is a simple metamorphic relation, since it is simply testing the identity relation.

Basic Replicability Offline: Offline mode traversals should be inherently replicable, since they are immune to social graph state changes between runs. We test this by starting two runs at identical initial states (randomly chosen starting states in the offline graph) and testing for traversal equality.

Feature Hiding: One of the ways in which harmful behaviours can be tackled is by hiding selected features from users who are engaging in harmful behaviour. There is a monotonic property that the more features are hidden, the less we would expect it to be possible to engage in harmful behaviour, and this is tested by feature hiding metamorphic tests.

Test Name	Metamorphic Principle tested
Basic replicability online	Replay should replicate a previously captured online run
Basic replicability offline	Offline mode should replicate all traversals
Feature hiding	Hiding features should tend to reduce harmful behaviour
Minimal mechanism equivalence	With a minimal mechanism (1 friend viewable only), bad actor bots are indistinguishable from random bots
Demographic homophilia	Bots will have a tendency to favour similar demographics in their graph traversal
Rate limit transitivity	Traversals respect rate limit transitivity. A detailed description is provided in Section V-A.
Personification	Bots with given personas exhibit lower variance than random bots
Randomisation	Random bot behaviour should be independent of whether they simulate bad actors or not

TABLE I
A Key Subset of the Facebook Metamorphic Test Suite

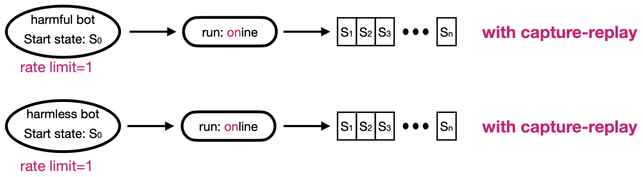


Fig. 3. The Essence of the Minimal Mechanism Metamorphic Relation. Capture replay essentially computes sufficient transient state information to render the two online mode executions equivalent to offline mode executions.

Minimal Mechanism Equivalence: Interaction faults are known to be subtle and hard to detect due to the number of interactions required [47], [48]. Fortunately, metamorphic relations can express equalities and inequalities between runs with different feature settings, thereby capturing that part of the oracle that relates the different desired/undesired feature interactions. In our case, with Web Enabled Simulation and Mechanism Design, there are many different mechanisms that rate limit the bots' behaviour. There are expected differential behaviours for bots subject to these limits. We capture these in metamorphic relations. For example, when we minimise the mechanism, by maximising the rate limit that applies to bot behaviour, this constrains the behaviour sufficiently strongly to create metamorphic identities.

One such identity applies when the maximum number of friends observable is mechanism–constrained to 1. The bots' behaviour is thus so–constrained that they *should* make the same choices, irrespective of the machine learner used to train them. Therefore, we start traversals from the same starting state with a minimal mechanism, and check that bots with different personas and machine learning decision-makers, produce identical trajectories because they are thereby constrained to follow the same traversal. The Minimal Mechanism metamorphic relation is depicted in Figure 3.

Demographic Homophilia: Since before the founding of Facebook it has been known that social networks exhibit homophily [49]: individuals are more likely to associate with other similar individuals. The Facebook social graph exhibits this property [50]. Therefore, when WW bots are behaving correctly and realistically, they should also exhibit this property; a bot with a given persona should be more likely to visit others with similar personas. This suggests a natural metamorphic test. We create two bots with different personas, and compare their traversals, measuring the difference in personas visited, to check that they exhibit homophily.

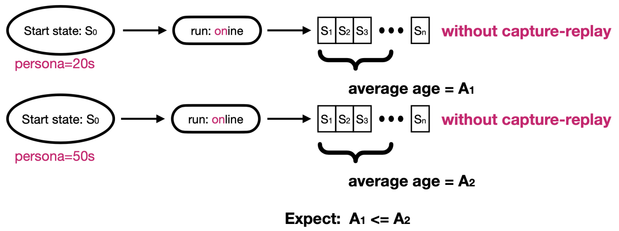


Fig. 4. The Essence of the Demographic Homophilia Metamorphic Relation. Note that this test does not require offline more nor capture replay and that it relies on an *expected* outcome; one that may *not always* hold in every case (but is expected to hold in most).

