Autonomic Testing: Testing with Scenarios from Production

Ketai Qiu
Faculty of Informatics, Università della Svizzera Italiana (USI)
Lugano, Switzerland
ketai.qiu@usi.ch

ABSTRACT

My PhD addresses the problem of detecting field failures with a new approach to test software systems under conditions that emerge only in production. Ex-vivo approaches detect field failures by executing the software system in the testbed with data extracted from the production environment. In-vivo approaches execute the available test suites in the production environment. We will define autonomic testing that detects conditions that emerge only in production scenarios, generates test cases for the new conditions, and executes the generated test cases in the new scenarios, to detect failures before they occur in production.

CCS CONCEPTS

• **Software and its engineering → Software testing and debugging**.

KEYWORDS

Autonomic Testing, Failure Detection, Test Generation

ACM Reference Format:
Ketai Qiu. 2024. Autonomic Testing: Testing with Scenarios from Production. In *2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion '24), April 14–20, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3639478.3639802

1 INTRODUCTION

Field failures, that is failures in the production environment, cannot be avoided [2, 7, 10, 14, 15, 23].

In-house testing, that is performed in testbed, can effectively reveal many software faults [1, 13], however, it is impossible to exercise all faulty configurations and detect all faulty statements in complex software applications. Recent studies also discuss failures that occur in scenarios that emerge only in the production environment, and thus cannot be detected in testbed [10, 11, 20]. For example, the Eclipse plugin *EclipseLink* developed before Java 8 failed to generate tables for persisting objects that included the *@Entity* annotation, since it silently ignored classes that contained lambda expressions that were introduced in Java 8 [12]. Testing in testbed executed before the Java 8 release could not detect the failure. Software faults that escape testing can result in severe consequences. For example, the UK air traffic control system crash that caused a financial loss of $126 million in September 2023 was due to an unexpected flight plan that had never been encountered before [22].

There are several studies to detect field failures by testing software systems. As well discussed in the excellent survey of Bertolino et al. [4], *ex-vivo* approaches test the system in the development environment with data extracted from the field, which focus on abstracting test templates to be instantiated with new input data, ignoring environmental conditions (e.g., [9]), while *in-vivo* testing techniques test the software system directly in production, which concentrate on the issues of executing test cases without interfering with the running system (e.g., [17]).

In this PhD, we will define autonomic testing, a new approach to detect failures that escape testing in production. Autonomic testing tests software applications with full scenarios from the production environment. Autonomic testing considers the whole environment that emerges in production, thus differs from both *ex-vivo* approaches that consider only some information extracted from the environment and *in-vivo* approaches that focus on the execution rather than on the inputs.

Autonomic testing generates test cases, which are not only simple abstractions of existing ones, with conditions that emerge only in production. Autonomic testing monitors the running system to detect emerging conditions, generates test cases to thoroughly test the system with the detected conditions, and executes the generated test cases without interfering with the production environment. We detect emerging conditions with either dynamic models of the tested system or neural networks. We use dynamic models to detect conditions that emerge at the unit level, similarly to extend Gazzola et al.'s approach that exploits dynamic grammars to model the tested conditions and reveal strings that emerge in the execution [9]. We leverage neural networks to detect anomalies that emerge at the integration level, inspired from Denaro et al.'s *Prevent* approach to detect anomalies in complex cloud systems [7]. We define autonomic tests to substitute mocks that characterize operational tests with probes in the execution environment. We generate test oracles for autonomic tests with natural language processing (NLP) and large language models (LLMs) inspired from Blasi et al.'s Jdoctor approach [5]. We execute autonomic tests on digital twins to avoid interferences with production.

In the PhD, we address following research questions:

RQ1 *Can autonomic testing detect field failures effectively and efficiently before they lead to catastrophic consequences in production?*

We plan to study the effectiveness and efficiency of autonomic testing by referring to following metrics: the quantity of detected failures with respect to well documented field failures, the earliness of failure detection, that is the interval between the detection and the otherwise unavoidable failure, and each component's runtime overhead.

