The Bug That Cried Wolf

Finding trends in statically checked bugs over time

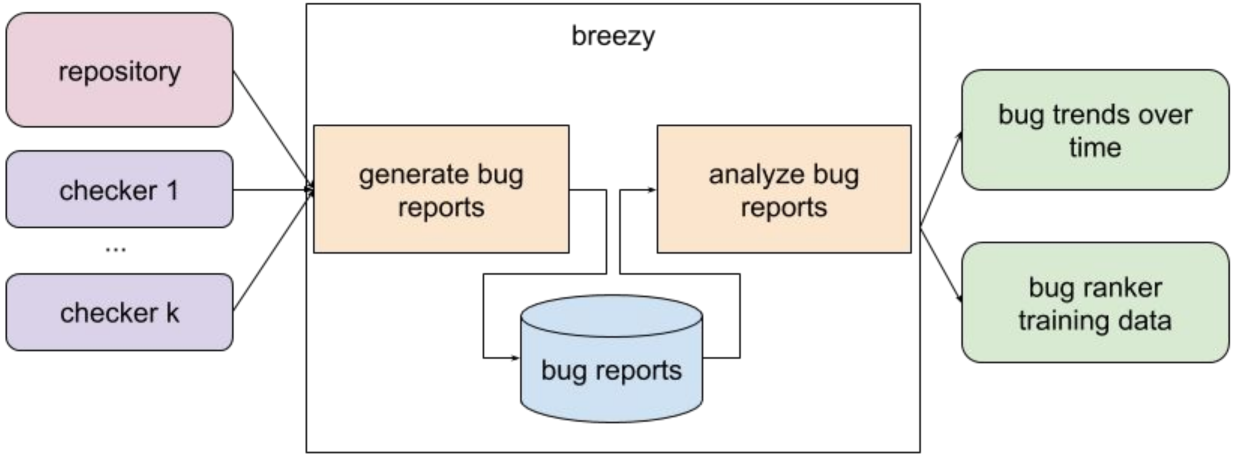
Edward Chen, Kevin Yu, Arvind Saripalli

Motivation

- Static bug analysis bug tools are notorious for creating false positive results. False positives can deter developers from relying on these tools, reducing their effectiveness.
- To better understand the effectiveness of bugs found using static checkers, especially in large repository involving multiple contributors, we built a tool that tracks the inception and lifespan of bugs by combining existing static analysis tools with version control systems.
- We also did some exploration on the effectiveness of static analysis tools on large code bases such as the Mozilla Firefox code by comparing its results against documented bug fixes.

Breezy Framework

Breezy tracks bugs over time provided a repository and set of static bug checkers to run



Breezy Framework

Demo Video

Breezy Framework

What is a bug?

- File
- Line of code (LOC)
- Code segment
- Type of bug
- Description

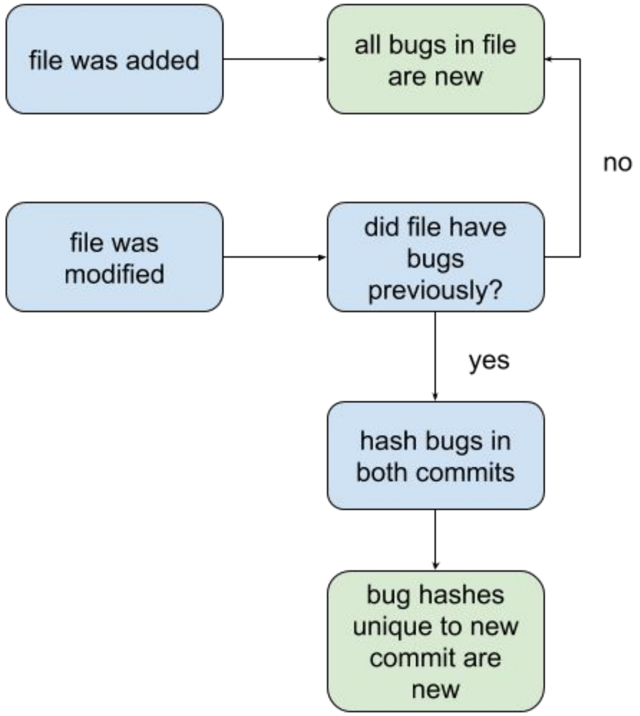
What is a bug report?

- Data structure to store list of bugs in a single commit
- Bug reports are generated and saved for specified commit window
- This can be the longest step in the process depending on repo size, num checkers used, etc.
- It took us ~2 hours to get single report on firefox with clang.

Breezy Framework

When is a bug born?

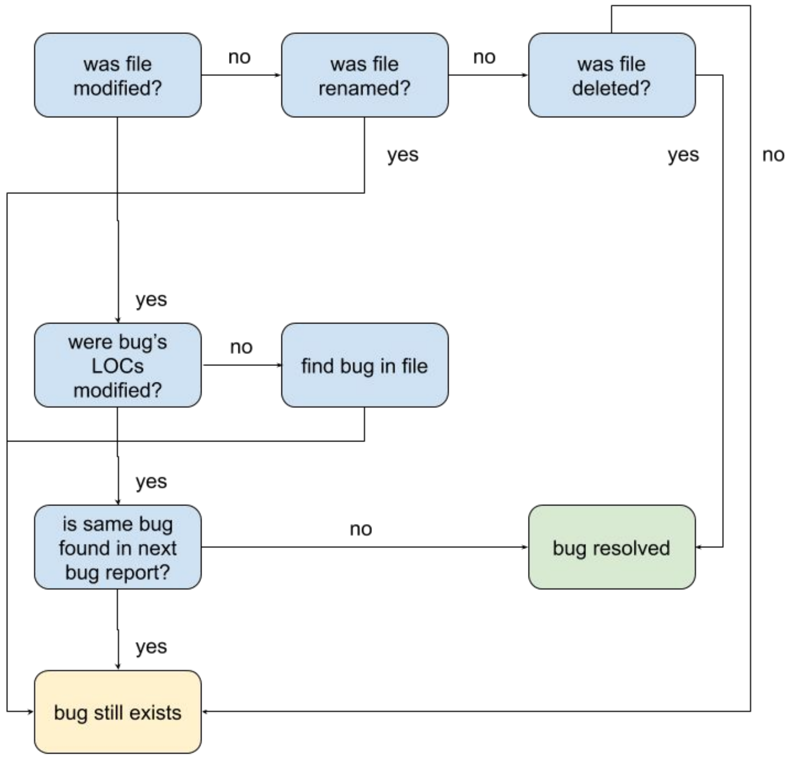
- In order to track bugs, we have to know when new ones pop up.
- Between two commits, new bugs show up in modified or added files.
- Bugs that are unique to the new commit that aren't in the previous commit are considered to be new.
- We hash the bugs to determine whether they are the same between commits.


file was added
all bugs in file are new
no
file was modified
did file have bugs previously?
yes
hash bugs in both commits
bug hashes unique to new commit are new

Breezy Framework

When is a bug resolved?

