Investigating Preprocessor-Based Bugs in C Program Families

Flávio Medeiros

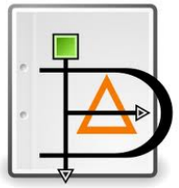
PhD Candidate

Rohit Gheyi
Márcio Ribeiro

UFAL

UFCG

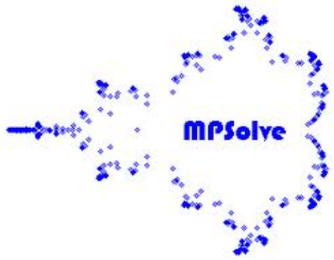
May 2014 - Dagstuhl

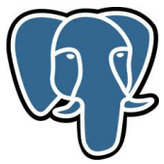


Dia

Linux

Xfig





PostgreSQL

CPP

Despite the widespread use of the C Preprocessor (CPP), it is often criticised due to:

- ☑ **Negative impact on code quality and maintainability**
- ☑ **Error-prone characteristics**

☑ Program family: Libpng

```
1.  static void progressive_row(png_structp ppln){
2.    if (new_row != NULL) {
3.      if (y >= dp->h)
4.        row = store_image_row(dp->ps, pp, 0, y);
5.  #ifdef INTERLACING
6.      if (dp->do_interlace){
7.        // Code Here..
8.      } else
9.        png_progressive_combine_row(pp, row, new_row);
10.   } else if (dp->interlace_type == PNG_INTERLACE_ADAM7)
11.     png_error("missing row");
12. #endif
13.}
```

☑ Syntax error: !INTERLACING

```
1. static void progressive_row(png_structp ppln){
2.   if (new_row != NULL) {
3.      if (y >= dp->h)
4.         row = store_image_row(dp->ps, pp, 0, y);
5. #ifdef INTERLACING
6.      if (dp->do_interlace){
7.         // Code Here..
8.      } else
9.         png_progressive_combine_row(pp, row, new_row);
10.   } else if (dp->interlace_type == PNG_INTERLACE_ADAM7)
11.      png_error("missing row");
12. #endif
13.}
```

```
1.   static bin_tree_t * parse_bracket_exp (){
2.      // Code Here..
3.      re_bitset_ptr_t sbcset;
4.   #ifdef I18N
5.      re_charset_t *mbcset;
6.      // Code Here..
7.   #endif
8.      // Code Here..
9.      sbcset = (re_bitset_ptr_t) calloc (sizeof (unsigned int), BITSET);
10. #ifdef I18N
11.    mbcset = (re_charset_t *) calloc (sizeof (re_charset_t), 1);
12. #endif
13. #ifdef I18N
14.    if (BE (sbcset == NULL || mbcset == NULL, 0))
15. #else
16.    if (BE (sbcset == NULL, 0))
17. #endif
18.      {
19.         *err = REG_ESPACE;
20.         return NULL;
21.      }
22.   // Code Here..
23. }
```

Memory leak: I18N

```
static bin_tree_t * parse_bracket_exp (){
   // Code Here..
   re_bitset_ptr_t sbcset;
#ifdef I18N
   re_charset_t *mbcset;
   // Code Here..
#endif
   // Code Here..
   sbcset = (re_bitset_ptr_t) calloc (sizeof (unsigned int), BITSET);
#ifdef I18N
  mbcset = (re_charset_t *) calloc (sizeof (re_charset_t), 1);
#endif
#ifdef I18N
  if (BE (sbcset == NULL || mbcset == NULL, 0))
#else
  if (BE (sbcset == NULL, 0))
#endif
   {
      *err = REG_ESPACE;
      return NULL;
   }
  // Code Here..
}
```

☑ Studies relate the CPP usage to bugs

☑ However, only a few studies to:

- ✓ Investigate preprocessor-based bugs
- ✓ Quantify to what extent bugs occur in practice

