



DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

# **Machine Learning of Bug Pattern Detection Rules**

Nils-Jakob Kunze





DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

# **Machine Learning of Bug Pattern Detection Rules**

## **Maschinelles Lernen von Analyseregeln zur Erkennung von Fehlermustern**

Author:	Nils-Jakob Kunze
Supervisor:	Prof. Dr. Dr. h.c. Manfred Broy
Advisor:	Dr. Elmar Jürgens, Dr. Benjamin Hummel, Dr. Lars Heinemann
Submission Date:	May 15, 2018



I confirm that this master's thesis is my own work and I have documented all sources and material used.

Munich, May 15, 2018

Nils-Jakob Kunze

## Acknowledgments

# Abstract

should for sure contain: - main idea of the thesis - my opinion or point of view - purpose of the thesis - answer to the research question (results!) (- element of surprise -> smth that is interesting and engaging and perhaps not expected) - CLARITY!

# Contents

<b>Acknowledgments</b>	<b>iii</b>
<b>Abstract</b>	<b>iv</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Contribution . . . . .	2
<b>2 Related Work</b>	<b>4</b>
2.1 general approaches for bug detection . . . . .	4
2.2 Section on approaches which learn from the given code . . . . .	4
2.3 Previous work on detecting missing method calls / object usages . . . .	4
2.4 Section on Anomaly Detection . . . . .	5
2.5 Section on Android specific stuff . . . . .	5
<b>3 Detecting Missing Method Calls</b>	<b>6</b>
<b>4 Implementation</b>	<b>7</b>
<b>5 Evaluation</b>	<b>8</b>
<b>6 Conclusion</b>	<b>9</b>
<b>List of Figures</b>	<b>10</b>
<b>List of Tables</b>	<b>11</b>
<b>Bibliography</b>	<b>12</b>

# 1 Introduction

In bigger software systems often a significant amount of work is outsourced to existing libraries or software frameworks, which expose their functionality through an application programming interface (API). While those APIs are usually designed to be easy to use, there is always a trade-off between usability and flexibility. To enable the users of a more complex library to use it in the way most appropriate for their specific use case, their API cannot remain trivial. In the case of large frameworks or very powerful libraries, often a lot of knowledge is required to invoke the API in the correct way. Often there are constraints or requirements which might not be clear from the outset, but which have to be heeded in order to avoid serious bugs or complications. In the worst case, this knowledge might not even be explicitly documented.

USAGE PATTERNS while there might be a manifold of correct ways to use an API, often there are underlying patterns which all correct usages of the api follow one or more of those patterns such patterns are also called api usage patterns (ref overview paper) if one deviates from those patterns during the usage of an api there are usually problems, also known as api misuse(?) while one can imagine many different types of api missusages (eg related to parameter choice, method order, ...) In this work we focus on one specific type of api usage problem, missing method calls

Some stuff about real life situation where missing method call can be a problem -> find different bug report then the one the Majority Rule papers uses, but something like that

[Example] bugs related to missing method calls can be found all over the internet the issues they cause range from runtime exceptions to problems in limit cases but generally reveal a code smell

## 1.1 Motivation

Consider for instance the following stack overflow post: Here a developer spent many hours of their valuable time on debugging a relatively simple problem related to just one method call which was missing.

However, developer not only spent time during development on bugs related to missing method calls, these kind of bugs also survive development and get checked into the code repository where they cause problems in the future. In their extensive analysis

of the Eclipse bug repository Monperrus et al. [MM13] show that even in mature code bases there are many bugs related to missing method calls. Together this makes it highly desirable to be able to automatically detect missing method calls in production code, not only to save expensive developer time, but also to make maintenance cheaper and easier.