- For a single file, we use this approach to determine whether bugs in the file are resolved between commits
- A bug can be resolved when its LOC is changed and it cannot be found in the next commit
- When a bug isn't modified but moves around in the file, it still exists and we find it using diff info



Breezy Framework

Adding support for new checkers

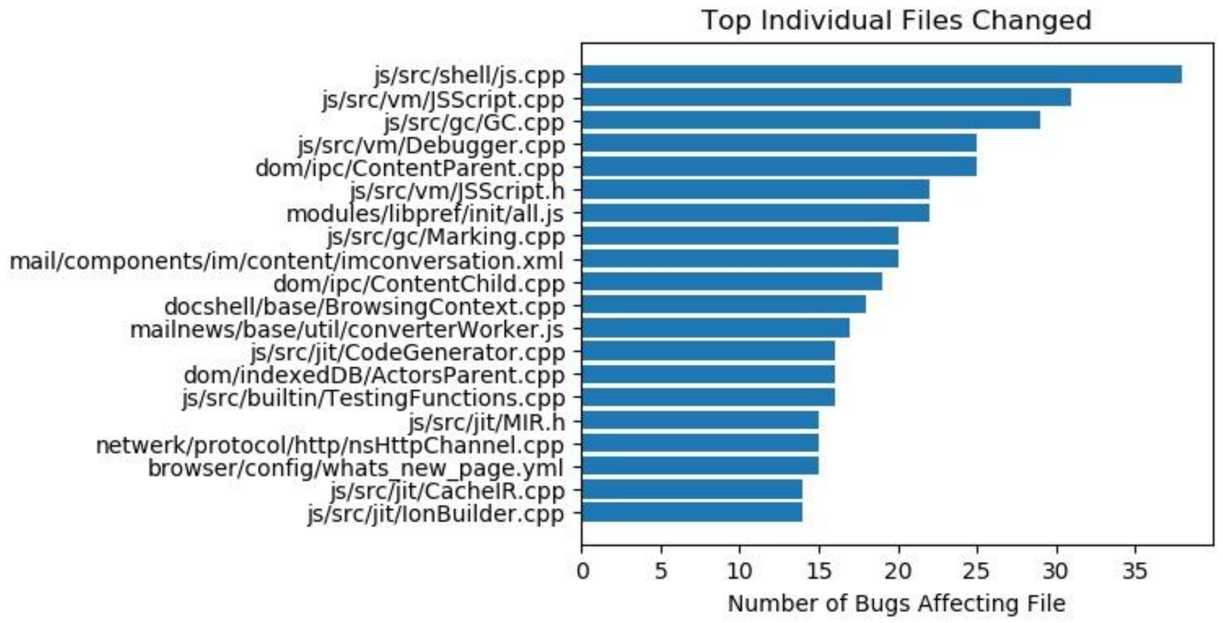
- We tested using cppcheck, clang, and infer
- In order to use a new checker, we have to parse the output of the checker into our simple bug format
- Sometimes when there is building from source involved (clang & infer), it can be a bit tricky, but not conceptually difficult to plug a new checker into breezy
- Allows users to fine-tune their checkers and combination of checkers to best suit analyzing their repositories.
- Can be used as a glorious linter for example.

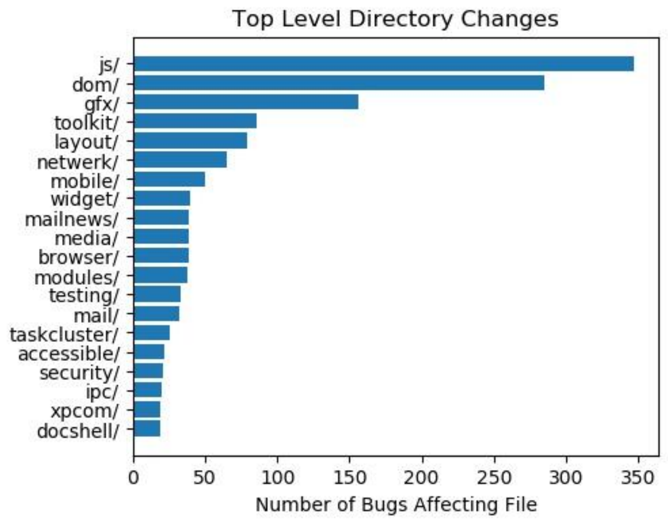
Analyzing Clang on Mozilla Firefox

Mozilla Firefox served as a perfect code base to investigate the effectiveness of static analysis tools as well as to evaluate our own tool. Benefits included:

- Large, bug-prone source code for analysis
- Well maintained bug documentation through Bugzilla
- Git diffs associated with each Bugzilla report help us cross reference bugs we find with static analysis tools
- Through Bugzilla, we looked at the FIXED bugs of all critical and major bugs over the past two years to see which files/directories were modified most frequently.

