

CWI

Centrum Wiskunde & Informatica

A Framework for PHP Program Analysis

Mark Hills

Postdoc in Software Analysis and Transformation (SWAT)

CWI Scientific Meeting

February 8, 2013



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

Overview

- Motivation
- Goals
- Current Progress
- Related Work



PHP

TRAINING WHEELS WITHOUT THE BIKE

PHP: Not Always Loved and Respected



- Created in 1994 as a set of tools to maintain personal home pages
- Major language evolution since: now an OO language with a number of useful libraries, focused on building web pages
- Growing pains: some “ease of use” features recognized as bad and deprecated, others questionable but still around
- Attracts articles with names like “PHP: a fractal of bad design” and “PHP Sucks, But It Doesn’t Matter”

So Why Focus on PHP?

- 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: big opportunities for program analysis to make a positive impact

Rascal: A Meta-Programming One-Stop-Shop



- Context: wide variety of programming languages (including dialects) and meta-programming tasks
- Typical solution: many different tools, lots of glue code
- Instead, we want this all in one language, i.e., the “one-stop-shop”
- Rascal: domain specific language for program analysis, program transformation, DSL creation

PHP Program Analysis Goals

- Build a Rascal framework for creating PHP program analysis tools
- Build a number of standard program analysis “passes”: type inference, alias analysis, etc
- Use this to experiment with more advanced PHP analysis tools and algorithms, e.g., to support code refactoring, security analysis, detection of problems caused by language changes
- Integrate all this with standard IDE tools (especially Eclipse)



What have we done so far?

- Built a number of standard tools for manipulating PHP programs
- Built basic analysis infrastructure (e.g., control flow graphs)
- Built a PHP-specific analysis for resolving dynamic file includes
- Studied actual PHP code to see what people are really doing with the language

Empirical Analysis of PHP Programs



- 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 these features do appear, are they really dynamic? Or are they used in static ways?

Experimental Setup: The Corpus

- 19 open-source PHP systems
- 3.37 million lines of PHP code
- Well-known systems: WordPress, Joomla, MediaWiki, MediaWiki
- Multiple domains: app frameworks, CMS, blogging, wikis, eCommerce, webmail, and others