ICSE-Companion '24, April 14–20, 2024, Lisbon, Portugal
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0502-1/24/04.
https://doi.org/10.1145/3639478.3639802

RQ2 *To what extent can the runtime information collected from production detect emerging conditions?*

We will study the correlation between different types of runtime information using common statistical methods and evaluate their usefulness in terms of contributions to the final decision of the test trigger using popular machine learning techniques after processing them into standard formats.

RQ3 *To what extent can the execution of autonomic testing affect the performance of the deployed software application?*

We plan to study the performance overhead of autonomic testing in terms of required resources and extra latency.

2 RELATED WORK

There are numerous approaches to detect field failures.

Orso et al.'s Gamma approach partially instruments the code to monitor software systems in production with low impact, and gather information for continuous evolution [19], impact analysis and regression testing [18]. Yabandeh et al.'s CrystalBall predicts the consequences of distributed nodes' actions and prevents future inconsistencies at runtime [21].

Some approaches leverage monitored information to generate test cases for detecting failures that may occur in production. Metzger et al. [16] investigate a proactive approach to integrate the monitoring service-based applications with online testing for failure prediction. Chaos Monkey tests the resilience of the deployed services by randomly terminating virtual machines that host production services [3]. Both Metzger et al.'s approach and Chaos Monkey rely on intrusive operations with significant side-effects.

Many approaches propose sandboxing techniques to execute tests in production without interfering with the running system. Murphy et al. [17] propose *in vivo* testing to execute test suites with objects extracted from the runtime status that the execution reaches after the call of the methods under test, without interfering with users. The *in vivo* testing framework leverages unit tests and program invariants created by testers while developing the software to both test a set of methods-under-observation when they are invoked and record the test results. The *in vivo* testing framework executes the methods-under-observation by forking the methods, with a significant amount of overhead when dealing with a large number of methods-under-observation.

Gazzola et al. [9] define *field-ready* testing, an approach to detect faults by generating and executing test cases with scenarios that emerge in production.

As Bertolino et al. [4] well summarize in their survey, the approaches proposed so far to test software in production mostly focus on the problem of executing test cases without interfering with the running systems, and rely on existing test cases and simple templates to generate test cases with data from the field.

3 EXPECTED CONTRIBUTIONS

In this PhD we focus on the problem of automatically generating and executing test cases that exercise conditions that emerge only in production. We define autonomic testing, an approach to trigger execution scenarios that emerge only in production, and generate test cases that exercise the system in the new scenarios to detect otherwise unavoidable failures.

We exploit dynamic analysis to identify scenarios that involve small clusters of classes, and machine learning to detect scenarios in large subsystems, where dynamic analysis would be impractical. Our dynamic analysis extends the dynamic analysis of Gazzola et al. [9]. Gazzola et al.'s triggers dynamically identify objects that have not been tested yet with generative grammars, and are limited to objects of type string. Our triggers will handle values of different types with various dynamic analysis techniques, and scale to large subsystems with suitably trained neural networks that detect emerging conditions.

We define a new type of artifact, autonomic tests, that substitute the mocks of classic tests with probes. While mocks artificially build the conditions for executing the test calls, probes gather the conditions to execute the test calls from the execution environment. For example, the autonomic tests for a web shopping application will not mock shopping carts and goods, but will get the values of shopping carts and goods that emerge in production from the production environment, thus exercising the application in the conditions that emerge in production.

We will extend the state-of-the-art techniques that generate oracles from commonly available information (e.g., comments) by means of NLP, such as TOGA [8] and CallMeMaybe [6], to automatically generate assertion oracles for conditions that emerge in the field.

We will largely rely on existing technology for sandbox execution and digital twins to execute autonomic tests without interfering with the production environment.

4 EVALUATION CRITERIA

We plan to evaluate our approach in terms of both effectiveness and efficiency. We measure the effectiveness of the approach in terms of both the number of detected failures and detection earliness. We measure the number of failures that the approach can detect, by referring to both field failures reported in the literature and new field failures that the approach reveals. We measure the earliness of the detection as the interval between the detection and the incorrect behavior that the user would perceive if the failure occurs.

We measure the efficiency of the approach in terms of both the runtime overhead and the number of test cases that the approach generates and executes to detect a failure, that is, the ratio between the number of detected failures and the number of generated test cases. We measure the overhead of the triggers as the time needed to distinguish unexpected scenarios from common ones. We measure the overhead of the test case generator as the time needed to generate test cases with conditions that emerge in production.