Such a metamorphic test would be useful in any system that associates users. The Demographic Homophilia metamorphic relation is depicted in Figure 4. In this case, the test focuses on the homophilic nature of the age demographic.

Rate Limit Transitivity: Mechanism rate limits also have interesting transitive effects on decision-making. For example, suppose we start a traversal from a state T, with a rate limit R_1. Then, we start two other traversals from the state T with rate limits R_2 and R_3, such that R_2 is more permissive than R_1 (e.g. a superset of friends are visible through R_2 compared to R_1) and such that R_3 is more permissive still than R_2.

The trajectory induced by execution with the intermediately permissive rate limit (R_2) must start with the common prefix of the trajectories induced by the least and most permissive rate limits, R_3 and R_1 respectively. This is satisfied (trivially) if there is *no* common prefix between the least and most permissive rate limits. However, should there exist any non-empty common prefix, p, between trajectories resulting from the least and most permissive rate limits, then the trajectory induced by the intermediately permissive rate limit must start with p.

Provided that execution is not stochastic for such rate limits, this transitive property applies to rate limits that have this 'subset' effect on permissiveness; the tighter the rate limit, the smaller the subset of possibilities they offer to the bot to choose from. Fortunately, although overall simulations are generally stochastic, their stochastic behaviours are typically controlled by a random seed, so that execution for a given seed is deterministic. Indeed, this is one of the desirable properties needed support scientific replication of simulation results (see, for example, recent verification reports for the replicability of nation–critical simulation results on COVID-19 [51]).

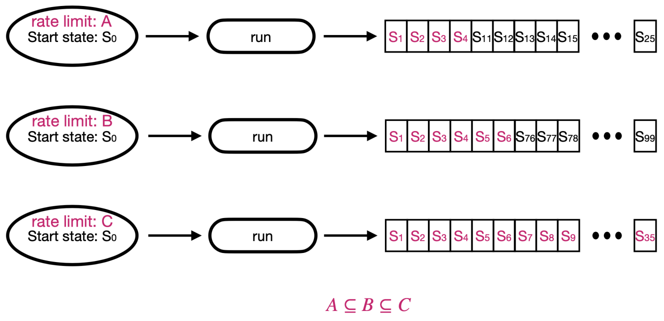

rate limit: A
Start state: S0
run
S1 S2 S3 S4 S11 S12 S13 S14 S15 ••• S25
rate limit: B
Start state: S0
run
S1 S2 S3 S4 S5 S6 S76 S77 S78 ••• S99
rate limit: C
Start state: S0
run
S1 S2 S3 S4 S5 S6 S7 S8 S9 ••• S35
A ⊆ B ⊆ C

Fig. 5. The Essence of the Rate Limit Transitivity Metamorphic Relation.

This test is useful at bug revelation because it connects several different rate limits and the decision making of the bots (which can be arbitrarily complex) in a simple transitive relation. Many of our rate limits are parameterised by a value that produces precisely this subset effect. This kind of metamorphic test principle is therefore very useful. Furthermore, it seems likely that many other forms of simulation system would have properties that would tend to constrain the simulation. Where these are similarly parameterised, this same metamorphic relation principle will apply.

The Rate Limit Transitivity metamorphic relation is depicted in Figure 5.

Personification: Bots with given personas must respect the general statistical properties of real users who share persona similarities. We never model any individual user, only coarse-grained statistical properties at the persona level. Nevertheless, we can still use metamorphic testing to capture one salient property of a persona: it should, in some high-level sense, *constrain* the bots' behaviour. Specifically, when we compare traversals made by a bot with a persona to those made by a bot without a persona, the variance in types of persona (e.g their age demographics) visited should be lower for the with–persona bot.

Randomisation: Although conceptually simple, effective randomisation is at the very heart of any simulation system. We have several metamorphic tests that check for properties of random and pseudo random behaviour to ensure that randomisation is less vulnerable to subtle bugs that might influence the outcome of simulations. For example, bots behaving entirely randomly should be independent of the personas.