A simple and straightforward approach for this would be to build a set of hard-coded rules regarding method calls, such as for example:

- “always call `setControl()` after instantiating a `TextView`”
- “in Method `onCreate()` of classes extending `AppCompatActivity` always call `setContentView()`”
- “when calling `foo()` also call `bar()`”

Well crafted and thought-out rules like this could facilitate a very high precision in detecting missing method calls, however, creating and maintaining such a rules list would require a tremendous effort. While such work might be justified for large and important libraries, the necessary effort would also grow with the size of the library until it becomes completely infeasible.

To circumvent this problem, we would like to automatically detect locations in a code base where a method call is potentially missing without needing any further input. Such an approach would adapt to a changing system without requiring additional work of a developer and could also be applied to proprietary code which is not open to the public. While the discovered locations will probably not be 100% accurate, they could then be examined by an expert who would determine the severity of the finding and issue a fix if necessary.

-> express some more much better than fixed preprogrammed rules, can adapt to changing system, be specific for own not open library, etc

the approach chosen in this work bases of recommender systems / learning (Mention the ideas of [Bru12], chapter 2 as an inspiration / the way to the idea - maybe) first find likely recommendations, for writing, then realize, if something is super likely given a particular situation, but it is not there, it seems like a good indicator of an error

## 1.2 Contribution

In this thesis we present a thorough reevaluation of the type usage characterization first introduced by Monperrus et al.[MBM10] and further refined in a follow up publication [MM13] type usage: list of method calls invoked on variable of a given type which occur in the body of a specific method Majority rules idea: if a type is used in one



particular way many many times and differently only one (or few) times, this probably indicates a bug

Further evaluation of the concept with comparison test using different similarity measures, application to big data set of open source android applications, . . .

FINALLY: Summary of the other chapters of this thesis

## 2 Related Work

What is the general state of the art (maybe extra section?) Why is this potentially better than other approaches (actually learning an api vs static rules comes to mind)

How to split this in the best way? -> check interesting related work and split it by topic / used technique / goal

General: read the summary paper [Rob+13] again and check which approaches are there and which could / should be mentioned + check to read stuff. . .

### 2.1 general approaches for bug detection

static stuff, rules based, etc mention findbugs?

### 2.2 Section on approaches which learn from the given code

also mention some stuff about recommender systems in general (it is the official thesis topic after all. . .)

Mention [Eng+01] as probably the first paper which proposed the general idea behind DMMC -> learning from the code at hand, instead of using static predefined rules

Remember the general machine learning approaches which were not super successful, but at least learned a LOT especially like general code smells for example

### 2.3 Previous work on detecting missing method calls / object usages

short summaries of the two DMMC papers: [MBM10] and [MM13]

mention the related work from detecting mmc paper

The paper on graph based object using pattern mining seems also super related [Ngu+09] downside: relatively complicated and high false positive rate

This [WZL07] goes in the direction of detecting method order and seems quite interesting quite old, but has implementation and also an interesting related work section!

## **2.4 Section on Anomaly Detection**

## **2.5 Section on Android specific stuff**

### 3 Detecting Missing Method Calls

trennen: previous work + what i did -> move their stuff to related work? theoretical explanation of the different approaches Majority Rule - intuition (restaurant example) + formalized

my work: naive bayesian learner as baseline? clustering approach (hypersphere) something as of now unknown from anomaly detection working with the inheritance hierarchy! per class type usages wrong method call, superfluous method

— generally: (in this section or implementation? or somewhere totally else or not at all?)

3 main parts: Feature Extraction, Learning, Results - for each tons of different options and combinations FE: what are the features, restricted to method calls?, granularity? (including parameters?) method order?

Learning: Struktur finden und anomalien ermitteln consider what the expected structure MEANS (eg bruch expects a kind of uniformity to object usages, which is probably mostly present in GUI etc Frameworks) maybe some method can also detect multiple missing calls, order, ...