Empirical Study

- ☑ Investigate and quantify preprocessor-based bugs in C Program Families

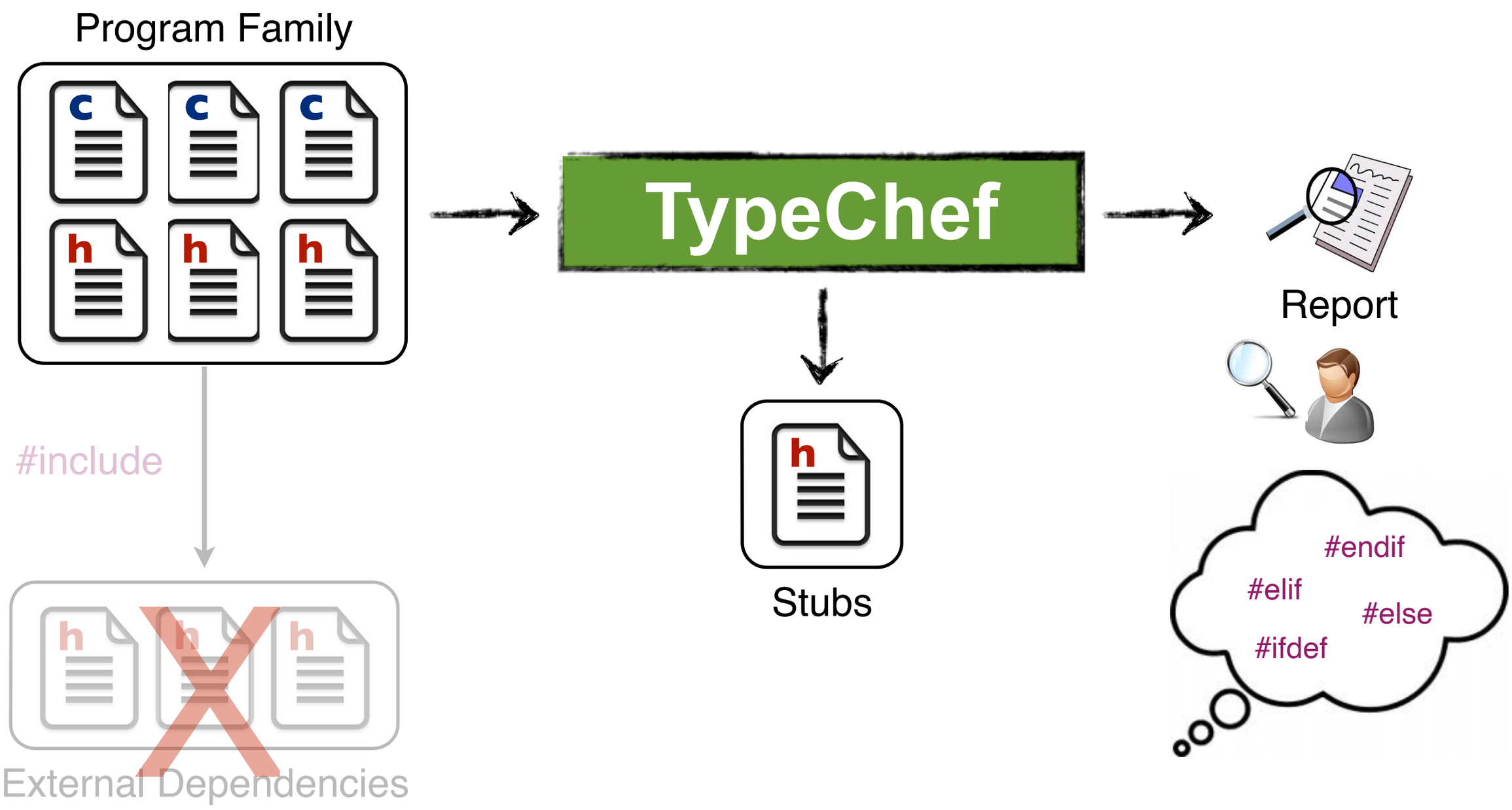
40 Releases

84,000 Commits
22 program families

Strategies to Detect Preprocessor-Based Bugs