VI. Quantitative Experience on Execution Time and Test Flakiness

This section contains quantitative results on test execution times and test flakiness, providing evidence for the scalability of our approach and the way in which offline mode tackles flakiness.

A. *Test Suite Execution Time*

Figure 6 shows the execution time required by the regression and metamorphic test suites run at diff time. As can be seen the peak execution time over the month of September 2020 did not rise above 20 minutes.

This is acceptable for a periodic execution that runs every 6 hours in continuous deployment. It is also acceptable for diff time testing, because it is unlikely that a human reviewer will comment in under one hour [20]. Key to this scalable deployment was the use of offline mode (which executes an order of magnitude faster than online mode) and the use of asynchronous IO to concurrently execute test cases where there is no test order dependence.

B. *Flakiness*

Table II presents results for test flakiness. Each row of the table corresponds to a deployment of a metamorphic test, in either online or offline mode, and identifies the bot personas tested. Each test is executed three times sequentially. The outcome is deemed to be flaky if the test outcome differs on any of the three executions. In order to collect this data, the test suite was executed every hour over a period of three days and nine hours. The rows are ordered by decreasing flakiness.

The most obvious finding from Table II is that all of the tests that exhibit any non-zero flakiness do so when executed in online mode. By contrast, offline mode is far less flaky exhibiting, in this experiment, *zero* flakiness. Interestingly, the persona variance test is not flaky in either online or offline mode, highlighting the fact that persona variance is a statistical property. However, the test has revealed an important (and unexpected) difference between online and offline mode[1]

This difference highlights another value of metamorphic testing. Offline mode is an order of magnitude faster and reduces flakiness considerably, as these results show. Nevertheless, it remains challenging to capture *all* of the offline information and pre-compute it ahead of time in order to guarantee to replicate all of the statistical properties witnessed in online mode.

There is also, of course, no guarantee that Offline mode captures all of the bugs that can occur in online mode; it's more likely that it captures a subset. We therefore need to remain constantly vigilant for differences in online and offline mode, with online mode playing the role of ultimate 'litmus test', whereupon, flakiness reappears as an issue with which we have to contend.

Furthermore, it is interesting to observe that some tests can exhibit zero flakiness while being executed online. Consider the demographic homophilia test, which checks the behaviour of bots with different age demographics. This metamorphic test concerns a statistical property of the overall behaviour of the bot, related to the demographics. As such, the test is not strongly affected by state changes in the real world graph, since these state changes also respect this demographic property. This is an interesting finding, because it points to the possibility of also using statistical properties to overcome flakiness (even in the inherently non–deterministic online execution mode).

[1] This unexpected difference is under investigation at the time of writing.

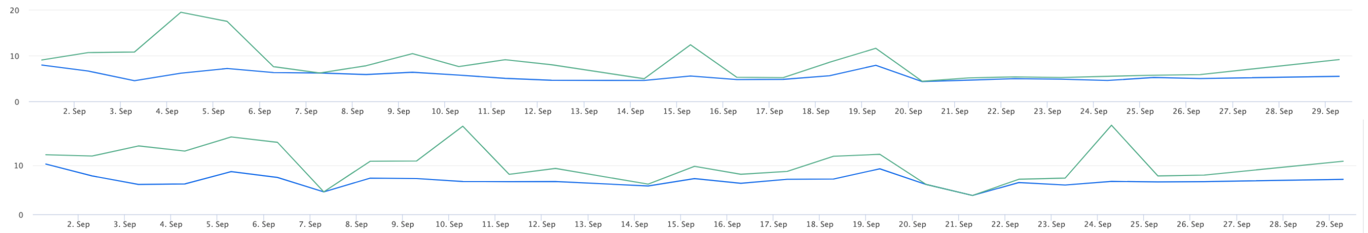


Fig. 6. Execution times for the test suites. The horizontal axis shows the date on which the sample of test suite execution times was taken. The vertical axes show the elapsed (wall clock) time in minutes from initiating the test to receiving the test suite outcomes as a signal, per test suite. The upper sub-figure is for regression test performance, while the lower sub-figure is for metamorphic test performance. Within each sub-figure, the upper line shows the maximum execution time observed in the associated time period, while the lower line shows the average execution time for that period.