We plan to experiment on several widely used open-source projects and datasets. We will also conduct a comprehensive comparison with state-of-the-art approaches. We will start from well-known Java projects used in [9] as the initial benchmark. We will move forward with complex systems that contain complicated events and user interactions, for instance, learning management systems and online shopping applications.

REFERENCES

[1] M. Moein Almasi, Hadi Hemmati, Gordon Fraser, Andrea Arcuri, and Janis Benefelds. 2017. An Industrial Evaluation of Unit Test Generation: Finding Real Faults in a Financial Application. In *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*. 263–272. https://doi.org/10.1109/ICSE-SEIP.2017.27

[2] Joy Arulraj, Po-Chun Chang, Guoliang Jin, and Shan Lu. 2013. Production-Run Software Failure Diagnosis via Hardware Performance Counters. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems* (Houston, Texas, USA) *(ASPLOS '13)*. Association for Computing Machinery, New York, NY, USA, 101–112. https://doi.org/10.1145/2451116.2451128

[3] Ali Basiri, Niosha Behnam, Ruud de Rooij, Lorin Hochstein, Luke Kosewski, Justin Reynolds, and Casey Rosenthal. 2016. Chaos Engineering. *IEEE Software* 33, 3 (2016), 35–41. https://doi.org/10.1109/MS.2016.60

[4] Antonia Bertolino, Pietro Braione, Guglielmo De Angelis, Luca Gazzola, Fitsum Kifetew, Leonardo Mariani, Matteo Orrù, Mauro Pezzè, Roberto Pietrantuono, Stefano Russo, and Paolo Tonella. 2021. A Survey of Field-Based Testing Techniques. *ACM Comput. Surv.* 54, 5, Article 92 (may 2021), 39 pages. https://doi.org/10.1145/3447240

[5] Arianna Blasi, Alberto Goffi, Konstantin Kuznetsov, Alessandra Gorla, Michael D. Ernst, Mauro Pezzè, and Sergio Delgado Castellanos. 2018. Translating code comments to procedure specifications. In *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis* (Amsterdam, Netherlands) *(ISSTA 2018)*. Association for Computing Machinery, New York, NY, USA, 242–253. https://doi.org/10.1145/3213846.3213872

[6] Arianna Blasi, Alessandra Gorla, Michael D. Ernst, and Mauro Pezzè. 2023. Call Me Maybe: Using NLP to Automatically Generate Unit Test Cases Respecting Temporal Constraints. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering* (Rochester, MI, USA) *(ASE '22)*. Association for Computing Machinery, New York, NY, USA, Article 19, 11 pages. https://doi.org/10.1145/3551349.3556961

[7] Giovanni Denaro, Rahim Heydarov, Ali Mohebbi, and Mauro Pezzè. 2023. Prevent: An Unsupervised Approach to Predict Software Failures in Production. *IEEE Transactions on Software Engineering* (2023), 1–15. https://doi.org/10.1109/TSE.2023.3327583

[8] Elizabeth Dinella, Gabriel Ryan, Todd Mytkowicz, and Shuvendu K. Lahiri. 2022. TOGA: A Neural Method for Test Oracle Generation. In *Proceedings of the 44th International Conference on Software Engineering* (Pittsburgh, Pennsylvania) *(ICSE '22)*. Association for Computing Machinery, New York, NY, USA, 2130–2141. https://doi.org/10.1145/3510003.3510141

[9] Luca Gazzola, Leonardo Mariani, Matteo Orrú, Mauro Pezzè, and Martin Tappler. 2022. Testing Software in Production Environments with Data from the Field. In *2022 IEEE Conference on Software Testing, Verification and Validation (ICST)*. 58–69. https://doi.org/10.1109/ICST53961.2022.00017

[10] Luca Gazzola, Leonardo Mariani, Fabrizio Pastore, and Mauro Pezzè. 2017. An Exploratory Study of Field Failures. In *2017 IEEE 28th International Symposium on Software Reliability Engineering (ISSRE)*. 67–77. https://doi.org/10.1109/ISSRE.2017.10

