Navigating the WordPress Plugin Landscape

Mark Hills

24th IEEE International Conference on Program Comprehension
May 16-17, 2016
Austin, Texas, USA

http://www.rascal-mpl.org

Background: WordPress

- Extremely popular blogging/CMS platform
  - 23.3% of top 10 million websites run WordPress
  - 50% of all CMS sites run WordPress
  - WordPress runs roughly 25% of all websites
- Written in PHP, has supported *plugins* since version 1.2

Note: figures from early February 2016

Background: plugins

- Plugin: component written in PHP, bundled with needed resources (HTML, CSS, JavaScript, PHP libraries, images, etc)
- Plugins use the WordPress Plugin API
  - hooks, filters, and actions (our focus!), see here for a list: http://adambrown.info/p/wp_hooks/
  - others: shortcodes, custom meta-data, configuration options
- 54,512 plugins in official repository as of late September 2015, not all maintained

Note: icon from Jetpack plugin page, https://wordpress.org/plugins/jetpack/

Background: hooks, filters and actions

- *Hooks* are named events, triggered by API calls
  - *Actions* are used to respond to system events (e.g., logging in)
  - *Filters* are used to respond to input/output operations (e.g., displaying part of a page, loading/saving database records)
- Plugins register a *handler* for the hook, called by WordPress when the related event occurs
- Note: Plugins can create their own hooks, we focus on those defined by WordPress here

Research questions

- Q1: How has the hook mechanism grown, and how do developers use it in their plugins?
- Q2: How can we help developers to find the hooks they need to use in their own plugins?
- Q3: How can we link specific hooks to implemented handlers in popular plugins to provide easier access to sample code?

Our corpus

- Started with all plugins in official repository
- Filtered based on supported WordPress version (at least 4.0, latest in corpus 4.3.1, current 4.5.2) and last update date (in 2015)
- Stats on remaining plugins:
  - 12,860 plugins
  - 176,294 PHP files
  - 27,580,638 lines of PHP code

Methodology

- Corpus parsed with an open-source PHP parser
- All analysis scripted using Rascal and the PHP AiR framework
- Plugin filtering performed using regular expression matching over HTML pages for each plugin; script allows full checkout of matching corpus for replicating results
- All code available at https://github.com/ecu-sle-lab/wp-plugin-analysis

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

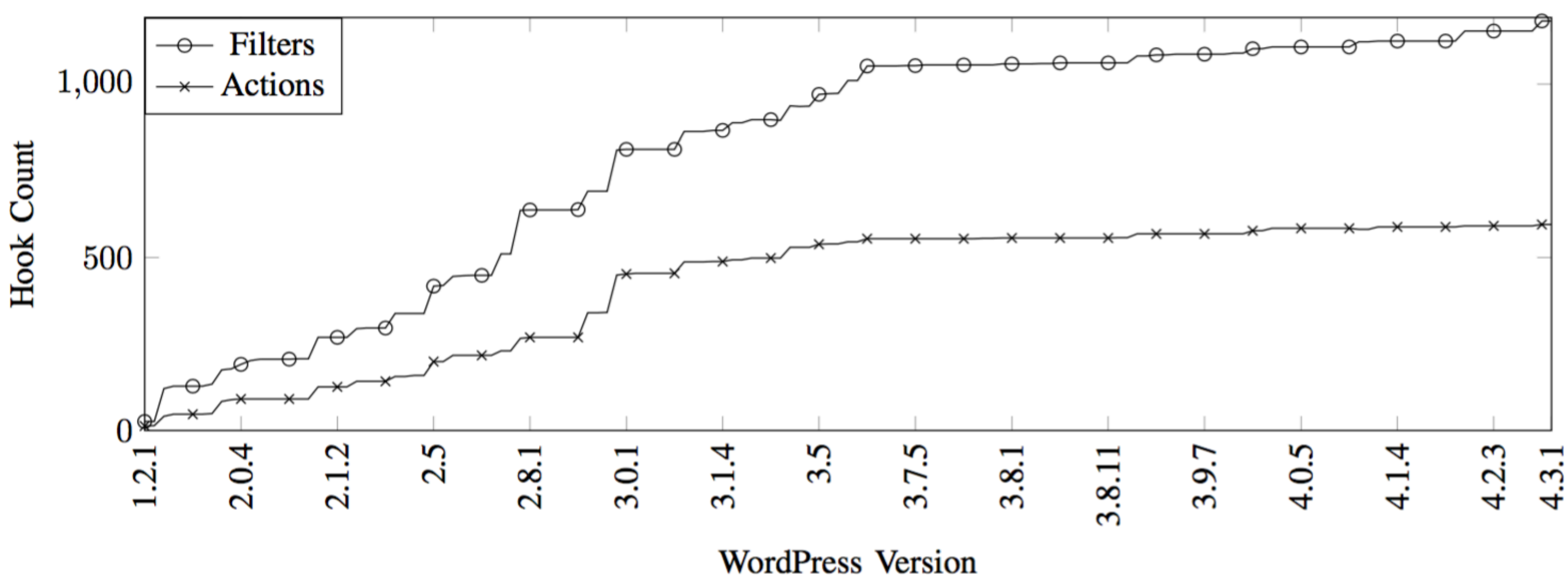
Q1: Plugin hook (filter and action) usage