Test name	Participating personas	Execution Mode (online/offline)	#Runs	#Flaky	#Pass	#Fail	#Broken
Somewhat flaky tests							
Replicability	harmful behaviour	online	81	21	1	59	0
Transitivity	arbitrary persona	online	81	10	66	3	2
Feature hiding	arbitrary persona	online	81	4	77	0	0
Minimal mechanism equivalence	(harmful behaviour, random behaviour)	online	81	1	79	0	1
Non flaky tests							
Replicability	non-harmful behaviour	offline	93	0	92	0	1
Replicability	non-harmful behaviour	online	81	0	80	0	1
Replicability	harmful behaviour	offline	81	0	81	0	0
Transitivity	arbitrary persona	offline	81	0	80	0	1
Minimal mechanism equivalence	(partially harmful behaviour, random behaviour)	offline	81	0	81	0	0
Minimal mechanism equivalence	(harmful behaviour, random behaviour)	online	81	0	81	0	0
Minimal mechanism equivalence	(harmful behaviour, random behaviour)	offline	81	0	81	0	0
Feature hiding	arbitrary persona	offline	81	0	81	0	0
Persona variance	(40-50 year old demographic, random)	online	81	0	72	9	0
Persona variance	(40-50 year old demographic, random)	offline	81	0	0	81	0
Demographic homophilia	(20-30 year old demographic, 50-60 year old demographic)	online	81	0	81	0	0
Demographic homophilia	(20-30 year old demographic, 50-60 year old demographic)	offline	81	0	81	0	0
Totals			1,308	36	1,114	152	6

Summary statistics on test flakiness

# Flaky runs (runs which both passed and failed on at least one of three sequential executions)	36	(3% of all runs (6% of all online runs))
# Online flaky runs (online mode runs which both passed and failed on at least one of three sequential executions)	36	(100% of all flaky runs)
# Offline flaky runs (offline mode runs which both passed and failed on at least one of three sequential executions)	0	(0% of all flaky runs)
# Flaky tests (test cases that exhibited at least one flaky triple of runs in the 3 day period studied)	4	(25% of all tests)
# Online flaky tests (online test cases that exhibited at least one flaky triple of runs in the 3 day period studied)	4	(50% of all online tests)
# Offline flaky tests (offline test cases that exhibited at least one flaky triple of runs in the 3 day period studied)	0	(0% of all offline tests)

TABLE II
RESULTS ON ONLINE AND OFFLINE METAMORPHIC TEST FLAKINESS

VII. SUBTLE SIMULATION ERROR BUGS CAUGHT

For reasons of confidentiality (and to avoid revealing potential counter–measures to potential harmful behaviours) we cannot give specific details of bugs found here. Instead, in this section, we give some examples of two *kinds* of bugs found to provide a simple flavour of our experience of MIA's bug revelation at Facebook.

Feature Interaction Bugs: We found that metamorphic testing is well–placed to reveal feature interaction bugs, because of the way it relates different features. For instance MIA, when we combine the capture–replay feature and the random bot decision making feature, a question arises:

> Should the behaviour of the bot be random, or should the previous random number choices be captured and then re-played?

Random decision-making is guided by a pseudo random number generator, and it is important that the next random number in the sequence is determined solely by the seed, which in turn, determines the specific sample from the population of all possible behaviours.

In this way, the seed can be used to sample from all possible bot behaviours. For this reason, when *both* random decision-making *and* capture–replay are enabled, there is an interesting feature interaction: random decision-making should take priority over capture–replay. Metamorphic testing revealed an interaction fault, in which the code incorrectly gave precedence to capture–replay over random decision-making. Other testing approaches may be less likely to distinguish such a subtle, yet important, difference.

Corner Case Bug Revelation: Metamorphic tests often involve corner cases or special behaviours, so a metamorphic test also plays the role of a pair (or more) of end–to–end tests that explore such corner cases. We found several subtle bugs were revealed simply by the execution of each test in the metamorphic pair (without needing to make the relational comparison). One example of this was a persona misconfiguration, in which the environment used to compute the bots' initial personas was incorrectly configured to include the mechanism. In most cases such a bug could be missed, because most mechanisms are highly unlikely to impact persona choices.

