

# Query Construction Patterns in PHP

---

David Anderson and Mark Hills

24th IEEE International Conference on Software Analysis, Evolution, and  
Reengineering (SANER 2017), ERA Track

February 21-24, 2017

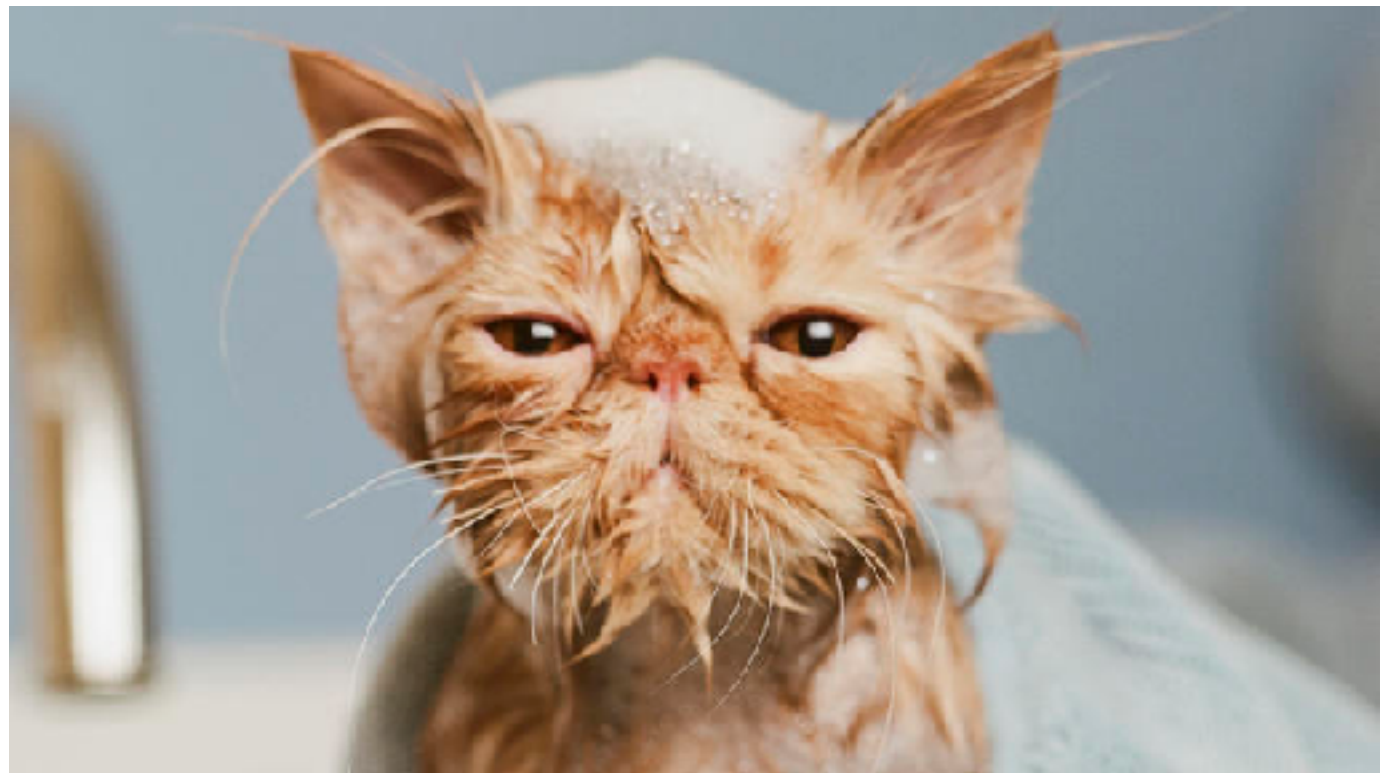
Klagenfurt, Austria



<http://www.rascal-mpl.org>



# Obligatory Cute Kitties!



# Context: PHP and MySQL

---



- Original MySQL API introduced in PHP 2
  - Widely used
  - No support for object-oriented language features
  - No support for prepared statements, stored procedures
- Deprecated in PHP 5.5, dropped in PHP 7

# Our goal: program transformation!

---



- We want to replace uses of MySQL API with either MySQL Improved (mysqli) or PDO, based on user preference
- Want to move more towards features like prepared statements, provides better protection against SQL injection vulnerabilities (like this one!):

```
$query = mysql_query("
SELECT title
FROM semesters
WHERE semesterid = $_POST[semester]
");
```

# So, what's the problem?

---

- Safe transformations challenging in PHP!
- Dynamic features, inclusion model, heavy use of strings and implicit type coercions all make this harder
- So, focus our efforts:
  - Can we exploit common usage patterns?
  - What additional analysis do we need?
  - Do we hit a point of diminishing returns? Where?





# Query construction patterns

---



- How are queries typically built in PHP scripts?
- What parts of a query tend to be dynamic?
- What features are used to build these dynamic query parts?



