Metamorphic Testing

Sergio Segura
University of Seville, Spain

ICSE MET 2018 - May 2018
Applied Software Engineering Research Group

21 members
18 senior research staff
3 research assistants
(>20 former members)

- Service governance
- Business process management
- Software product lines
- Software testing
- Empirical software engineering
- SBSE
Some time ago...

Hey, have you heard about metamorphic testing?

WHAT?

Oh, that simple technique to test numerical properties...

Not anymore!

Let's talk about serious stuff.
Metamorphic testing

Combinatorial testing
Search-based testing
Mutation testing
Community
Performance indicators

Publication Sets
- Combinatorial testing
- Metamorphic testing
- Mutation testing
- Search-based software testing

Scholarly Output vs. Publication Year
Publications in top journal percentiles (2012-2017)

Publications in top 10% journals according to Scopus ranking

- Combinatorial testing: 13.0%
- Metamorphic testing: 34.5% (1st)
- Mutation testing: 14.5%
- Search-based software testing: 27.1%
Community
Performance indicators

Outputs in top citation percentiles (2012-2017)
Publications in top 10% most cited worldwide

- Combinatorial testing: 7.7%
- Metamorphic testing: 9.1%
- Mutation testing: 9.1%
- Search-based software testing: 13.0%
Community
Performance indicators

International collaboration (2012-2017)
Publications co-authored with institutions from different countries

- Combinatorial testing: 25.5%
- Metamorphic testing: 37.4% (1st)
- Mutation testing: 26.3%
- Search-based software testing: 29.2%
Challenge

Keep current growing publication trend

- Better quality than quantity.
- Target top conferences and journals.
- Foster international collaborations.
- MET as a key meeting point.
How is MET doing?
Community
1st-3rd International workshops

Number of papers
- Mutation testing
- Combinatorial testing
- Search-based software testing
- Metamorphic testing

Number of authors

Number of countries
Community
1st-3rd International workshops

Combinatorial testing
Co-located with ICST

Metamorphic testing
Co-located with ICSE

Mutation testing
Co-located with ICST

Search-based software testing
Co-located with ICST
Challenge

Promoting MET

- @Authors. keep submitting (good) papers.
- @Organizers. Foster participation, e.g.:
  - Awards (best paper, best presentation, etc.)
  - Sponsored prizes.
  - Invited talks, e.g. authors of journal papers published the previous year.
Sergio Segura @sergio_segura2 · 6 Dec 2015
First Int. Workshop on Metamorphic Testing accepted at @ICSEconf @icse2016.
Have a look to the CfP: cs.montana.edu/met16/

ICSE @ICSEconf · Jan 8
#ICSE2018Workshops [WAPI] 2nd International Workshop on API Usage and Evolution, w-api.github.io @wapi2018 @ysamarin @sarahmadi @khatrachicse2018.org/track/icse-201... [submission: Feb 05]

WAPI
2nd International Workshop on API Usage and Evolution
WAPI will be held on June 2, 2018, at the 18th International Conference on Software Engineering, ICSE 2018, in Gothenburg, Sweden.

ICSE @ICSEconf · Jan 8
#ICSE2018Workshops [APSA] 1st International Workshop on Anti-Patterns for Software Analytics, sherbold.github.io/apsa/ @HerboldSteffen @shane_mcintosh @ProfMShepperd @timmenzies icse2018.org/track/icse-201... [submission: Feb 05]

1st International Workshop on Anti-Patterns for Software Analytics (APSA)
In conjunction with ICSE 2018, Gothenburg, Sweden on 2 June 2018.