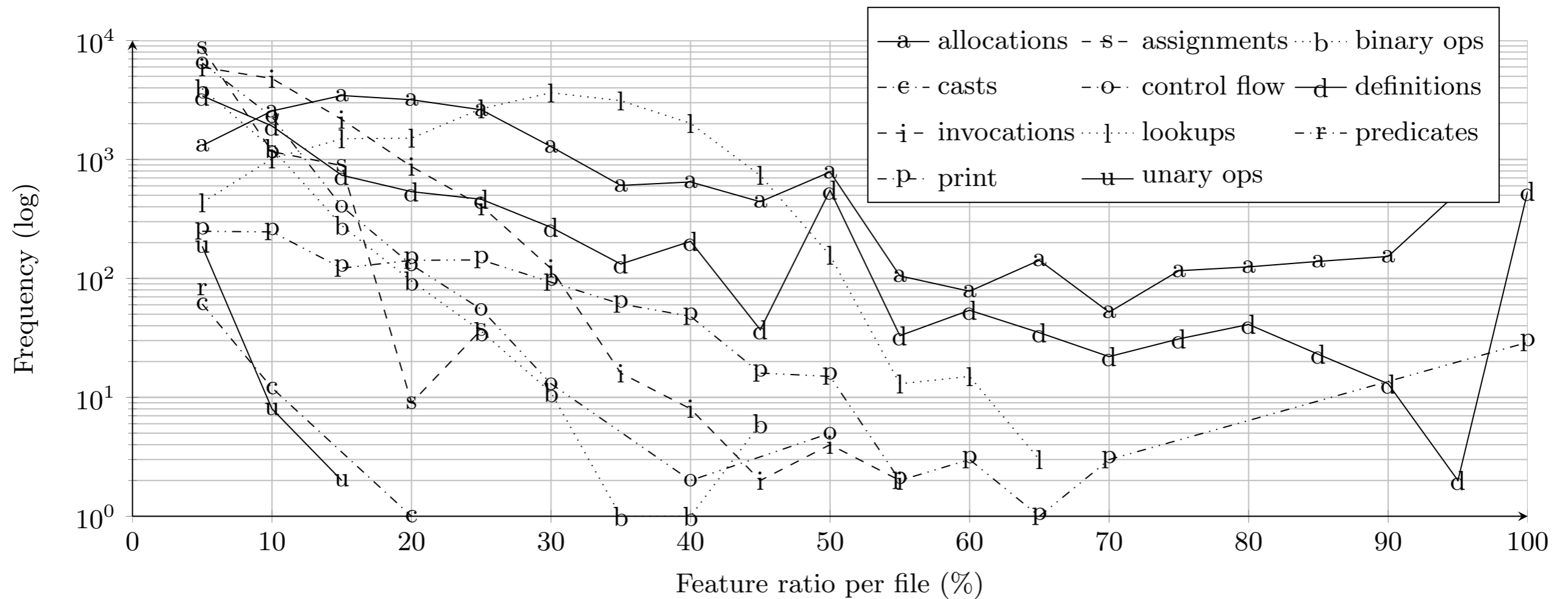


Prototyping Empirical Analyses with Rascal



- Used Rascal for all analysis steps, all computation, and generation of results in LaTeX
- Pattern matching gives feature usage counts
- More complex patterns give uses of dynamic features
- Interaction allows inspection and refinement
- String templates allow generation of LaTeX for tables and figures

Example: Feature Distribution



Example: Occurrences of Dynamic File Includes

System	Includes			Files	Gini
	Total	Dynamic	Resolved		
CakePHP	124	120	91	640(19)	0.28
CodeIgniter	69	69	28	147(20)	0.44
DoctrineORM	56	54	36	501(14)	0.19
Drupal	172	171	130	268(16)	0.42
Gallery	44	39	25	505(10)	0.26
Joomla	354	352	200	1,481(122)	0.17
Kohana	52	48	4	432(18)	0.55
MediaWiki	554	493	425	1,480(38)	0.34
Moodle	7,744	4,291	3,350	5,367(504)	0.39
osCommerce	683	539	497	529(22)	0.28
PEAR	211	11	0	74(9)	0.14
phpBB	404	404	313	269(51)	0.34
phpMyAdmin	819	52	15	341(27)	0.23
SilverStripe	373	56	27	514(10)	0.34
Smarty	38	36	25	126(7)	0.29
SquirrelMail	426	422	406	276(13)	0.14
Symfony	96	95	41	2,137(40)	0.22
WordPress	589	360	332	387(17)	0.32
ZendFramework	12,829	350	285	4,342(42)	0.29

Summary of Findings

- Most files are smaller than 1300 lines of code
- Of 109 total features, 7 are never used; there is no detectable “core”
- Supporting 74 features in an analysis would cover 80% of the files
- Many dynamic includes are static in practice
- Many variable variables use a statically detectable set of names
- Eval truly is dynamic

Why do all this?

- Prototypes can be built to cover a subset of the language and still cover a significant number of real program files
- Knowledge of how often dynamic features appear provides firmer ground in building realistic analysis algorithms
- Patterns of dynamic feature usage can be exploited in analysis tools to improve precision, mitigate against dynamic effects

Related Work

- Richards *et al.*: dynamic analysis of JavaScript
- Meawad *et al.*: transformation of JavaScript to remove eval
- Furr *et al.*: dynamic analysis of Ruby to replace dynamic features with easier to analyze static features
- Collberg *et al.*: static study of Java bytecode for counts and distributions of various language metrics


Thank you! Any Questions?

102 questions Search tip: add tags and a query to focus your search

Operator Overloading
operator overloading support
no votes 2 answers 19 views
Mar 07 Hossein

How to solve this MissingFormatArgumentException?
java exception
no votes 1 answer 11 views
Mar 07 Atze

Also see the RascalTutor.
Contributors



- Rascal: <http://www.rascal-mpl.org>
- SWAT: <http://www.cwi.nl/sen1>
- Me: <http://www.cwi.nl/~hills>