1. Strategy to identify syntax errors

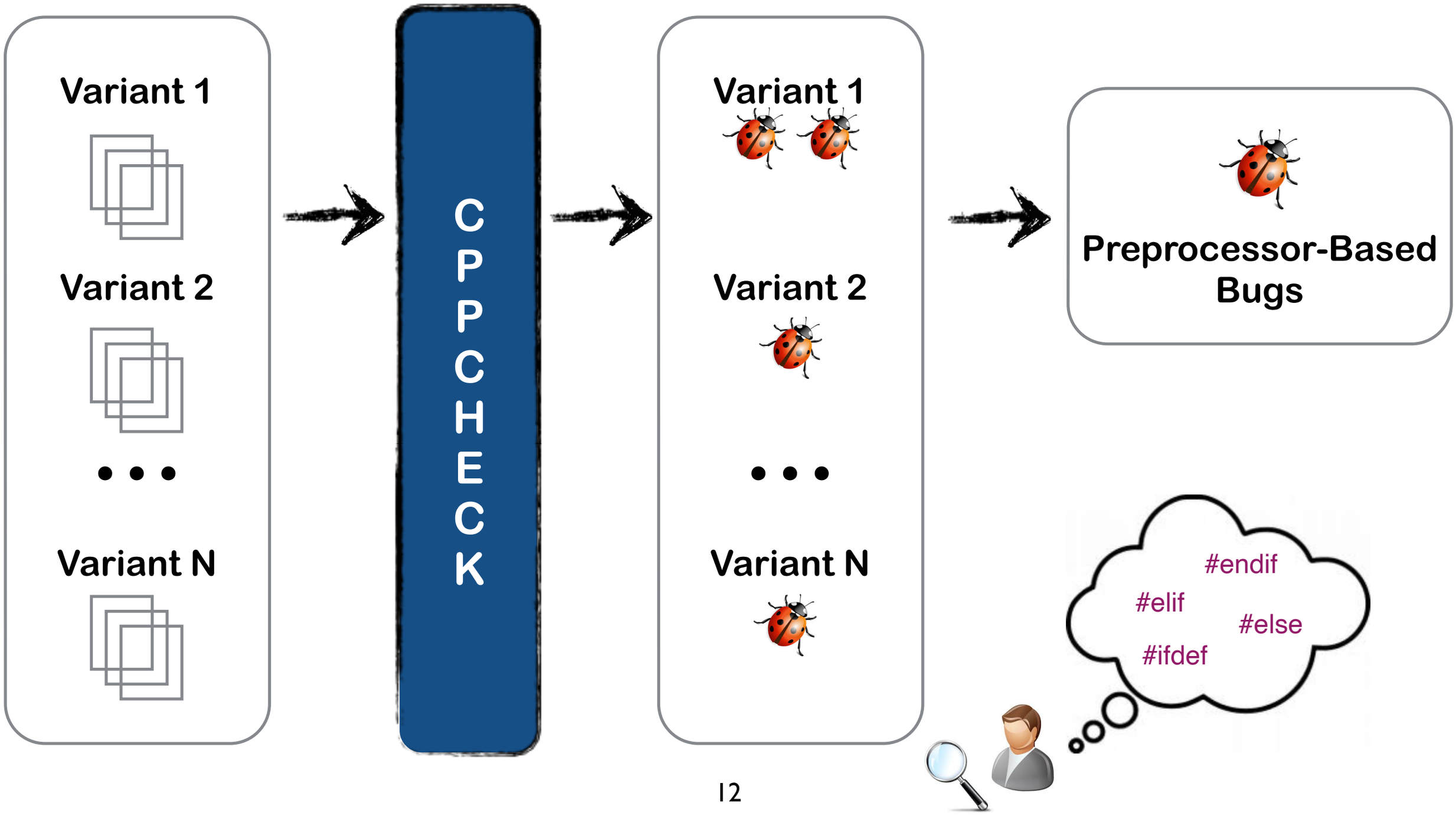
TypeChef



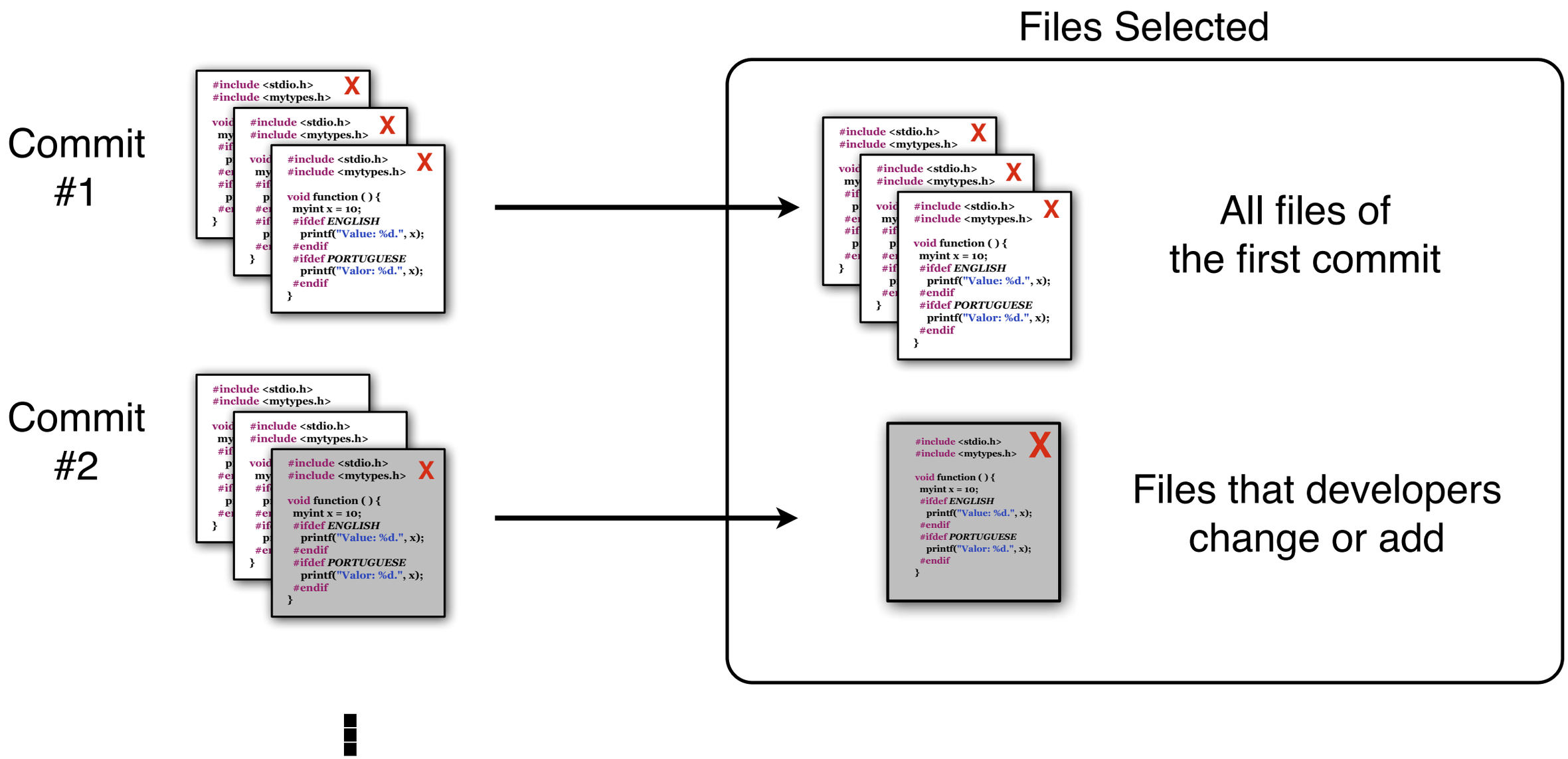
2. Strategy to identify semantic bugs

CppCheck

512 variants



Bugs in Software Repositories