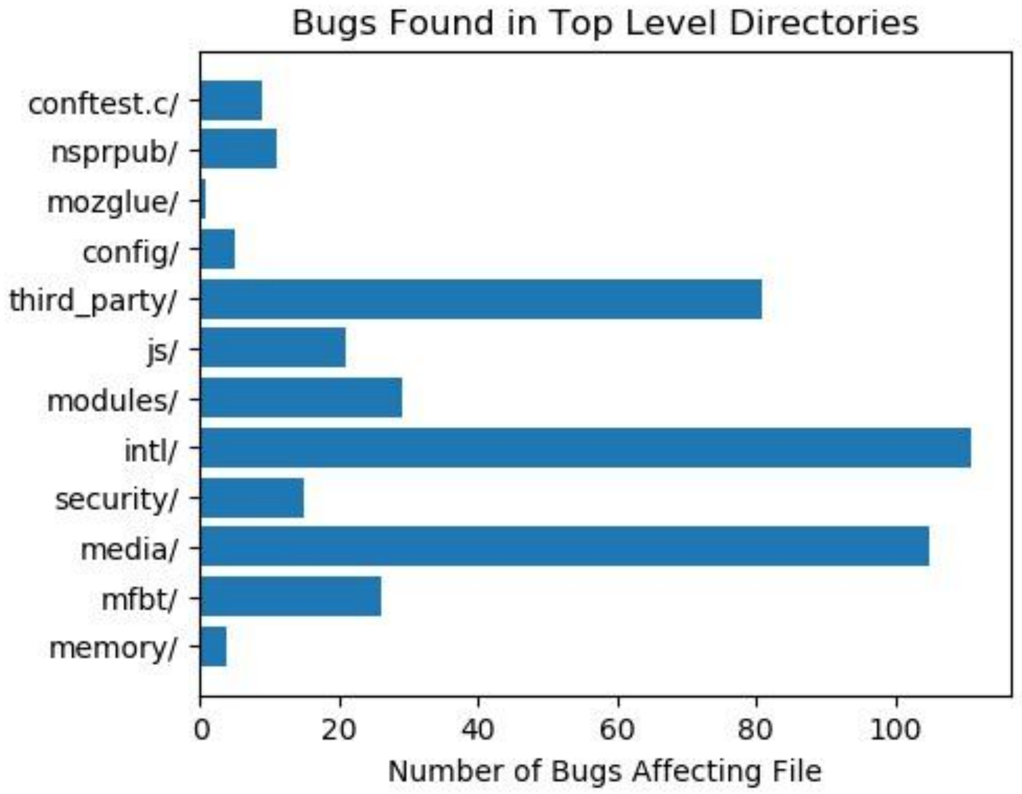
Firefox Bug File Change Frequencies



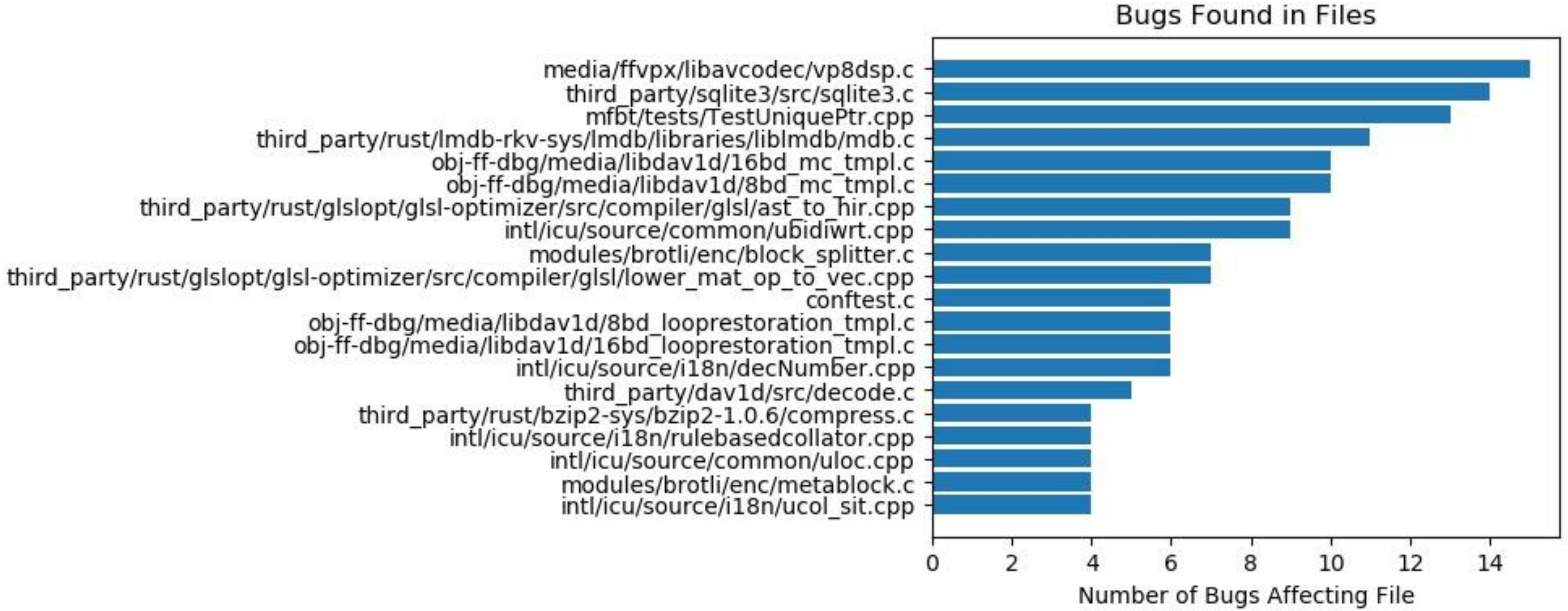


**Only top 20 files/directories were represented in the graphs

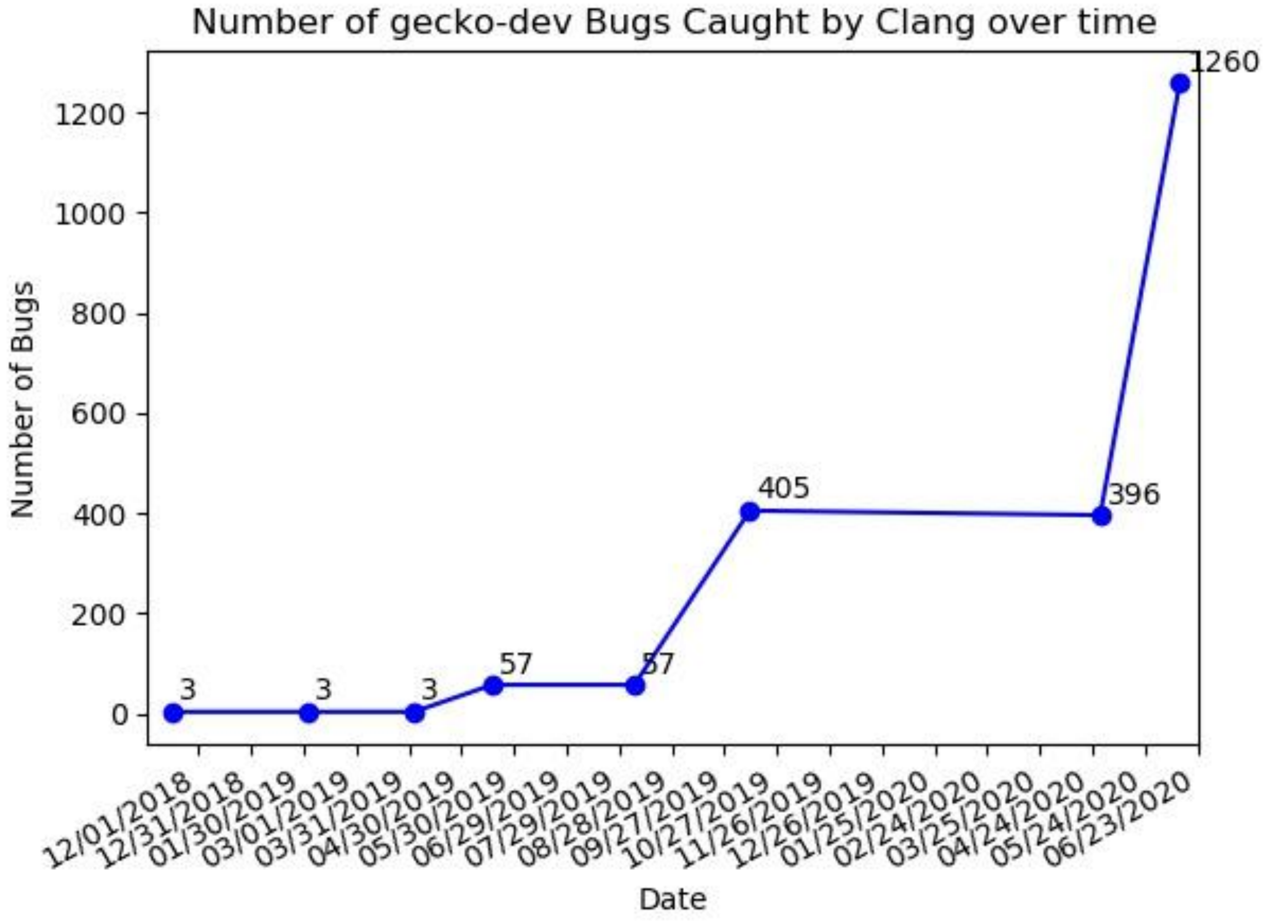
Bugs Found Using Clang



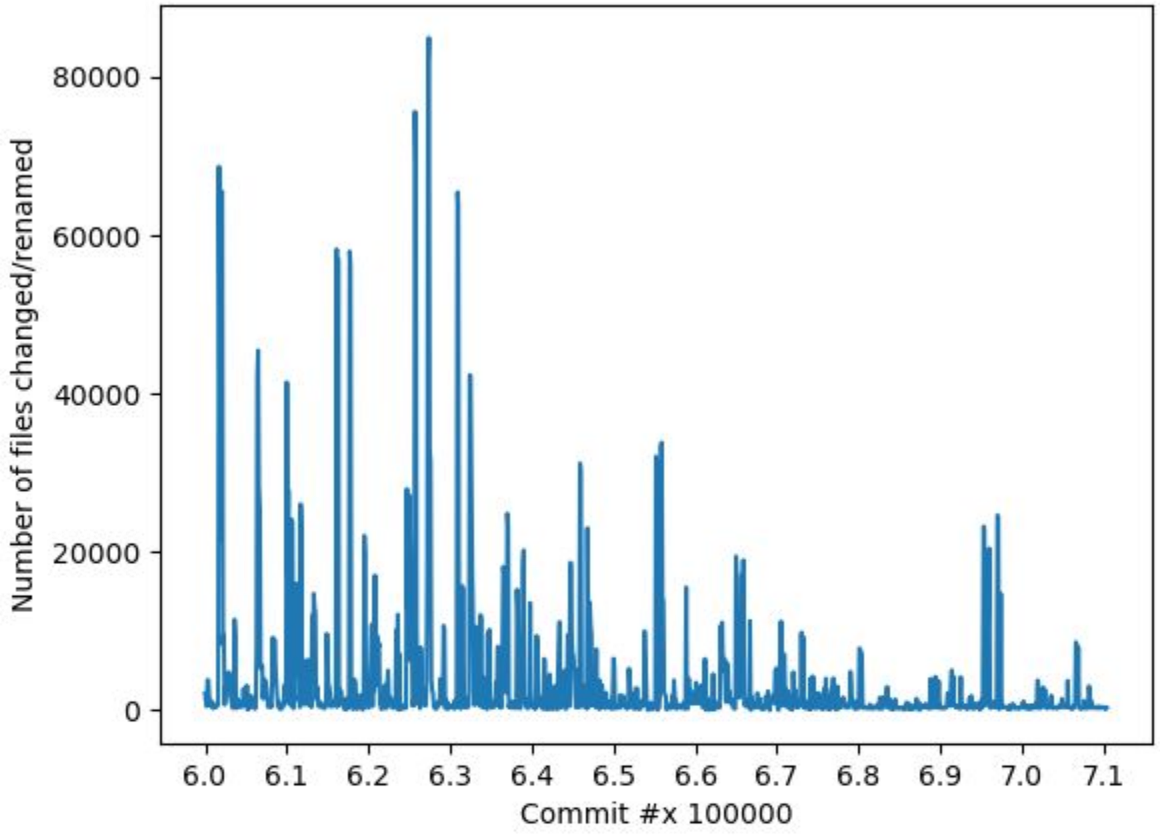
- Ran clang on gecko-dev in intervals of 10,000 commits across the entire lifespan
- Bugs documented by Bugzilla are significantly skewed towards JS related files.
- We hypothesize that this is because Bugzilla bugs are more logical bugs and bugs found in clang are on less crucial code paths.
- Many bugs found in media handling files and third party code



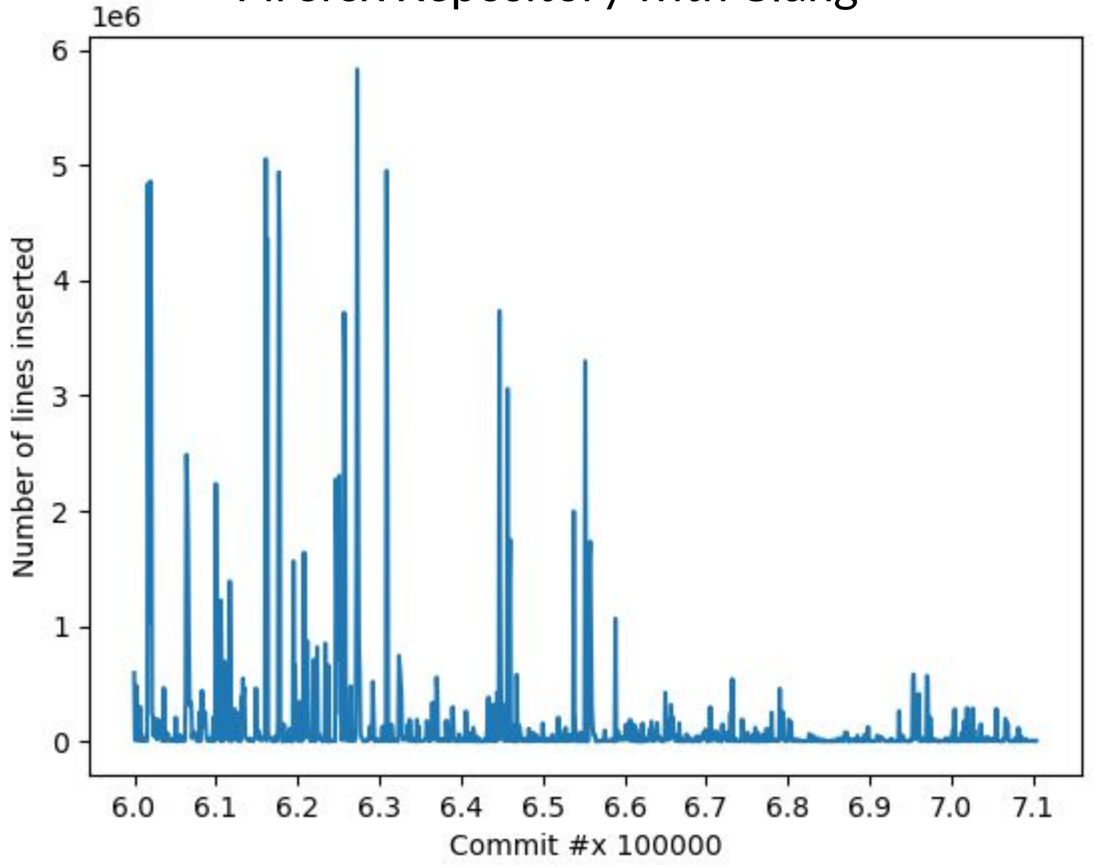
Bugs Found in Files
media/ffvpx/libavcodec/vp8dsp.c
third_party/sqlite3/src/sqlite3.c
mfbt/tests/TestUniquePtr.cpp
third_party/rust/lmdb-rkv-sys/lmdb/libraries/liblmdb/mdb.c
obj-ff-dbg/media/libdav1d/16bd_mc_tmpl.c
obj-ff-dbg/media/libdav1d/8bd_mc_tmpl.c
third_party/rust/glslopt/glsl-optimizer/src/compiler/glsl/ast_to_hir.cpp
intl/icu/source/common/ubidiwrt.cpp
modules/brotli/enc/block_splitter.c
third_party/rust/glslopt/glsl-optimizer/src/compiler/glsl/lower_mat_op_to_vec.cpp
conftest.c
obj-ff-dbg/media/libdav1d/8bd_looprestoration_tmpl.c
obj-ff-dbg/media/libdav1d/16bd_looprestoration_tmpl.c
intl/icu/source/i18n/decNumber.cpp
third_party/dav1d/src/decode.c
third_party/rust/bzip2-sys/bzip2-1.0.6/compress.c
intl/icu/source/i18n/rulebasedcollator.cpp
intl/icu/source/common/uloc.cpp
modules/brotli/enc/metablock.c
intl/icu/source/i18n/ucol_sit.cpp
0
2
4
6
8
10
12
14
Number of Bugs Affecting File