ICSE @ICSEconf · Jan 6
#ICSE2018Workshops [SEmotion '18] 3rd International Workshop on Emotion Awareness in Software Engineering, collab.di.uniba.it/semotion/ @SEmotion_ws @abegel @aserebrenik @dgraziotin icse2018.org/track/icse-201... [submission: Feb 05]

SEmotion '18
Third International Workshop on Emotion Awareness in Software Engineering
Community Dissemination

ACM SIGSOFT Webinar

- 1029 registrations
- 316 watched it live
- Many Q/As!!

https://goo.gl/rSWKUU
Community Dissemination

5  Test Design Techniques

5.1  Overview

5.2  Specification-Based Test Design Techniques

5.2.1  Equivalence Partitioning

5.2.2  Classification Tree Method

5.2.3  Boundary Value Analysis

5.2.4  Syntax Testing

5.2.5  Combinatorial Test Design Techniques

5.2.6  Decision Table Testing

5.2.7  Cause-Effect Graphing

5.2.8  State Transition Testing

5.2.9  Scenario Testing

5.2.10  Random Testing

5.3  Structure-Based Test Design Techniques

5.3.1  Statement Testing

5.3.2  Branch Testing

5.3.3  Decision Testing

5.3.4  Branch Condition Testing

5.3.5  Branch Condition Combination Testing

5.3.6  Modified Condition Decision Coverage (MCDC) Testing

5.3.7  Data Flow Testing

5.4  Experience-Based Test Design Techniques

5.4.1  Error Guessing
Challenge

Foster dissemination of MT

- YouTube, Wikipedia.
- Webinars, MOOCs, tutorials.
- Social networks
- Books, standards.
52% of the publications on MT are case studies
Applications

Application domains

1998 - 2018

84 case studies
Applications

Application domains

Compiler Validation via Equivalence Modulo Inputs
Vu Le, Mehrdad Afshari, Zhendong Su

2015 IEEE/ACM 37th IEEE International Conference on Software Engineering

Metamorphic Model-based Testing
Applied on NASA DAT
—an experience report
Mikael Lindvall*, Dharmalingam Ganesan†, Ragnar Árdal‡, and Robert E. Wiegand†

Metamorphic Testing for Adobe
Data Analytics Software
Darryl C. Jarman, Zhi Quan Zhou*, Tsong Yueh Chen

DeepTest: Automated Testing of
Deep-Neural-Network-driven Autonomous Cars
Yuchi Tian‡, Kexin Pei*, Suman Jana *, Baishakhi Ray‡
‡University Of Virginia, *Columbia University
Applications

Experimental validations

Cumulative number of programs

Year

Research programs
Real world programs
Applications

Hard to get accepted in top venues

“This is however somehow an incremental work: The use of a metamorphic testing approach has already been shown to be efficient to detect bugs in many situations.”
Challenge

Perform strongly validated applications of MT

- Target relevant domains.
- Use (large) OS and commercial tools when possible.
- Foster industrial applications.
- Real bugs first, mutants second.
Detailed review of challenges and opportunities
Technique development

Topics

- Construction of metamorphic relations: 38%
- Integration with other techniques / New uses: 19%
- Generation of source test cases: 10%
- Survey / Overview: 12%
- Assessment of metamorphic testing: 10%
- Execution of metamorphic test cases: 11%
Performance Metamorphic Testing
Technique development
New uses: Performance Metamorphic Testing

Example: Bookmark all tabs in Firefox

Is 650 ms an acceptable performance?
Is the observed performance acceptable? It depends on:
Typical approaches to assess software performance:

- Human judgement
- Comparison with similar programs

Technique development
New uses: Performance Metamorphic Testing
Technique development
New uses: Performance Metamorphic Testing

\( T(\text{bookmarkAll}(10 \text{ tabs})) \leq T(\text{bookmarkAll}(50 \text{ tabs})) \)
Technique development
New uses: Performance Metamorphic Testing

\[ T(\text{bookmarkAll}(N \text{ tabs})) \leq T(\text{bookmarkAll}(N+M \text{ tabs})) \]
Technique development
New uses: Performance Metamorphic Testing

\[ T(\text{shortestPath}(G,s,d)) \leq T(\text{shortestPath}(G',s,d)) \]
Technique development

New uses: Performance Metamorphic Testing

\[ M(\text{loadImg(largelImage)}) \geq M(\text{loadImage(smallImage)}) \]

https://bugs.chromium.org/p/chromium/issues/detail?id=337425
Technique development
New uses: Performance Metamorphic Testing

\[ E(\text{test}(\text{state}=\text{active})) \geq E(\text{test}(\text{state}=\text{background})) \]

https://zmanim.myjetbrains.com/youtrack/issue/Z-50
Performance Metamorphic Testing: Motivation and Challenges