- Q1.1: How has the number of hooks for filters and actions grown over time?
- Q1.2: How many hooks does a typical plugin provide handlers for?
- Q1.3: Which hooks are the most popular? Which are the least popular?

Q1.1: Results

- How has the number of hooks for filters and actions grown over time?

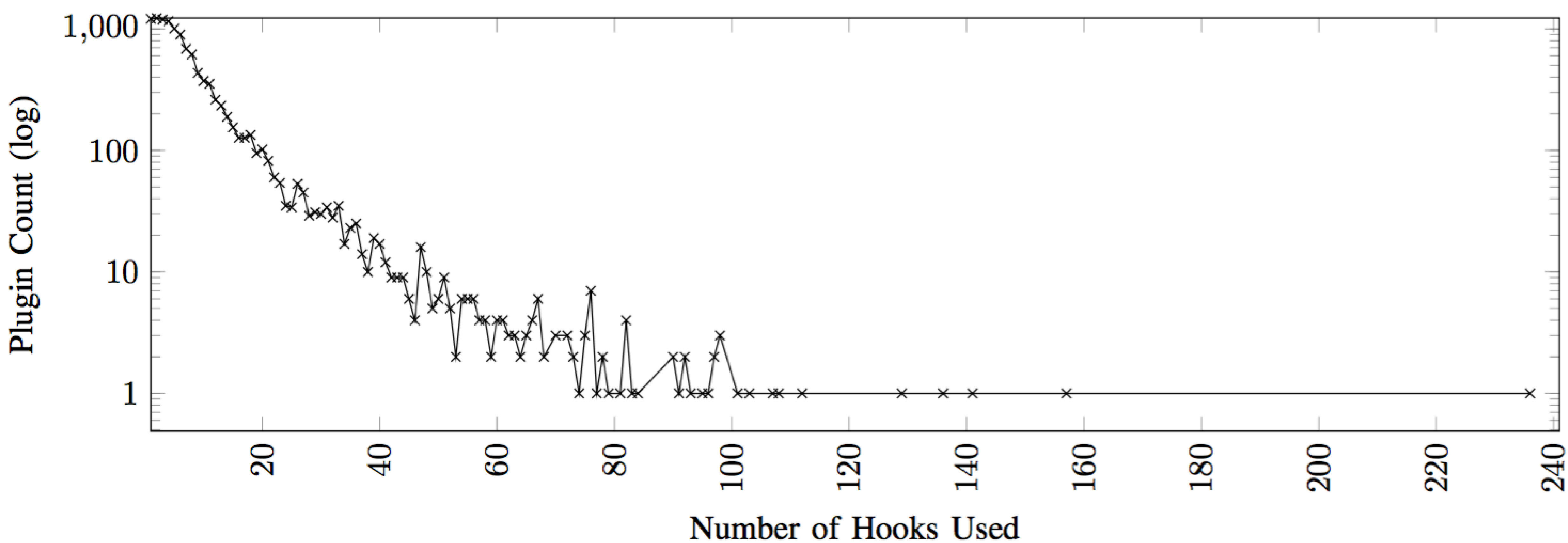
  Filters are more popular than actions; both have grown over time, but this growth appears to be slowing (see Figure 6 in paper); WordPress 4.3.1 has 1,182 hooks for filters and 595 for actions



Q1.2: Results

- How many hooks does a typical plugin provide handlers for?