Number of gecko-dev Bugs Caught by Clang over time
Number of Bugs
Date
0
200
400
600
800
1000
1200
12/01/2018
12/31/2018
01/30/2019
03/01/2019
03/31/2019
04/30/2019
05/30/2019
06/29/2019
07/29/2019
08/28/2019
09/27/2019
10/27/2019
11/26/2019
12/26/2019
01/25/2020
02/24/2020
03/25/2020
04/24/2020
05/24/2020
06/23/2020
3
3
3
57
57
405
396
1260

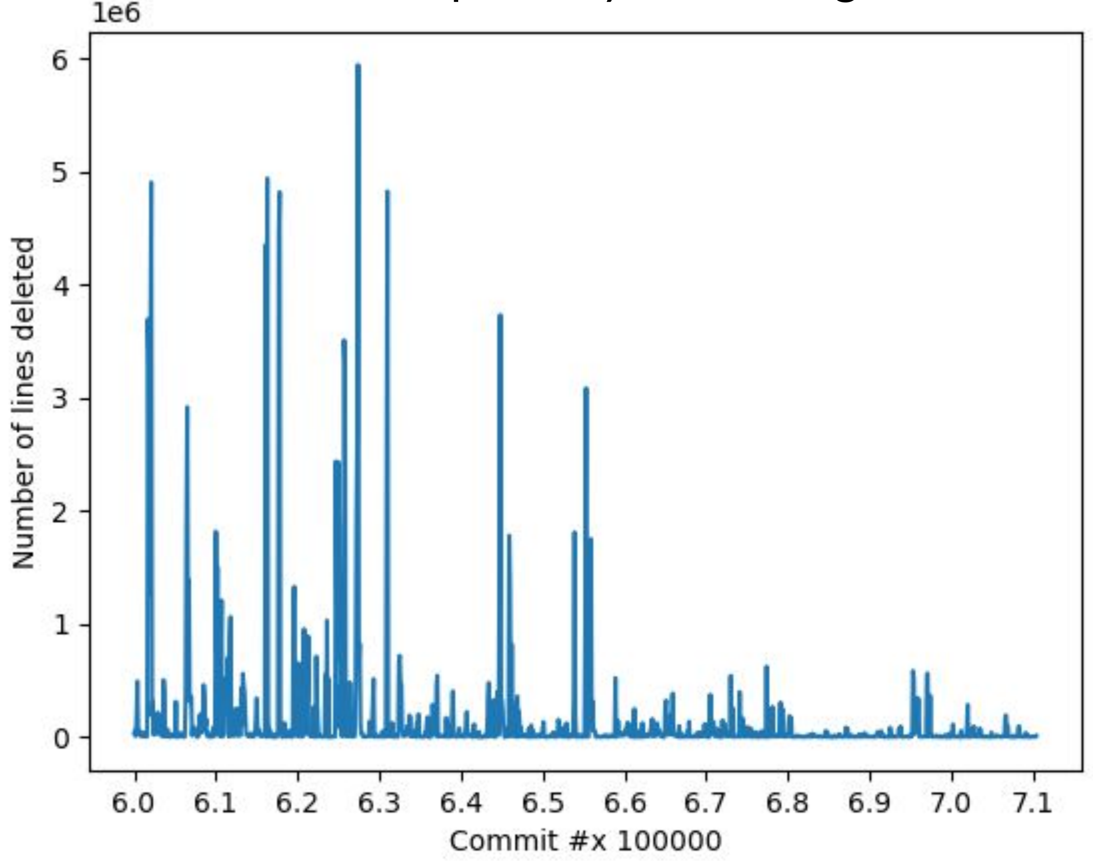
Firefox Repository with Clang



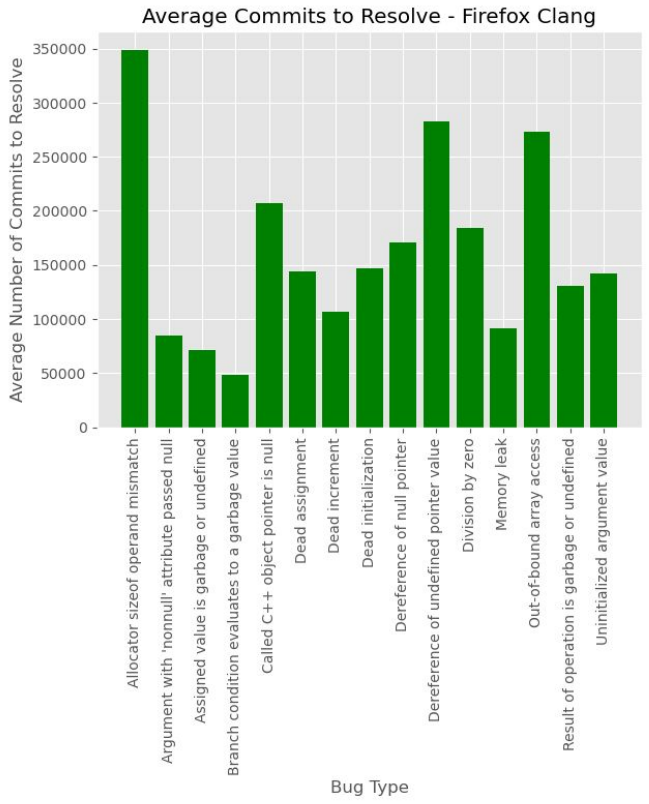
Firefox Repository with Clang



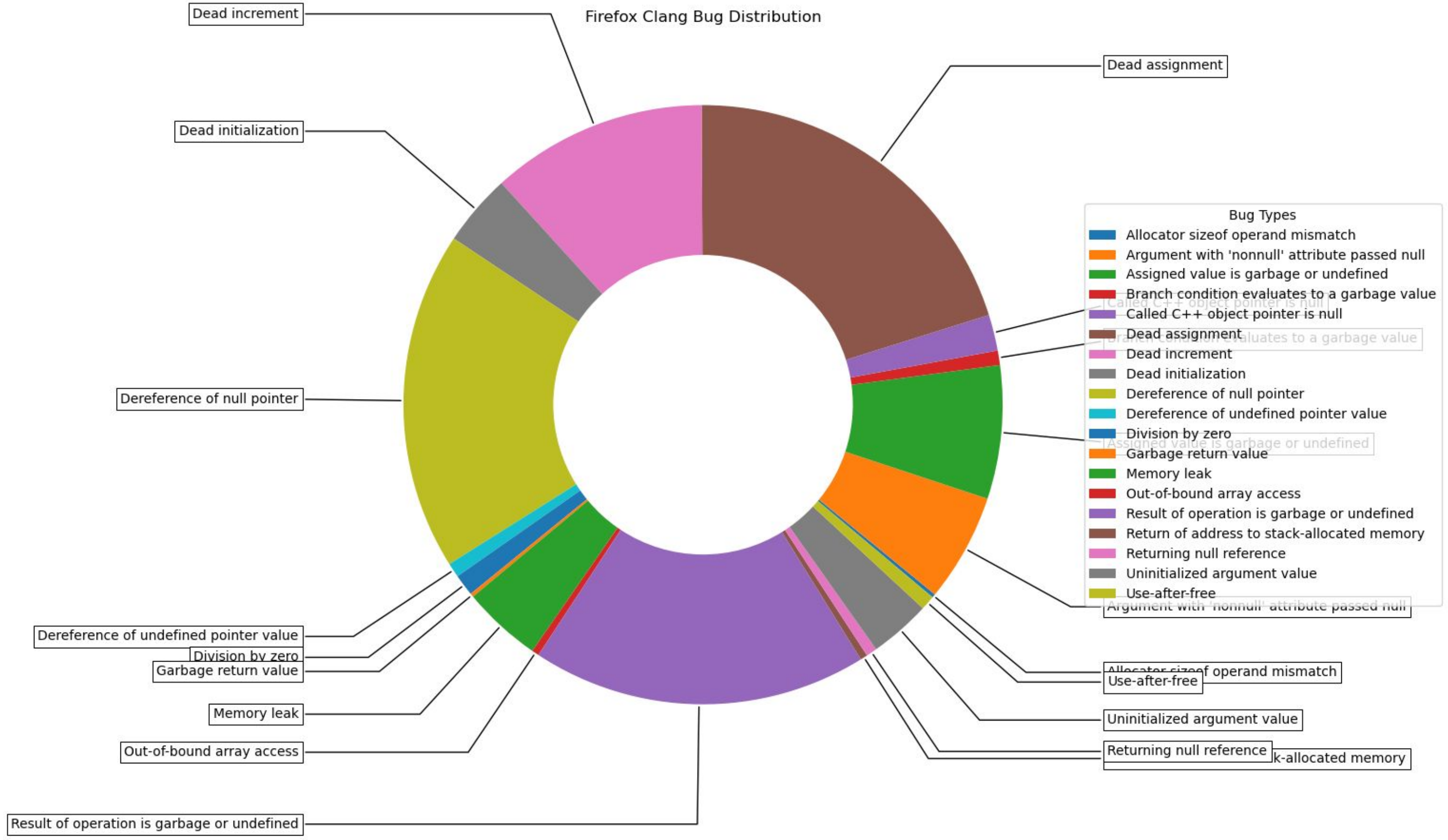
Firefox Repository with Clang



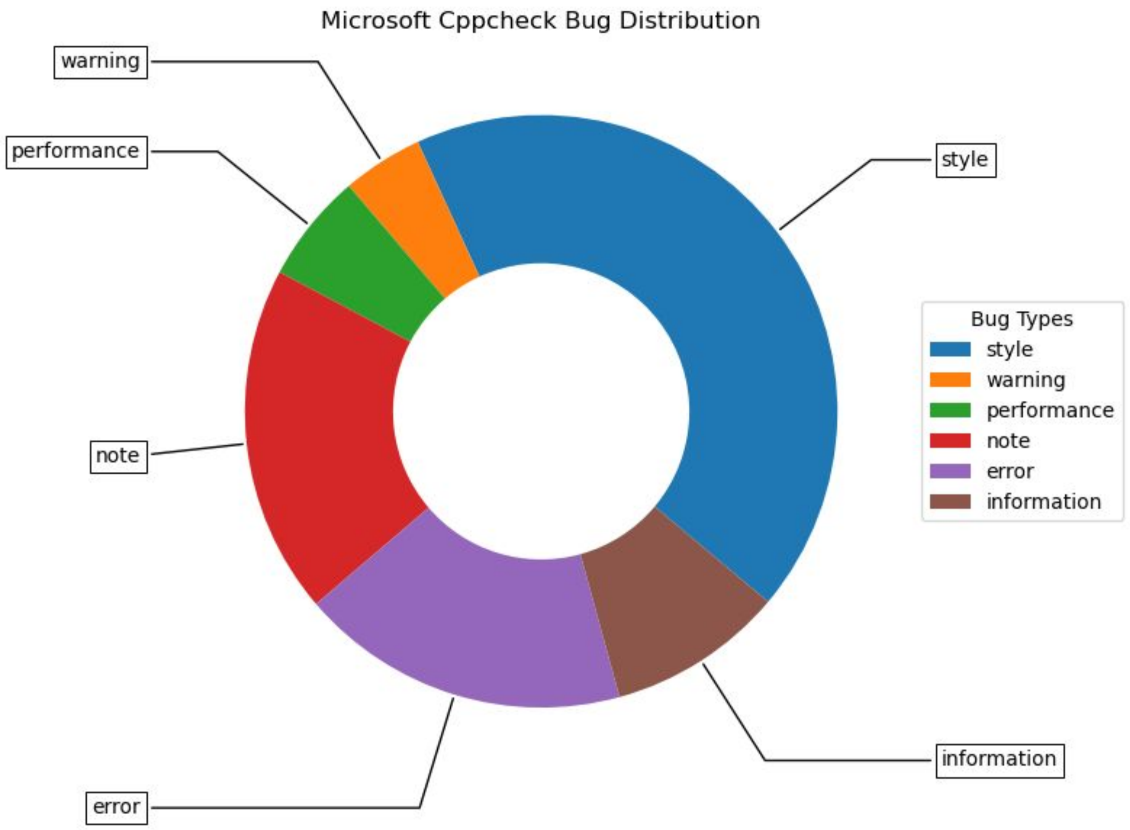
Firefox Repository with Clang