Sergio Segura, Javier Troya, Amador Durán and Antonio Ruiz-Cortés
Department of Computer Languages and Systems
Universidad de Sevilla, Seville, Spain
{sergiosegura,j.troya,amador.amiz}@us.es

Abstract—Performance testing is a challenging task mainly due to the lack of test oracles, that is, mechanisms to decide whether the performance of a program under a certain workload is either acceptable or poor due to a performance bug. Metamorphic testing enables the generation of test cases in the absence of an oracle by exploiting the relations (so-called metamorphic relations) between the inputs and outputs of multiple executions of the program under test. In the last two decades, metamorphic testing has been successfully used to detect functional faults in a variety of domains, ranging from web services to simulators. However, the applicability of metamorphic testing to detect performance bugs is a topic that remains unexplored.

In this vision paper, we introduce Performance Metamorphic Relations (PMRs) as expected relations between the performance measurements of multiple executions of the program under test. We hypothesize that these relations can be turned into assertions for the automated detection of performance bugs removing the need for complex benchmarks and domain experts’ guidance. As a further benefit, PMRs can be turned into fitness functions to guide search-based techniques on the generation of test data that violate the relations, revealing bugs. This novel idea is motivated with examples and an overview of some of the challenges in this promising topic.

Keywords—Metamorphic testing, performance testing, search-based testing

1. INTRODUCTION

Performance testing aims to reveal programming errors that cause significant performance degradation in the system under test [1], e.g., excessive memory consumption. Performance defects are very common in released software programs. For example, Mozilla developers fix between 5 and 60 performance bugs reported by users every month [2]. Similarly, the emerging trend of mobile applications brings new challenges in terms of performance testing like detecting the infrequent energy leaks or memory leaks [3], [4]. Overall, performance bugs cause poor usability and a waste of resources, which might lead to lots of users or hundred-million-dollar software projects to be abandoned [1], [2].

In contrast to functional bugs, performance bugs do not produce wrong results or crashes in the program under test and therefore they cannot be detected by simply inspecting the program output. For example, suppose that a browser takes 200ms to render a given Web page; is this expected performance? How slow should it be considered as a performance bug? Answering these questions requires not only a good knowledge of the application, but also considering other aspects as the computer hardware or its current workload. Compared to functional faults, performance bugs are significantly harder to detect and require more time and effort to be fixed [1]. This is partly due to the lack of test oracles, that is, mechanisms to decide whether the performance of the program under a certain workload is acceptable, i.e., the oracle problem. Typical oracles in performance testing are human judgment, often involving long discussion among developers, or comparisons among different programs with similar functionality (or different versions of the same program) [1]–[3].

The lack of automated oracles is recognized as one of the key challenges on performance testing. Jin et al. [2] conducted an empirical study of 199 real-world performance bugs and concluded that “techniques that can reliably compare performance numbers across inputs and automatically discover the existence of performance problems are desired”. Liu et al. [3] studied 70 real-world performance bugs in Android applications and reached the same conclusion: “effective performance testing needs automated oracles to judge performance degradation”. Similarly, Nistor et al. [1] analysed 210 performance bugs from three mature open source projects and concluded that “better oracles are needed for discovering performance bugs”.

Metamorphic testing alleviates the oracle problem by providing an alternative when the expected output of a test execution is unknown [5]. Rather than checking the output of an individual program execution, metamorphic testing checks whether multiple executions of the program under test fulfill certain necessary properties called metamorphic relations. For instance, consider the program merge(L1, L2) that merges two lists into a single ordered list. The order of the parameters should not influence the result, which can be expressed as the following metamorphic relation: merge(L1, L2) = merge(L2, L1). A metamorphic relation comprises of a so-called source test case (L1, L2) and one or more follow-up test cases (L3, L4), derived from the source test case. A metamorphic relation can be instantiated into one or more metamorphic tests by using specific input values, e.g., merge([2, 3, 4], [5, 6]) = merge([5, 6], [2, 3]). If the outputs of a source test case and its follow-up test cases(s) violate the metamorphic relation, the metamorphic test is said to have failed, indicating that the program under test contains a bug. In a recent survey, Segura et al. [6] reviewed about 120 papers on metamorphic testing and identified successful or comparisons among different programs with similar functionality [1], [2], [5], which are far from trivial.