Research Questions

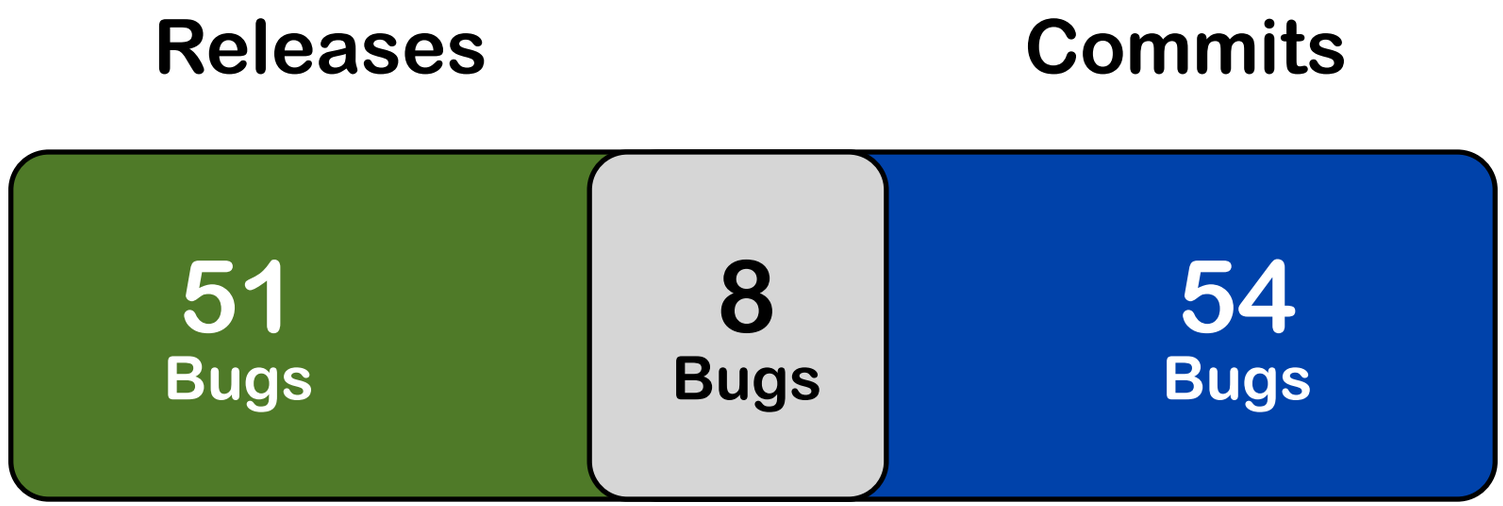
Q1. Do program families contain preprocessor-based bugs?

Q2. For how long a preprocessor-based bug remains in commits of a particular source file?

Q3. How do program families developers introduce the preprocessor-based bugs?

Q4. Do developers introduce more bugs in functions with preprocessor directives?

Q1. Do program families contain preprocessor-based bugs?