Nevertheless, the metamorphic tests include those that cover a particular restrictive mechanism that limits the rate limit with a 'minimal mechanism equivalence' test (see Table I). This mechanism proved to be sufficiently restrictive such that the persona misconfiguration was revealed. The reason was that too few friends were available to form a meaningful bot persona, leading to a failure being induced and flagged by MIA.

VIII. Open Problems and Directions for Further Impact from Metamorphic Testing

Here, we briefly review some of the remaining open problems for this research agenda on metamorphic testing of simulations.

Metamorphic Testing for Simulation Oracles: It is clearly important to define further metamorphic relations, and to define them for other simulations, so that we can better understand the commonalities. It would also be useful to identify techniques for automatically inferring metamorphic relations.

Probabilistic Metamorphic Testing: Simulations are inherently statistical, and therefore statistical metamorphic testing techniques [36] ought to prove a good fit. More work is needed on probabilistic inference techniques for metamorphic testing, and further development of statistical metamorphic testing.

Theoretical Relationships between Regression and Metamorphic Testing: This paper further developed the relationship [32] between regression testing and metamorphic testing. We deployed regression testing as a special case of metamorphic testing, but more work is required on the theoretical connections between these two apparently different, but clearly related techniques for testing.

Flakiness: In this paper we showed that offline computation can be used as a way to tackle flakiness problems. However, more work is needed to understand the cost–benefit trade off when pre-computing large amounts of simulation data in this offline manner. We also know that it remains sensible to assume that all tests are essentially flaky [2]. Therefore, further research is required on techniques to adapt statistical metamorphic testing to handle flakiness as a special case.

Metamorphic Testability: We needed extra event logging in order to facilitate metamorphic testing. We also augmented the simulation input space in order to facilitate metamorphic testing. These features were added to support testability [52]. For example, we allowed the traversal to start at the same initial point, specifically (and solely) to enable metamorphic tests, suggesting possible further applications of testability transformation [53].

Automated Diagnostics and Debugging: More work is required to automate the process of diagnosis and debugging. Techniques such as automated debugging [54] and spectrum based fault localisation [55], [56] could prove useful here, but may need adaption to handle metamorphic testing scenarios. Ultimately, we might even hope that some faults could be automatically fixed [24], [57].

IX. Conclusions

We presented our experience of the MIA metamorphic testing system at Facebook. This is the first industrial scale deployment of metamorphic testing reported in the literature. MIA supports general metamorphic testing and regression testing as a continuous test process, and also as a bot commenting on each change submitted in code review. MIA includes a dashboard that is monitored for Continuous Integration within a DevOps setting. MIA uses automated reporting and alarms to developers and diagnostics on test performance. We reported on our experience of the system and the kinds of faults it has revealed thus far.

References

[1] J. Ahlgren, M. E. Berezin, K. Bojarczuk, E. Dulskyte, I. Dvortsova, J. George, N. Gucevska, M. Harman, R. Laemmel, E. Meijer, S. Sapora, and J. Spahr-Summers, "WES: Agent-based user interaction simulation on real infrastructure," in *GI @ ICSE 2020*, S. Yoo, J. Petke, W. Weimer, and B. R. Bruce, Eds. ACM, 3 Jul. 2020, pp. 276–284, invited Keynote.

[2] M. Harman and P. O'Hearn, "From start-ups to scale-ups: Opportunities and open problems for static and dynamic program analysis (keynote paper)," in 18^{th} *IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM 2018)*, Madrid, Spain, September 23rd-24th 2018, pp. 1–23.

[3] Q. Luo, F. Hariri, L. Eloussi, and D. Marinov, "An empirical analysis of flaky tests," in 22^{nd} *International Symposium on Foundations of Software Engineering (FSE 2014)*, S.-C. Cheung, A. Orso, and M.-A. Storey, Eds. Hong Kong, China: ACM, November 16 - 22 2014, pp. 643–653.

[4] E. J. Weyuker, "On testing non-testable programs," *The Computer Journal*, vol. 25, no. 4, pp. 465–470, Nov. 1982.