Results: how is the "quality", under which circumstances? some kind of statement about the initial assumptions behind idea and

## 4 Implementation

details about implementation and pitfalls that had to be overcome why some solutions were discarded (eg pure database / could be revisited if it turns out to be the best anyways - performance)

explaining all the work i did first reading their code, coming across a couple of discrepancies between code and paper, later check if everthing still works (evaluation) refactoring everything + saving stuff to database building python infrastructure for analysis

details on the different detectors I implemented

## 5 Evaluation

Which questions are we trying to answer with the evaluation? qualitative evaluation: are the findings useful in practice? quantitative evaluation: given some assumptions, how useful can we expect this technique to be? make sure I will be able to a) answer those questions and b) the answers are “interesting”

general idea behind evaluation method and what we are trying to detect actually it's a sort of simulation (see recommender system book p. 301f) - micro vs macro evaluation, ...imitation of the real system of software development using a much simpler system, namely dropping method calls obvious question: how similar are the such created mmcs to mmcs in the wild? Robustness explanation wie viel brauche ich um sinnvoll viel zu lernen - hälfte der leute machen fehler -> wie robust brauchst du die daten / ist das verfahren ! what is needed for it to work (lines of code, feature richness, etc) problem: how to evaluate when it is "working" vs not working -> again simulate "missing methods", compare performance? Metrics explanation - Precision, Recall additional metrics from recommender systems book p.245f (robustness, learning rate - p. 261) performance -> training time + performance when working

checking if their “mistakes” / inaccuracies make a difference or not

(Case Study - if time...) ANDROID?! would be super sexy to apply the best method to some library / framework and see what happens

subsection: threats to validity

qualitative evaluation: pick a couple of high scoring findings and explain the failure modes (eg doesn't take branching into account, etc)

explain problems with the automatic evaluation how related are the degraded TUs to missing method calls you can find in the wild? improving the metrics by dropping cases where we know we won't find an answer ...

## 6 Conclusion

new method is amazing and way better in case xy but worse in zz robustness has shown this and that future research would be interesting in the direction of ...

contribution: Datenset generiert, zumindest was über software sturktur lernen, bla !

future research: apply this anomaly detection method to other features (implements, overrides, pairs of methods, ...) Reihenfolge von Methoden - hat gut funktioniert und wäre potentiell nützlich, aber kleines subset order of method calls? (probably should wait for empirical results - how long are typical method lists (state explosion, ...) / latice solution?) efficient update (only update files touched by change + the scores for affected type usages) - maybe database view for scores will already do this automatically? higher precision by some means? (clustering, etc) Performnance (shouldn't be a problem?)

## List of Figures



## List of Tables

# Bibliography

- [Bru12] M. Bruch. “IDE 2.0: Leveraging the wisdom of the software engineering crowds.” PhD thesis. Technische Universität, 2012.
- [Eng+01] D. Engler, D. Y. Chen, S. Hallem, A. Chou, and B. Chelf. “Bugs as deviant behavior: A general approach to inferring errors in systems code.” In: *ACM SIGOPS Operating Systems Review*. Vol. 35. 5. ACM. 2001, pp. 57–72.
- [MBM10] M. Monperrus, M. Bruch, and M. Mezini. “Detecting missing method calls in object-oriented software.” In: *European Conference on Object-Oriented Programming*. Springer. 2010, pp. 2–25.
- [MM13] M. Monperrus and M. Mezini. “Detecting missing method calls as violations of the majority rule.” In: *ACM Transactions on Software Engineering and Methodology (TOSEM)* 22.1 (2013), p. 7.
- [Ngu+09] T. T. Nguyen, H. A. Nguyen, N. H. Pham, J. M. Al-Kofahi, and T. N. Nguyen. “Graph-based mining of multiple object usage patterns.” In: *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. ACM. 2009, pp. 383–392.
- [Rob+13] M. P. Robillard, E. Bodden, D. Kawrykow, M. Mezini, and T. Ratchford. “Automated API property inference techniques.” In: *IEEE Transactions on Software Engineering* 39.5 (2013), pp. 613–637.
- [WZL07] A. Wasylkowski, A. Zeller, and C. Lindig. “Detecting object usage anomalies.” In: *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. ACM. 2007, pp. 35–44.