121 distinct bugs

9

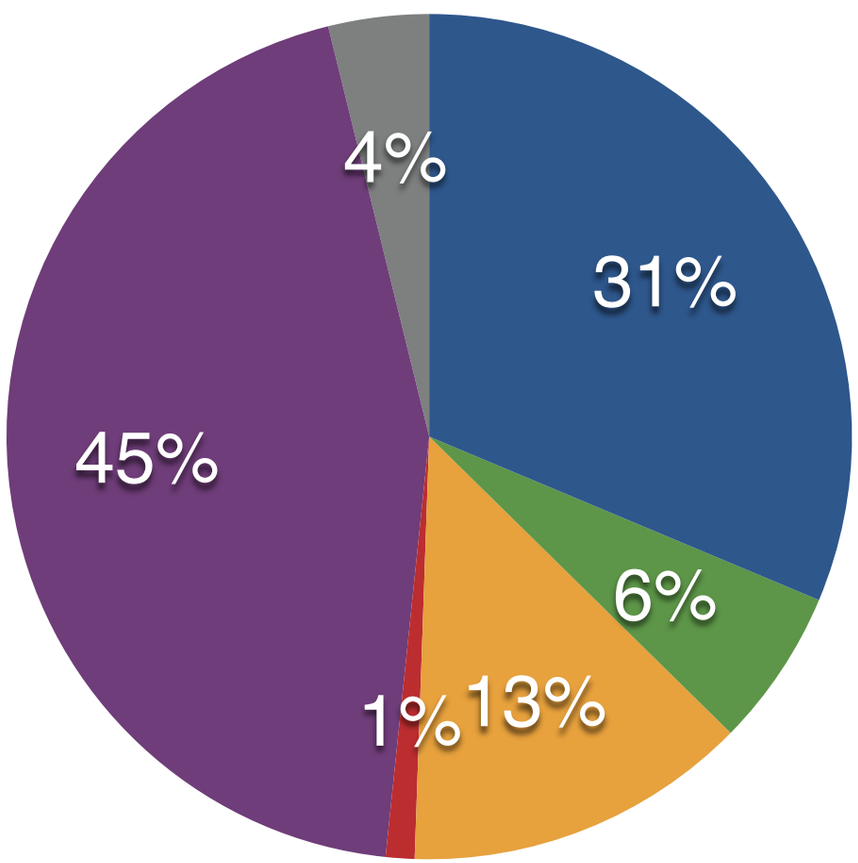
6

6

4

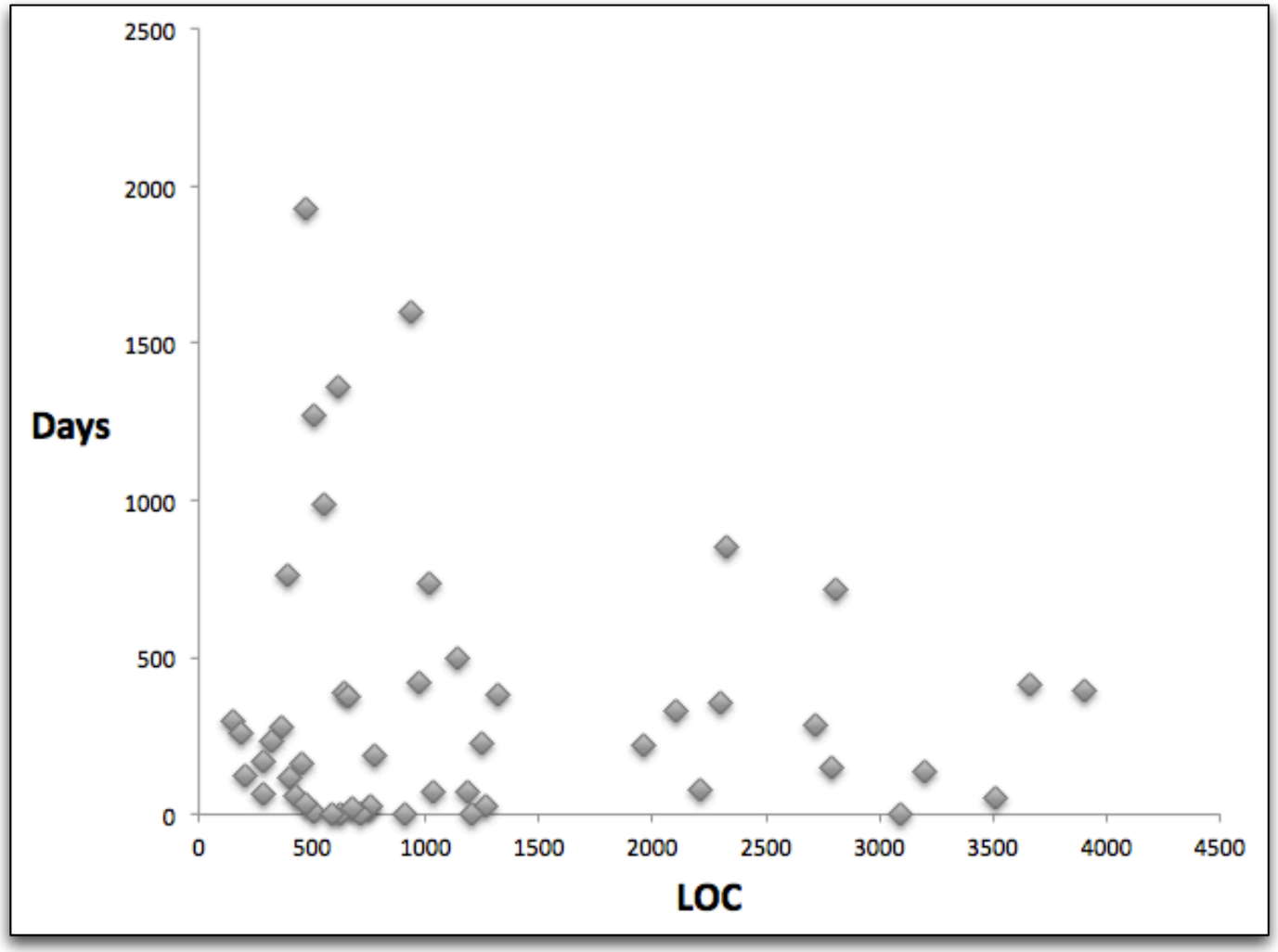
4

121 distinct bugs



Q2. For how long a preprocessor-based bug remains in commits of a particular source file?