[5] S. Terzi and S. Cavalieri, "Simulation in the supply chain context: a survey," *Computers in Industry*, vol. 53, no. 1, pp. 3–16, 2004.

[6] G. L. Johnson, C. L. Hanson, S. P. Hardegree, and E. B. Ballard, "Stochastic weather simulation: Overview and analysis of two commonly used models," *Journal of Applied Meteorology*, vol. 35, no. 10, pp. 1878–1896, 1996.

[7] S. Al-Sultan, M. M. Al-Doori, A. H. Al-Bayatti, and H. Zedan, "A comprehensive survey on vehicular ad hoc network," *Journal of Network and Computer Applications*, vol. 37, pp. 380 – 392, 2014.

[8] D. Adam, "Special report: The simulations driving the world's response to COVID-19," *Nature*, 2020.

[9] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The oracle problem in software testing: A survey," *IEEE Transactions on Software Engineering*, vol. 41, no. 5, pp. 507–525, May 2015.

[10] T. Y. Chen, J. Feng, and T. H. Tse, "Metamorphic testing of programs on partial differential equations: A case study," in 26^{th} *Annual International Computer Software and Applications Conference (COMPSAC'02)*. IEEE Computer Society, 2002, pp. 327–333.

[11] S. Segura, G. Fraser, A. B. Sánchez, and A. R. Cortés, "A survey on metamorphic testing," *IEEE Transactions on Software Engineering*, vol. 42, no. 9, pp. 805–824, 2016.

[12] T. Y. Chen, F.-C. Kuo, H. Liu, P.-L. Poon, D. Towey, T. H. Tse, and Z. Q. Zhou, "Metamorphic testing: a review of challenges and opportunities," vol. 51, no. 1, pp. 4:1–4:27, January 2018.

[13] Z. Q. Zhou, S. Zhang, M. Hagenbuchner, T. H. Tse, F.-C. Kuo, and T. Y. Chen, "Automated functional testing of online search services," *Software Testing, Verification and Reliability*, vol. 22, no. 4, pp. 221–243, 2012.

[14] F.-C. Kuo, T. Y. Chen, and W. K. Tam, "Testing embedded software by metamorphic testing: A wireless metering system case study," in *IEEE 36th Conference on Local Computer Networks, (LCN 2011), Bonn, Germany, October 4-7, 2011*. IEEE Computer Society, 2011, pp. 291–294.

[15] A. F. Donaldson, "Metamorphic testing of android graphics drivers," in *Proceedings of the 4th International Workshop on Metamorphic Testing, MET@ICSE 2019, Montreal, QC, Canada, May 26, 2019*, X. Xie, P.-L. Poon, and L. L. Pullum, Eds. IEEE / ACM, 2019, p. 1.

[16] T. Chen, S. Cheung, and S. Yiu, "Metamorphic testing: A new approach for generating next test cases," Department of Computer Science, Hong Kong University of Science and Technology, Tech. Rep. HKUST-CS98-01, 1998.

[17] S. Yoo and M. Harman, "Regression testing minimisation, selection and prioritisation: A survey," *Journal of Software Testing, Verification and Reliability*, vol. 22, no. 2, pp. 67–120, 2012.

[18] A. M. Memon, Z. Gao, B. N. Nguyen, S. Dhanda, E. Nickell, R. Siemborski, and J. Micco, "Taming Google-scale continuous testing," in 39^{th} *International Conference on Software Engineering, Software Engineering in Practice Track (ICSE-SEIP)*. Buenos Aires, Argentina: IEEE, May 20-28 2017, pp. 233–242.

[19] F. Palomba and A. Zaidman, "Does refactoring of test smells induce fixing flakey tests?" in *International conference on software maintenance and evolution (ICSME 2017)*. IEEE Computer Society, 2017, pp. 1–12.

[20] N. Alshahwan, X. Gao, M. Harman, Y. Jia, K. Mao, A. Mols, T. Tei, and I. Zorin, "Deploying search based software engineering with Sapienz at Facebook (keynote paper)," in 10^{th} *International Symposium on Search Based Software Engineering (SSBSE 2018)*, Montpellier, France, September 8th-10th 2018, pp. 3–45, springer LNCS 11036.