[11] Maggie Hamill and Katerina Goseva-Popstojanova. 2009. Common Trends in Software Fault and Failure Data. *IEEE Transactions on Software Engineering* 35, 4 (2009), 484–496. https://doi.org/10.1109/TSE.2009.3

[12] Davy Herben. 2014. Bug 429992 - EclipseLink silently ignores Entity classes with lambda expressions. (10 March 2014). https://bugs.eclipse.org/bugs/show_bug.cgi?id=429992

[13] He Jiang, Xiaochen Li, Zijiang Yang, and Jifeng Xuan. 2017. What Causes My Test Alarm? Automatic Cause Analysis for Test Alarms in System and Integration Testing. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. 712–723. https://doi.org/10.1109/ICSE.2017.71

[14] Baris Kasikci, Benjamin Schubert, Cristiano Pereira, Gilles Pokam, and George Candea. 2015. Failure Sketching: A Technique for Automated Root Cause Diagnosis of in-Production Failures. In *Proceedings of the 25th Symposium on Operating Systems Principles* (Monterey, California) *(SOSP '15)*. Association for Computing Machinery, New York, NY, USA, 344–360. https://doi.org/10.1145/2815400.2815412

[15] Bev Littlewood and Lorenzo Strigini. 1993. Validation of Ultrahigh Dependability for Software-Based Systems. *Commun. ACM* 36, 11 (nov 1993), 69–80. https://doi.org/10.1145/163359.163373

[16] Andreas Metzger, Osama Sammodi, Klaus Pohl, and Mark Rzepka. 2010. Towards Pro-Active Adaptation with Confidence: Augmenting Service Monitoring with Online Testing. In *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems* (Cape Town, South Africa) *(SEAMS '10)*. Association for Computing Machinery, New York, NY, USA, 20–28. https://doi.org/10.1145/1808984.1808987

[17] Christian Murphy, Gail Kaiser, Ian Vo, and Matt Chu. 2009. Quality Assurance of Software Applications Using the In Vivo Testing Approach. In *2009 International Conference on Software Testing Verification and Validation*. 111–120. https://doi.org/10.1109/ICST.2009.18

[18] Alessandro Orso, Taweesup Apiwattanapong, and Mary Jean Harrold. 2003. Leveraging Field Data for Impact Analysis and Regression Testing. In *Proceedings of the 9th European Software Engineering Conference Held Jointly with 11th ACM SIGSOFT International Symposium on Foundations of Software Engineering* (Helsinki, Finland) *(ESEC/FSE-11)*. Association for Computing Machinery, New York, NY, USA, 128–137. https://doi.org/10.1145/940071.940089

[19] Alessandro Orso, Donglin Liang, Mary Jean Harrold, and Richard Lipton. 2002. Gamma System: Continuous Evolution of Software after Deployment. In *Proceedings of the 2002 ACM SIGSOFT International Symposium on Software Testing and Analysis* (Roma, Italy) *(ISSTA '02)*. Association for Computing Machinery, New York, NY, USA, 65–69. https://doi.org/10.1145/566172.566182

[20] Sina Shamshiri, René Just, José Miguel Rojas, Gordon Fraser, Phil McMinn, and Andrea Arcuri. 2015. Do Automatically Generated Unit Tests Find Real Faults? An Empirical Study of Effectiveness and Challenges. In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 201–211. https://doi.org/10.1109/ASE.2015.86

[21] Maysam Yabandeh, Nikola Knežević, Dejan Kostić, and Viktor Kuncak. 2010. Predicting and Preventing Inconsistencies in Deployed Distributed Systems. *ACM Trans. Comput. Syst.* 28, 1, Article 2 (aug 2010), 49 pages. https://doi.org/10.1145/1731060.1731062

[22] Sarah Young. 2023. UK air traffic control meltdown fault won't happen again - NATS. *Reuters* (30 September 2023). https://www.reuters.com/world/uk/uk-air-traffic-control-says-problem-which-caused-flight-cancellations-wont-2023-08-30/

[23] Andreas Zeller. 2009. *Why programs fail: a guide to systematic debugging*. Elsevier.