- It varies from days to years
- No correlation with the LOC or number of developers



Q3. How do program families developers introduce the preprocessor-based bugs?

Syntax errors

46.67% by modifying existing code and adding directives, e.g., to support a new operating system.

Semantic Bugs

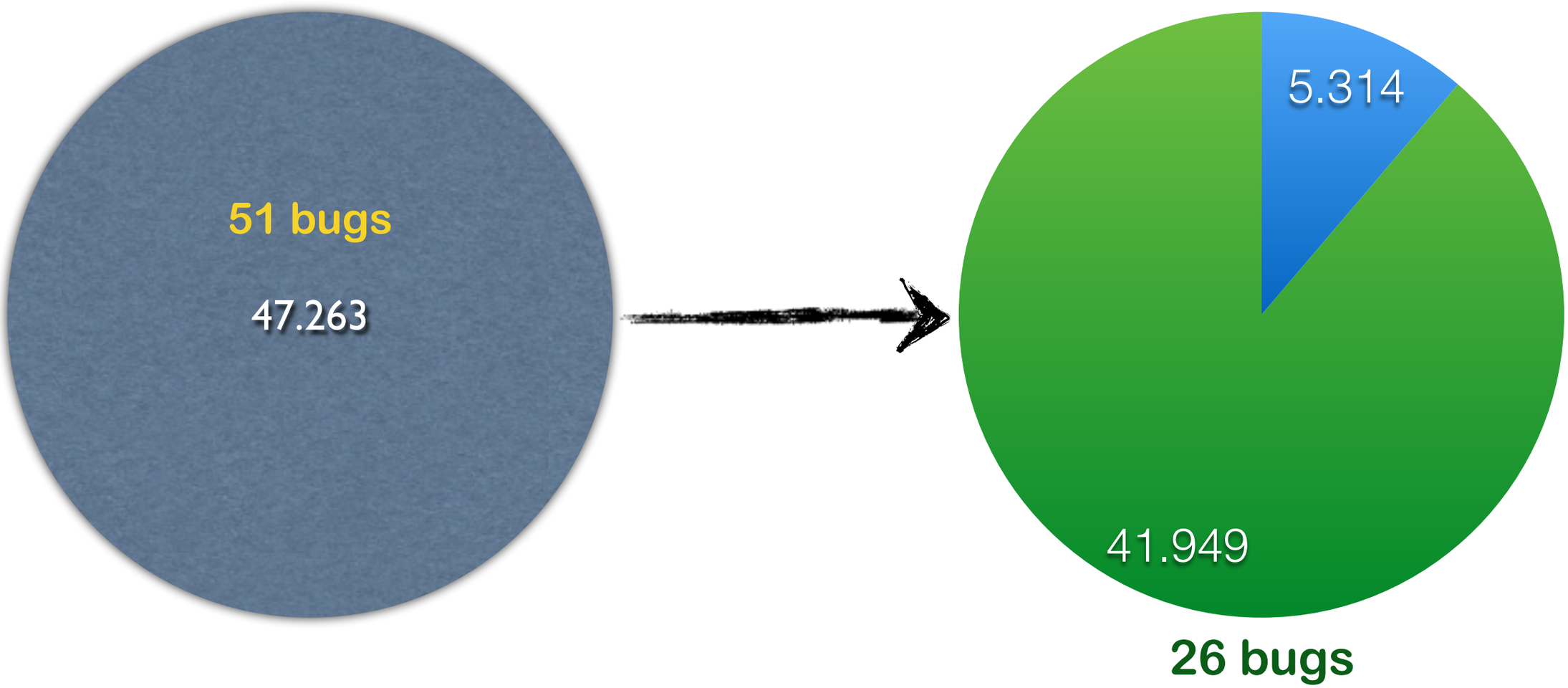
44.12% by adding existing new code, e.g., a new source file, or a new function

Q4. Do developers introduce more bugs in functions with preprocessor directives?