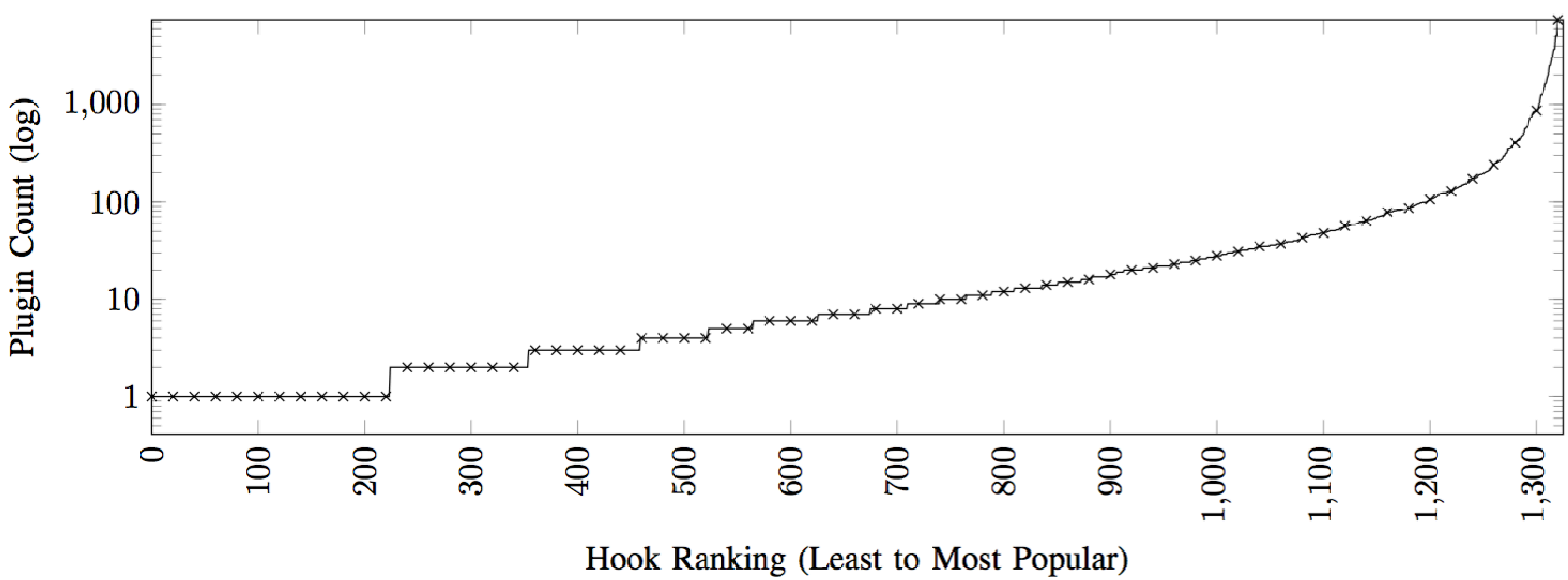
  Most use very few: 1,210 use only 1, half use at most 6, only 6 use 50, very few use more



Q1.3: Results

- Which hooks are the most popular? Which are the least popular?