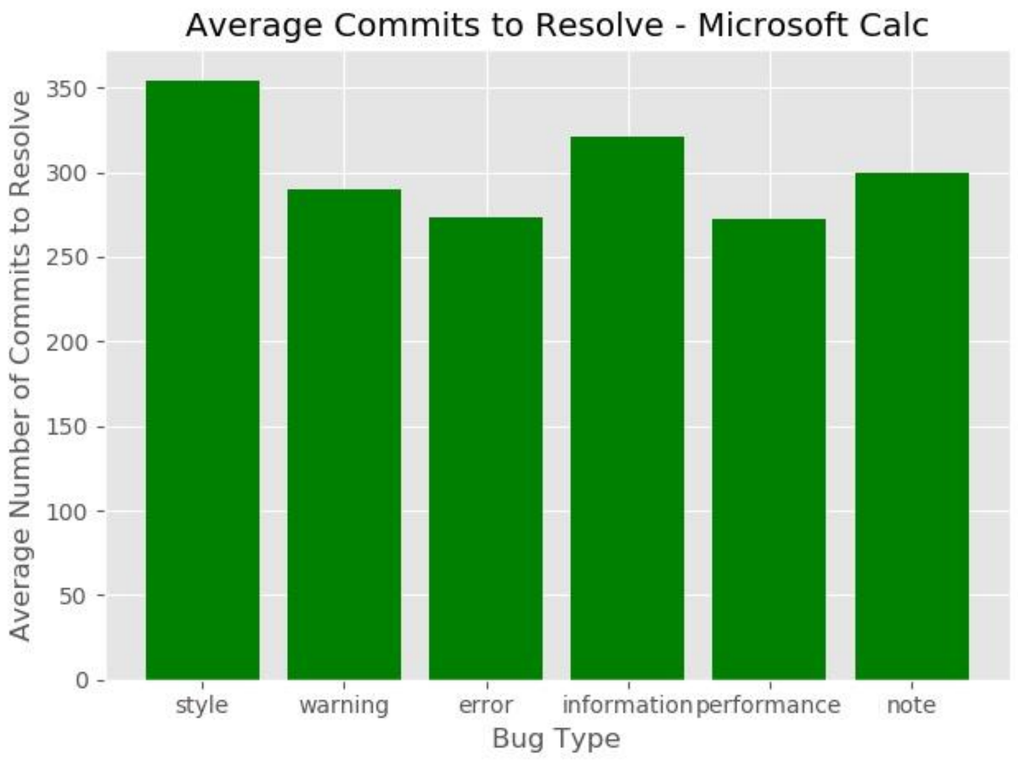
Firefox Repository with Clang



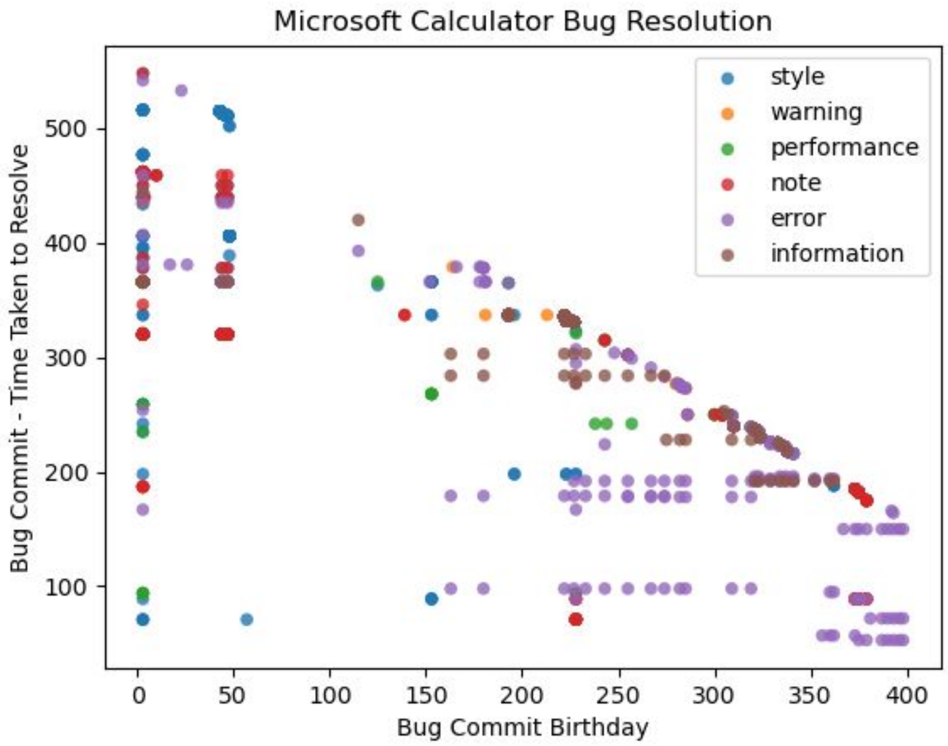
Microsoft Calculator Repository with Cppcheck



Microsoft Calculator Repository with Cppcheck



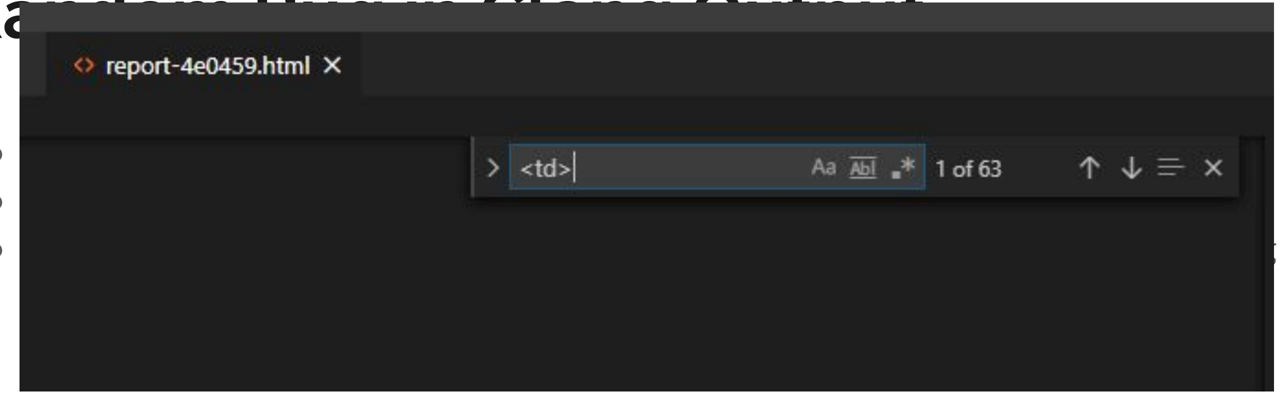
Microsoft Calculator Repository with Cppcheck