Metamorphic testing alleviates the oracle problem by checking whether multiple executions of the PIT fill certain necessary properties called metamorphic relations. For instance, consider the program merge(L1, L2) that merges two ordered lists into a single ordered list. The order of the parameters should not influence the result, which can be expressed as the following metamorphic relation: merge(L1, L2) = merge(L2, L1). A metamorphic relation comprises of a so-called source test case (L1, L2) and one or more follow-up test cases (L3, L4), derived from the source test case. A metamorphic relation can be instantiated into one or more metamorphic tests by using specific input values, e.g., merge([2, 3, 4], [5, 6]) = merge([5, 6], [2, 3]). If the outputs of a source test case and its follow-up test cases(s) violate the metamorphic relation, the metamorphic test is said to have failed, indicating that the program under test contains a bug. In a recent survey, Segura et al. [6] reviewed about 120 papers on metamorphic testing and identified successful or comparisons among different programs with similar functionality [1], [2], [5], which are far from trivial.

Performance Metamorphic Testing: A Proof of Concept

Sergio Segura, Javier Troya, Amador Durán and Antonio Ruiz-Cortés
Department of Computer Languages and Systems
Universidad de Sevilla, Seville, Spain

Abstract—Context. Performance testing is a challenging task mainly due to the lack of test oracles, i.e., mechanisms to decide whether the performance of a program is acceptable or not because of a bug. Metamorphic testing enables the generation of test cases in the absence of an oracle by exploiting the so-called metamorphic relations between the inputs and outputs of multiple executions of the program under test. In the last two decades, metamorphic testing has been successfully used to detect functional faults in different domains. However, its applicability to performance testing remains unexplored.

Objective. We propose the application of metamorphic testing to reveal performance failures.

Method. We define Performance Metamorphic Relations (PMRs) as expected relations between performance measurements of multiple executions of the program under test. These relations can be turned into assertions for the automated detection of performance bugs, removing the need for complex benchmarks and domain experts’ guidance. As a further benefit, PMRs can be turned into fitness functions to guide search-based techniques on the generation of test data.

Results. The feasibility of the approach is illustrated through an experimental proof of concept in the context of the automated analysis of feature models.

Conclusion. The results confirm the potential of metamorphic testing, in combination with search-based techniques, to automate the detection of performance bugs.

Keywords—Metamorphic testing, performance testing, search-based testing

1. Introduction

Performance testing [1] aims to reveal errors that cause significant performance degradation in the program under test (PIT). Performance defects are very common in released software programs. For example, Mozilla developers fix between 5 and 60 performance bugs reported by users every month [2]. Similarly, mobile applications bring new challenges like detecting energy leaks or memory leaks [3], [4].

In contrast to functional bugs, performance bugs do not produce wrong results or crashes in the program under test and therefore they cannot be detected by simply inspecting the program output. For example, suppose that a browser takes 200ms to render a given Web page; is this expected performance? How slow should it be considered as a performance bug? Answering these questions requires not only a good knowledge of the application, but also considering other aspects as the computer hardware or its current workload. Compared to functional faults, performance bugs are significantly harder to detect and require more time and effort to be fixed [1]. This is partly due to the lack of test oracles, that is, mechanisms to decide whether the performance of the program under a certain workload is acceptable, i.e., the oracle problem. Typical oracles in performance testing are human judgment, often involving long discussion among developers, or comparisons among different programs with similar functionality (or different versions of the same program) [1]–[3].