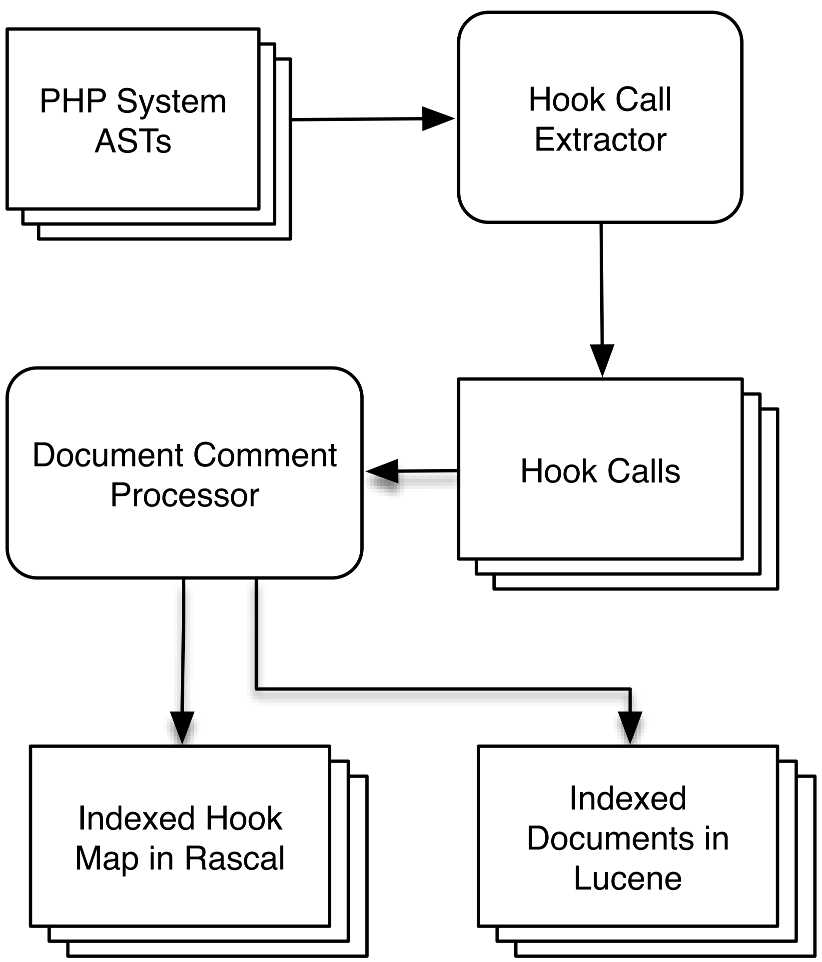
  453 hooks never implemented, 224 used by only 1 plugin, 765 by 10 or fewer; most used are very common, `admin_menu` used by 7,377 plugins (allows plugins to extend the admin menu in WordPress)