Random Bug in Clang Output

- Using BeauitfulSoup to parse through Clang Outputs and record bugs
- HTML from Clang Output is malformed
- Python’s built-in html.parser tries to accommodate malformed HTML by inserting in additional tags when needed

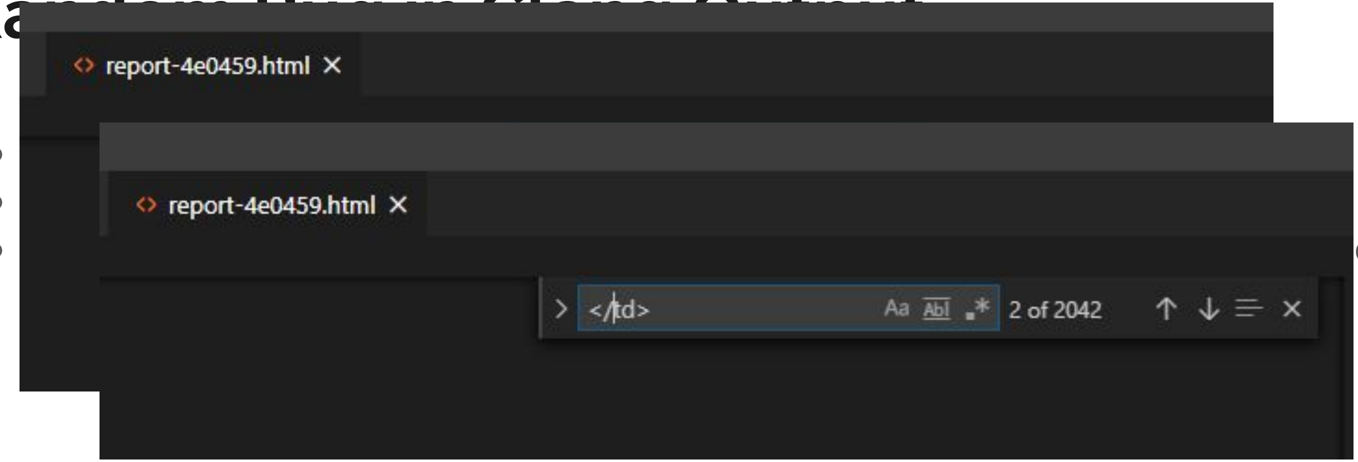
Ra



report-4e0459.html

-
-
- in additional

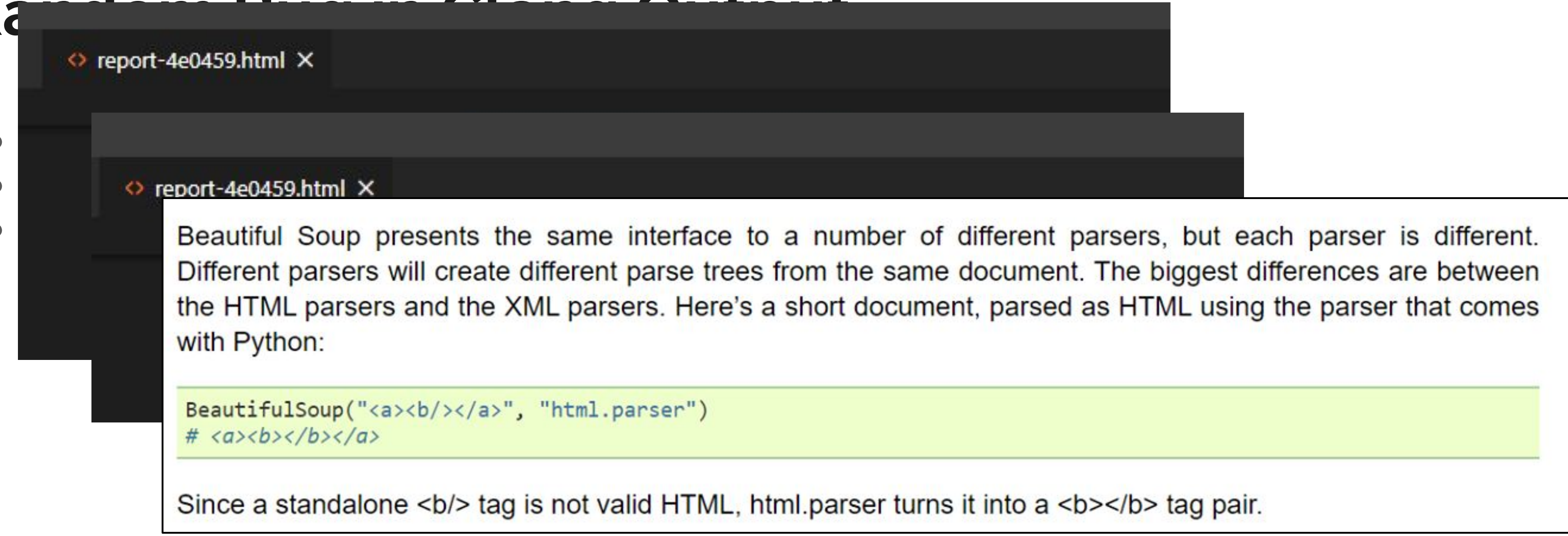
Ra



```
</td>    2 of 2042
```

ditional

Ra



Beautiful Soup presents the same interface to a number of different parsers, but each parser is different. Different parsers will create different parse trees from the same document. The biggest differences are between the HTML parsers and the XML parsers. Here's a short document, parsed as HTML using the parser that comes with Python:

```
BeautifulSoup("<a><b/></a>", "html.parser")
# <a><b></b></a>
```

Since a standalone <b/> tag is not valid HTML, html.parser turns it into a <b></b> tag pair.

Future Work: Better Bug Ranking

- Current static bug checkers have tons of false positives and no legitimate way of filtering or ranking the reported bugs.
- We can potentially use the time it takes for a bug to resolve as an indicator for how important the bug is.
    - This is not necessarily always true, but with a data driven approach in which multiple bugs and multiple checkers are used, it might be a useful signal for bug importance.
- With the data, we can train a model to predict when a new bug will be resolved, thereby giving bugs that we are currently facing a ranking.
- This ranking system can be used downstream in IDEs, code reviews, etc. in production environments.

Future Work: Large Scale Bug Trends

- Can we observe common patterns of bugs that developers are making in general by examining several large open source projects?
- Creating bug histories for repositories can give developers an idea of how healthy their repository currently is as compared to at different points in the past
    - For example, we might be able to pick up that a certain type of bug is made frequently when a large scale refactor takes place in a repository
- Can we extend the same principles used in breezy to more complicated, more than single LOC bugs, and potentially detect these?

Questions?