PHP AiR: Analyzing PHP Systems with Rascal

Mark Hills and Paul Klint

CSMR/WCRE Software Evolution Week 2014
February 5, 2014
Antwerp, Belgium

http://www.rascal-mpl.org
Why look at PHP applications?

- Popular with programmers: #6 on TIOBE Programming Community Index, behind C, Java, Objective-C, C++, and C#, and 6th most popular language on GitHub

- Used by 78.8% of all websites whose server-side language can be determined, used in sites such as Facebook, Hyves, Wikipedia

- Big projects (MediaWiki 1.19.1 > 846k lines of PHP), wide range of programming skills, very limited tool support

- Hostile environments: most PHP code runs on the web
What are we trying to do?

• Big picture: develop a framework for PHP analysis

• Specifics:
  • Empirical software engineering
  • Software metrics
  • Program analysis (static/dynamic)
  • Developer tool support
Rascal to the Rescue!

• “Rascal is a domain specific language for source code analysis and manipulation a.k.a. meta-programming.” (http://www.rascal-mpl.org/)

• Language focus: program analysis, program transformation, domain-specific language creation

• Current projects across large numbers of domains, both within and outside academia

• Open source, over 30 committers worldwide
Why Rascal?

• Built-in language support for matching & transforming code

• Rich data types: relations, maps, lists, sets, tuples, parse trees, higher-order functions

• Console supports interactive exploration

• Extensible with Java and Eclipse

• Empirical research support: code querying, statistical analysis, interaction with external data (e.g., code repositories, external databases), visualization
Design Decisions

• Parsing: roll our own, or use existing parsers?

• Where should we optimize?
  • Inside PHP AiR?
  • Inside Rascal?
  • Both?

• How do we cleanly access external data sources that hold analysis data we care about?
Result: PHP AiR (Analysis in Rascal)
One Example: Empirical Study of PHP Feature Usage

• Perspective: Creators of program analysis tools

• What does a typical PHP program look like?

• What features of PHP do people really use?

• How often are dynamic features, which are hard for static analysis to handle, used in real programs?

• When dynamic features appear, are they really dynamic? Or are they used in static ways?
Lessons Learned

• Rascal data types and declarative programming lead to smaller, more expressive code

• Having source locations as a built-in datatype provides a powerful abstraction for referencing code

• Tool flexibility is important: an all or nothing approach to Rascal would slow us down (e.g., parsing)

• Scripting analyses eases reproducibility

• Performance is a persistent issue, and needs more work
Related Work (PHP Frameworks)

• PHP-sat & PHP-tools

• PHP CodeSniffer (standards conformance)

• PHP Copy/Paste Detector (only exact copies)

• PHPDepend, PHPLoc (metrics)

• PHPMD (metrics, simple bugs)

• php, HipHop (analysis & compilation)
Demo: PHP AiR

The technology demo

The software isn’t 100% complete.

If it had a user interface you would see something here... here... and sometimes here.

And then you’d be saying, “I gotta get me some of that.”

Any questions?
Thank you!
Any Questions?

- Rascal: http://www.rascal-mpl.org

- SWAT: http://www.cwi.nl/sen1

- Me: http://www.cwi.nl/~hills