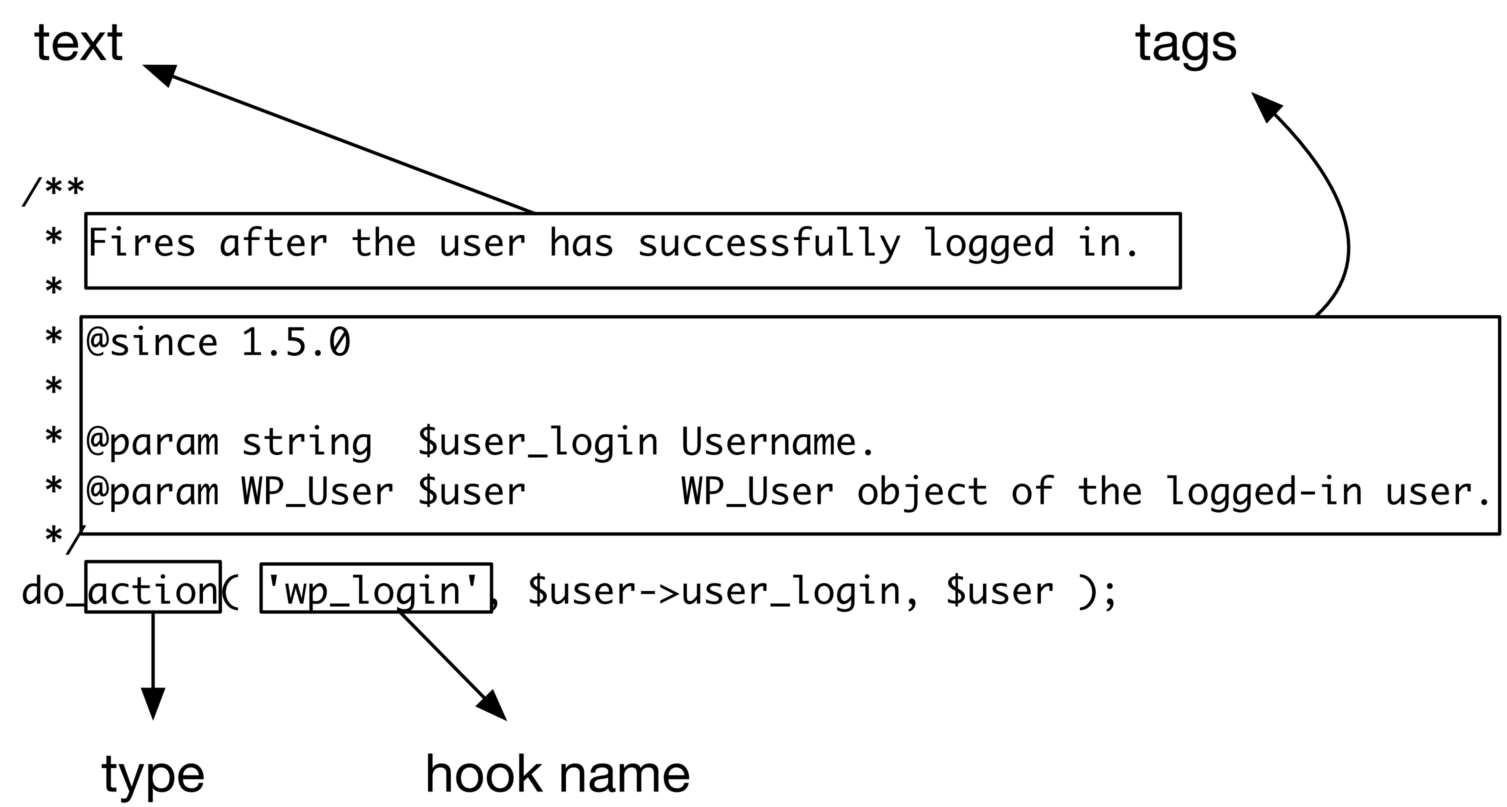


Q2 & Q3: Finding hooks, linking to handlers

- Core idea: use text search to find hooks of interest, then identify matching pairs of hook call/registration functions, then link handler callables in registrations to actual implementations
- Challenge: there are thousands of plugins, need a way to do this where we don't need to install each one for analysis
- Solution: extract *summaries* of each plugin, each version of WordPress, perform linking using summaries

Step 1: Text search for hooks





Step 2: Linking extension points to plugins

- Linking relation built between each plugin and most recent version of WordPress plugin supports
- Needs to support potential matches, since hook names may be computed instead of given as string literals

```
// WordPress 4.2.4
apply_filters( "get_{$meta_type}_metadata", null,
  $object_id, $meta_key, $single )

// Responsive Nagivation plugin
add_filter( 'get_post_metadata',
  array( 'cmb_Meta_Box_ajax', 'hijack_oembed_cache_get' ),
  10, 3 )
```

Step 2: Linking extension points to plugins

- Linking relation built between each plugin and most recent version of WordPress plugin supports
- Needs to support potential matches, since hook names may be computed instead of given as string literals
  - Hook names in WP generate regular expressions, hook names in plugins generate strings to match against
  - Linking given as regex matching, patterns and strings with more static portions weighted most heavily

Step 3: Linking registrations to implementations

- Linking relation built from each handler registration to handler, based on callable

- Rules given in paper

  - Essentially a variant of a call graph construction algorithm

  - Falls back to using name models for ranked matches against function or method names where needed

Putting it all together

- Initial search winnows the list of possible hooks, based on the user's search terms.
- This allows us to link from the selected hooks to registrations of handlers…
- …and then from registrations of handlers to the handlers themselves.
- These are then ranked in order of popularity (based on numbers maintained by WordPress) of the containing plugin.