The lack of automated oracles is recognized as one of the key challenges on performance testing. Jin et al. [2] conducted an empirical study of 199 real-world performance bugs and concluded that “techniques that can reliably compare performance numbers across inputs and automatically discover the existence of performance problems are desired”. Liu et al. [3] studied 70 real-world performance bugs in Android applications and reached the same conclusion: “effective performance testing needs automated oracles to judge performance degradation”. Similarly, Nistor et al. [1] analysed 210 performance bugs from three mature open source projects and concluded that “better oracles are needed for discovering performance bugs”.

Metamorphic testing alleviates the oracle problem by providing an alternative when the expected output of a test execution is unknown [5]. Rather than checking the output of an individual program execution, metamorphic testing checks whether multiple executions of the program under test fulfill certain necessary properties called metamorphic relations. For instance, consider the program merge(L1, L2) that merges two lists into a single ordered list. The order of the parameters should not influence the result, which can be expressed as the following metamorphic relation: merge(L1, L2) = merge(L2, L1). A metamorphic relation comprises of a so-called source test case (L1, L2) and one or more follow-up test cases (L3, L4), derived from the source test case. A metamorphic relation can be instantiated into one or more metamorphic tests by using specific input values, e.g., merge([2, 3, 4], [5, 6]) = merge([5, 6], [2, 3]). If the outputs of a source test case and its follow-up test cases(s) violate the metamorphic relation, the metamorphic test is said to have failed, indicating that the program under test contains a bug. In a recent survey, Segura et al. [6] reviewed about 120 papers on metamorphic testing and identified successful or comparisons among different programs with similar functionality [1], [2], [5], which are far from trivial.

Metamorphic testing alleviates the oracle problem by checking whether multiple executions of the PIT fill certain necessary properties called metamorphic relations. For instance, consider the program merge(L1, L2) that merges two ordered lists into a single ordered list. The order of the parameters should not influence the result, which can be expressed as the following metamorphic relation: merge(L1, L2) = merge(L2, L1). A metamorphic relation comprises of a so-called source test case (L1, L2) and one or more follow-up test cases (L3, L4), derived from the source test case. A metamorphic relation can be instantiated into one or more metamorphic tests by using specific input values, e.g., merge([2, 3, 4], [5, 6]) = merge([5, 6], [2, 3]). If the outputs of a source test case and its follow-up test cases(s) violate the metamorphic relation, the metamorphic test is said to have failed, indicating that the program under test contains a bug. In a recent survey, Segura et al. [6] reviewed about 120 papers on metamorphic testing and identified successful
Challenge

Exploit MT beyond functional testing

- Performance testing (time, memory, energy)
- Load testing
- Usability testing
- (...
Metamorphic Relation Patterns
Construction of metamorphic relations
Metamorphic relation patterns

“ICSE Conference”

 searchText = "video"

“ICSE Conference”
type = "video"
duration = "short"
Construction of metamorphic relations

Metamorphic relation patterns
Construction of metamorphic relations
Metamorphic relation patterns

“Nexus”

FindProducts

“A”

“Nexus AND 5X”

FindProducts

“B”

“Nexus AND 5X AND cover”

FindProducts

“C”
Construction of metamorphic relations
Metamorphic relation (output) patterns

If [TBC]

Then the follow-up output should be a subset of the source output
Construction of metamorphic relations

Metamorphic relation (output) patterns

- Equivalence
- Equality
- Subset
- Disjoint
- Complete
- Difference
Construction of metamorphic relations

Metamorphic relation (output) patterns

- Equivalence
- Equality
- Subset
- Disjoint
- Complete
- Difference
Construction of metamorphic relations
Equivalence pattern

q = “winter pentathlon 1949”

15 results

order = “date”

Zero results

Request

Response
200 OK

```
{
    "kind": "youtube#searchListResponse",
    "etag": "\"5g81s4-w52b4VpSn9d6zC5Y-8k/wB8cw2gAkJxKLniupTFwNM6sOII\"",
    "nextPageToken": "CAUQAA",
    "regionCode": "ES",
    "pageInfo": {
        "totalResults": 15,
        "resultsPerPage": 5
    },
    "items": [
        {
            "kind": "youtube#searchResult",
            "etag": "\"5g81s4-w52b4VpSn9d6zC5Y-8k/c1rD28FPVqoHdX25qTP_RB_F0m\"
        }
    ]
}
```