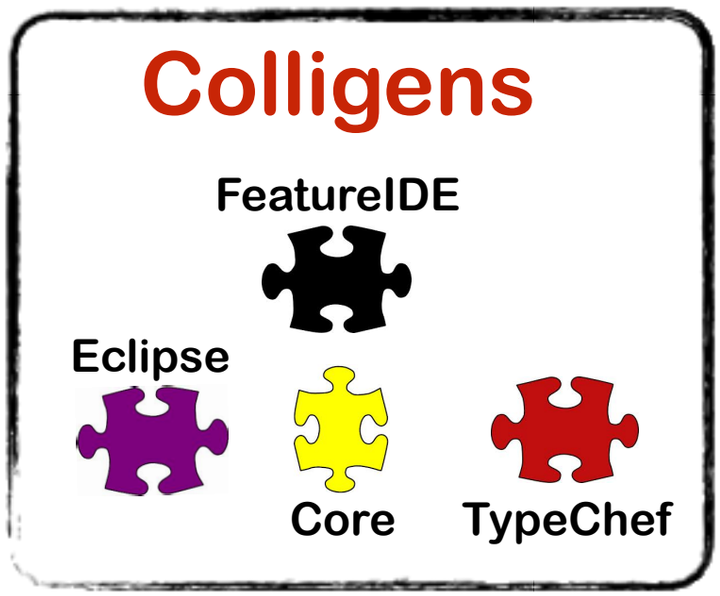
- Function with Directives **11%**
- Functions without Directives **89%**

40 Program Family Releases

- **Total Number of Functions**



Tool Support

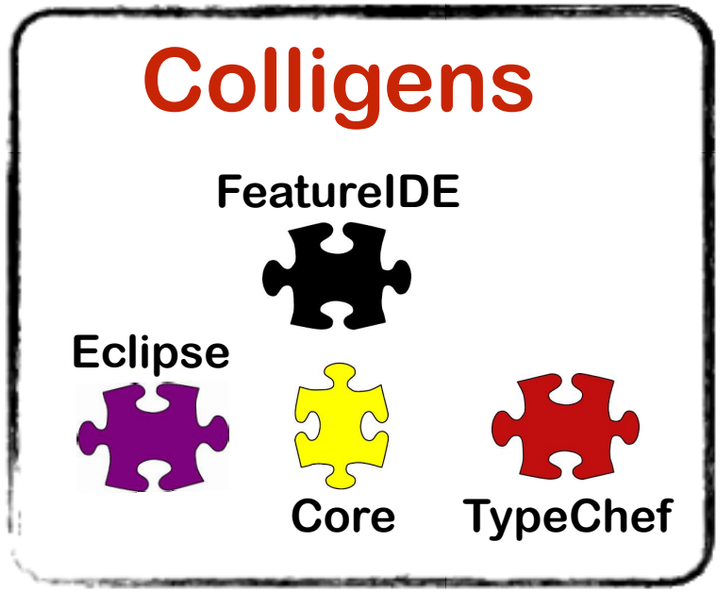


- ☑ Strategies to detect preprocessor-based bugs
- ☑ Eclipse plug-in based on FeatureIDE

Defective Evolution

Detecting existing problems like bugs

Tool Support



+ Refactorings to remove incomplete (undisciplined) annotations without cloning code

Perfective Evolution

Improve code quality to avoid problem in the future

Incomplete Conditions

Bad Smell 

```
1. if ( condition1
2. #ifdef expression1
3.     && condition2
4. #endif
5. ){
6.     // Statements..
7. }
```


```
1. bool test = condition1;
2. #ifdef expression1
3.     test = test && condition2;
4. #endif
5. if (test){
6.     // Statements..
7. }
```

Precondition: original code is not using variable test in this scope

- **No code cloning**
- **No extra lines of code**
- **No extra preprocessor-directives**

Conclusions

We find 121 preprocessor-based bugs in program family releases and commits

We submit 23 patches to fix bugs, developers accept more than 74% of them

Developers introduce more syntax errors when changing code and adding directives

Developers introduce more semantic bugs when adding new code

Our results suggest that developers introduce more bugs in functions with preprocessor directives

Thanks!