[21] J. P. Kleijnen, "Supply chain simulation tools and techniques: a survey," *International journal of simulation and process modelling*, vol. 1, no. 1-2, pp. 82–89, 2005.

[22] F. Michel, J. Ferber, and A. Drogoul, "Multi-agent systems and simulation: A survey from the agent community's perspective," in *Multi-Agent Systems*. CRC Press, 2018, pp. 17–66.

[23] L. Hurwicz and S. Reiter, *Designing Economic Mechanisms*. Cambridge University Press, 2006.

[24] A. Marginean, J. Bader, S. Chandra, M. Harman, Y. Jia, K. Mao, A. Mols, and A. Scott, "SapFix: Automated end-to-end repair at scale," in *International Conference on Software Engineering (ICSE) Software Engineering in Practice (SEIP) track*, Montreal, Canada, 2019.

[25] J. Petke, S. O. Haraldsson, M. Harman, W. B. Langdon, D. R. White, and J. R. Woodward, "Genetic improvement of software: a comprehensive survey," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 3, pp. 415–432, Jun. 2018.

[26] M. Harman, A. Mansouri, and Y. Zhang, "Search based software engineering: Trends, techniques and applications," *ACM Computing Surveys*, vol. 45, no. 1, pp. 11:1–11:61, November 2012.

[27] M. Harman and B. F. Jones, "Search based software engineering," *Information and Software Technology*, vol. 43, no. 14, pp. 833–839, Dec. 2001.

[28] J. Bell, G. Kaiser, E. Melski, and M. Dattatreya, "Efficient dependency detection for safe Java test acceleration," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 770–781.

[29] M. Hilton, N. Nelson, T. Tunnell, D. Marinov, and D. Dig, "Trade-offs in continuous integration: assurance, security, and flexibility," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 2017, pp. 197–207.

[30] T. Chen, D. Huang, H. Huang, T.-H. Tse, Z. Yang, and Z. Zhou, "Metamorphic testing and its applications," in *Proceedings of the 8th International Symposium on Future Software Technology*, ser. ISFST 2004, 2004, pp. 310–319.

[31] F. T. Chan, T. Y. Chen, S. C. Cheung, M. F. Lau, and S. M. Yiu, "Application of metamorphic testing in numerical analysis," in *Proceedings of the IASTED International Conference on Software Engineering*, 1998, pp. 191–197.

[32] M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "A comprehensive survey of trends in oracles for software testing," Department of Computer Science, University of Sheffield, Tech. Rep. Research Memoranda CS-13-01, 2013.

[33] L. C. Briand, J. Feng, and Y. Labiche, "Using genetic algorithms and coupling measures to devise optimal integration test orders," in *Software Engineering and Knowledge Engineering (SEKE 02)*, 2002, pp. 43–50.

[34] C. Murphy, K. Shen, and G. Kaiser, "Using JML runtime assertion checking to automate metamorphic testing in applications without test oracles," *2009 International Conference on Software Testing Verification and Validation*, pp. 436–445, 2009.

[35] ——, "Automatic system testing of programs without test oracles," in *ISSTA*. ACM Press, 2009, pp. 189–200.

[36] R. Guderlei and J. Mayer, "Statistical metamorphic testing testing programs with random output by means of statistical hypothesis tests and metamorphic testing," in *QSIC*, October 2007, pp. 404–409.

[37] S. Yoo, "Metamorphic testing of stochastic optimisation," in *Proceedings of the 2010 Third International Conference on Software Testing, Verification, and Validation Workshops*, ser. ICSTW '10. IEEE Computer Society, 2010, pp. 192–201.

[38] A. M. Memon and M. B. Cohen, "Automated testing of GUI applications: models, tools, and controlling flakiness," in 35^{th} *International Conference on Software Engineering (ICSE 2013)*, D. Notkin, B. H. C. Cheng, and K. Pohl, Eds. San Francisco, CA, USA: IEEE Computer Society, May 18-26 2013, pp. 1479–1480.

[39] C. Sadowski, J. van Gogh, C. Jaspan, E. Söderberg, and C. Winter, "Tricorder: Building a program analysis ecosystem," in *37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 1*, 2015, pp. 598–608.