Request
GET https://www.googleapis.com/youtube/v3/search?part=snippet&order=date&q=winter+pentathlon+1949&key=(YOUR_API_KEY)

Response
200 OK

```
{
    "kind": "youtube#searchListResponse",
    "etag": "\"5g81s4-w52b4VpSn9d6zC5Y-8k/01407EC7yUbnsssUgAphQFT10I\"",
    "nextPageToken": "CAUQAA",
    "regionCode": "ES",
    "pageInfo": {
        "totalResults": 14,
        "resultsPerPage": 5
    },
    "items": []
}
```

Empty array!
Construction of metamorphic relations

Equivalence pattern

$q = \text{“redhouse”}$
results_per_page = \text{“20”}

Search album

21 matches (in 2 pages)

$q = \text{“redhouse”}$
results_per_page = \text{“30”}

Search album

27 matches

We've rolled out a fix which should stabilise this, and I ran your examples successfully. Thanks for the detailed report.
Metamorphic Testing of RESTful Web APIs

Sergio Segura, José A. Parejo, Javier Troya, and Antonio Ruiz-Cortés

Abstract—Web Application Programming Interfaces (APIs) allow systems to interact with each other over the network. Modern Web APIs often adhere to the REST architectural style, being referred to as RESTful Web APIs. RESTful Web APIs are decomposed into multiple resources (e.g., a video in the YouTube API) that clients can manipulate through HTTP interactions. Testing Web APIs is critical but challenging due to the difficulty to assess the correctness of API responses, i.e., the oracle problem. Metamorphic testing alleviates the oracle problem by exploiting relations (so-called metamorphic relations) among multiple executions of the program under test. In this paper, we present a metamorphic testing approach for RESTful Web APIs.

Traffic [1]. Many companies are also exposing their existing assets as private APIs, enabling their own developers to build innovative mobile, social, or cloud applications [1,5]. Web APIs are usually compliant with the Representational State Transfer (REST) architectural style, being referred to as RESTful Web APIs [4]. RESTful Web APIs comprise a set of so-called RESTful Web services, where each service implements one or more CRUD operations (create, read, update, delete)

In this paper, we present a metamorphic testing approach for the automated detection of faults in RESTful Web APIs (henceforth referred to as simply Web APIs). We observed that Web APIs have very clear semantics, specified as CRUD operations over resources, and a very consistent use of parameters for standard operations such as filtering, ordering and pagination [2,12-14]. This suggests that...

Challenge

Exploit MRP to guide the construction of MRs

- Identify MRPs in different domains.
- Similar idea used in numerical and machine learning programs (inputs): additive, multiplicative, permutative, invertive, inclusive, exclusive, compositional.
- Assess the effectiveness of MRPs in guiding the construction of metamorphic relations.
**Challenge**

Keep current growing publication trend

- Better quality than quantity.
- Target top conferences and journals.
- Foster international collaborations.
- MET as a key meeting point.

**Challenge**

Promoting MET

- @Authors. keep submitting (good) papers.
- @Organizers. Foster participation, e.g.:
  - Awards (best paper, best presentation, etc.)
  - Sponsored prizes.
  - Invited talks, e.g. authors of journal papers published the previous year.

**Challenge**

Foster dissemination of MT

- YouTube, Wikipedia.
- Webinars, MOOCs, tutorials.
- Social networks
- Books, standards.

**Challenge**

Perform strongly validated applications of MT

- Target relevant domains.
- Use (large) OS and commercial tools when possible.
- Foster industrial applications.
- Real bugs first, mutants second.

**Challenge**

Exploit MT beyond functional testing

- Performance testing (time, memory, energy)
- Load testing
- Usability testing
- (...)

**Challenge**

Exploit MRP to guide the construction of MRs

- Identify MRPs in different domains.
- Similar idea used in numerical and machine-learning programs (inputs): additive, multiplicative, permutative, inverte, inclusive, exclusive, compositional.
- Assess the effectiveness of MRPs in guiding the construction of metamorphic relations.
Thanks!

www.isa.us.es
sergiosegura@us.es