# Corpus & methodology

---

- Starting small, see paper for details...
- Analysis scripted in Rascal for reproducibility



# Which patterns were found?

---

- Literal query strings (QCP-1)
- Cascading concatenating assignments (QCP-2)
- Assignments distributed over control flow (QCP-3)
- Dynamic query strings (QCP-4)





# How often did they appear?

---

- Literal strings: surprisingly often
- Dynamic queries: most common
  - Most dynamic pieces are variables or array lookups
  - Several are function calls or ternary operations, but comparatively few
  - Almost all used as parameters
- Other two: not very common (may mean patterns are too specific!)



# What does this mean?

---

- Assuming reasonable queries being built (an assumption we are still validating), results are encouraging — many dynamic parts used as parameters, others often variables or arrays versus unusual features
- But, need more powerful analysis in general, especially to ensure soundness for transformations

# Threats to validity

---

- Results could be very specific to the selected systems
- Some systems older, no longer maintained



# Threats to validity

---

- Results could be very specific to the selected systems
- Some systems older, no longer maintained
- Mitigation: focus on evolution, so older systems are normal; have included a variety of systems
- Still an issue, though: a more extensive evaluation is ongoing



# What have we learned? What's left?

---



- Queries appear to be built in predictable patterns
- Dynamic parts are mainly in the “right” places, making a transformation to prepared statements possible
- We need a more extensive analysis with more systems and more precise and sound analysis algorithms
- We need better models of the queries themselves (current work) for more precise pattern identification
- We need to build the transformation!

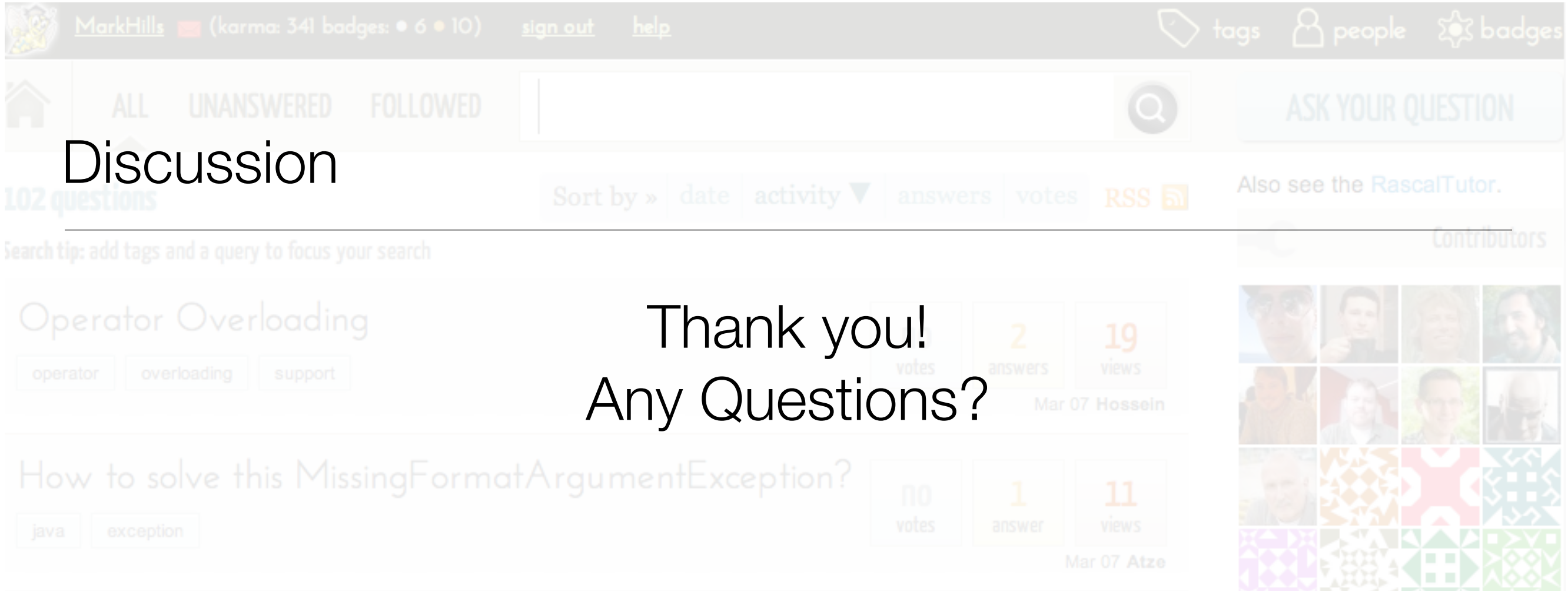
# Questions for the audience

---



- What about our results is unexpected, or would maybe be invalidated by a more extensive analysis?
- What analysis are we missing? What should we add?
- Do you think newer systems would show much different results?
- Can you think of other applications (e.g., program comprehension) that we could apply this to?





- Rascal: <http://www.rascal-mpl.org>
- Me: <http://www.cs.ecu.edu/hillsma>