Threats to validity

- Name computation has to deal with dynamic (computed) names, means we could be under- or over-counting the total number of hooks; most popular hooks use static or very specific dynamic names, so very little effect on resulting numbers

- Analysis attempts to be useful, but not sound or complete, could make false links or miss actual links; low quality links dropped to avoid false links, most matches very specific, empirical numbers indicate this is fairly accurate

- Changes to the corpus could yield different results

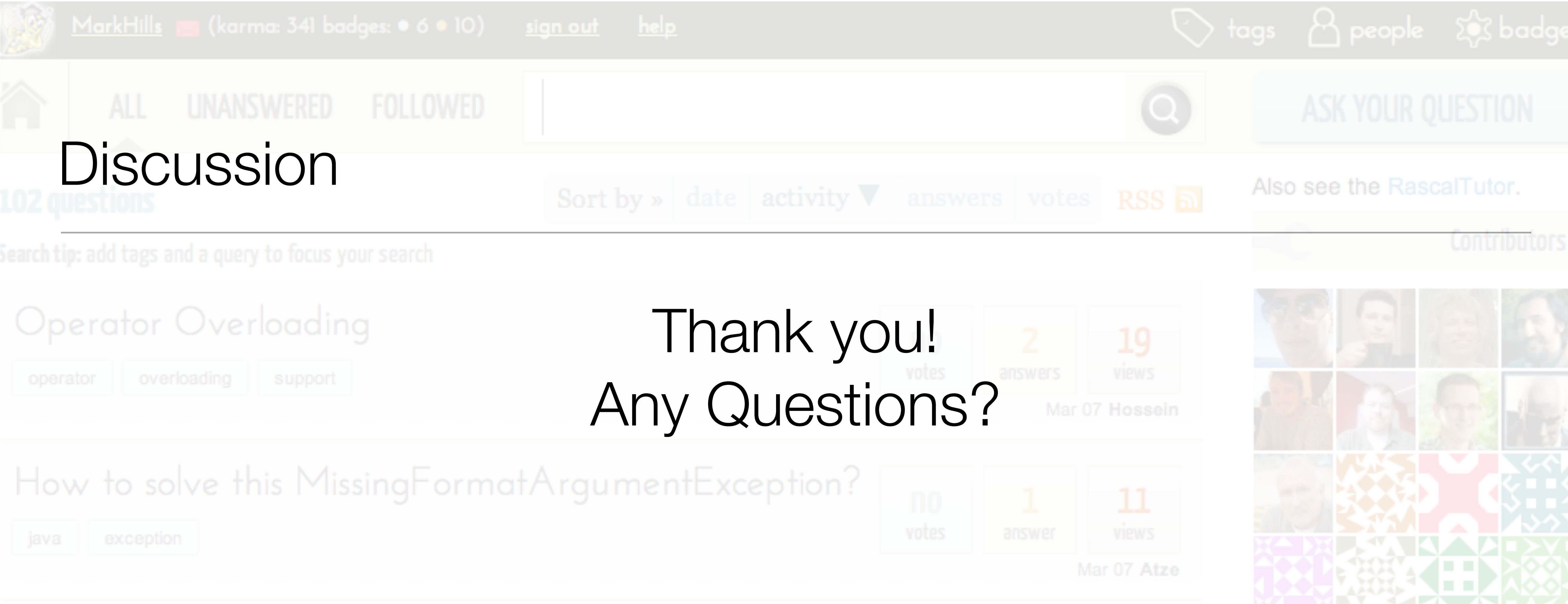
Summary

- We've presented a combination of text search and static analysis to find relevant hooks and link these hooks to actual handler implementations
- We've presented empirical results about how hooks are used in actual plugins, how the number of hooks has changed over time
- These empirical results indicate that the analysis is useful, even if it is not sound or complete

Future Work

- What is left to do?
  - Tool support
  - Developer studies
  - Expansions of empirical study
  - Enhancement of text search into more general code search

Discussion



Thank you!

Any Questions?

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

East Carolina University