[40] N. Alshahwan and M. Harman, "Automated web application testing using search based software engineering," in 26^{th} *IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*, Lawrence, Kansas, USA, 6th - 10th November 2011, pp. 3 – 12.

[41] M. A. Chang, B. Tschaen, T. Benson, and L. Vanbever, "Chaos Monkey: Increasing SDN reliability through systematic network destruction," *Computer Communication Review*, vol. 45, no. 5, pp. 371–372, 2015.

[42] D. G. Feitelson, E. Frachtenberg, and K. L. Beck, "Development and deployment at Facebook," *IEEE Internet Computing*, vol. 17, no. 4, pp. 8–17, 2013.

[43] F. Ferrucci, M. Harman, J. Ren, and F. Sarro, "Not going to take this anymore: Multi-objective overtime planning for software engineering projects," in 35^{th} *ACM/IEEE International Conference on Software Engineering (ICSE 2013)*, San Francisco, USA, 2013.

[44] T. Y. Chen, F.-C. Kuo, T. H. Tse, and Z. Q. Zhou, "Metamorphic testing and beyond," in *Proceedings of the International Workshop on Software Technology and Engineering Practice (STEP 2003)*, September 2004, pp. 94–100.

[45] W. Chan, S. Cheung, and K. R. Leung, *A metamorphic testing approach for online testing of service-oriented software applications*. IGI Global, 2009, ch. 7, pp. 2894–2914.

[46] S. Segura, R. M. Hierons, D. Benavides, and A. Ruiz-Cortés, "Automated metamorphic testing on the analyses of feature models," *Information and Software Technology*, vol. 53, no. 3, pp. 245 – 258, 2011.

[47] C. Nie and H. Leung, "A survey of combinatorial testing," *ACM Computing Surveys*, vol. 43, no. 2, pp. 11:1 – 11:29, 2011.

[48] J. Petke, M. B. Cohen, M. Harman, and S. Yoo, "Practical combinatorial interaction testing: Empirical findings on efficiency and early fault detection," *IEEE Transactions on Software Engineering*, vol. 41, no. 9, pp. 901–924, 2015.

[49] M. McPherson, L. Smith-Lovin, and J. M. Cook, "Birds of a feather: Homophily in social networks," *Annual Review of Sociology*, vol. 27, no. 1, pp. 415–444, 2001.

[50] L. Backstrom, P. Boldi, M. Rosa, J. Ugander, and S. Vigna, "Four degrees of separation," *CoRR*, vol. abs/1111.4570v3, 2012.

[51] S. J. Eglen, "CODECHECK certificate for paper: Report 9: impact of non-pharmaceutical interventions (NPIs) to reduce COVID-19 mortality and healthcare demand," May 2020.

[52] J. M. Voas and K. W. Miller, "Software testability: The new verification," *IEEE Software*, vol. 12, no. 3, pp. 17–28, May 1995.

[53] M. Harman, L. Hu, R. M. Hierons, J. Wegener, H. Sthamer, A. Baresel, and M. Roper, "Testability transformation," *IEEE Transactions on Software Engineering*, vol. 30, no. 1, pp. 3–16, Jan. 2004.

[54] A. Zeller, "Beautiful debugging," in *Beautiful Code*, A. Oram and G. Wilson, Eds. Sebastopol, CA 95472: O'Reilly & Associates, Inc., 2007, pp. 463–476, chapter 28.

[55] W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa, "A survey on software fault localization," *IEEE Trans. Software Eng*, vol. 42, no. 8, pp. 707–740, 2016.

[56] X. Xie, F.-C. Kuo, T. Y. Chen, S. Yoo, and M. Harman, "Provably optimal and human-competitive results in SBSE for spectrum based fault localisation," in 5^{th} *International Symposium on Search Based Software Engineering (SSBSE '13)*, vol. 8084. St. Petersburg, Russia: Springer, 24-26 August 2013, pp. 224–238.

[57] C. Le Goues, T. Nguyen, S. Forrest, and W. Weimer, "GenProg: A generic method for automatic software repair," *IEEE Transactions on Software Engineering*, vol. 38, no. 1, pp. 54–72, 2012.