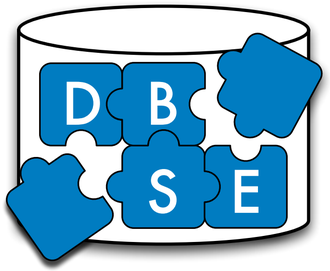

Databases
and
Software
Engineering

The Effect of Preprocessor Annotations on Code Comprehension

A Developer's Perception

FOSD Meeting 2019
Tue. 12 – Fri. 15 March 2019, Weimar, Germany

Maria Kanyshkova, **Wolfram Fenske,** Sandro Schulze

Databases and Software Engineering
University of Magdeburg, Germany

C Preprocessor Annotations: Frequently Used & Often Criticized

- “#ifdef considered harmful” [Spencer & Collyer, USENIX’92; Favre, IWPC’97; Ernst et al., TSE’02; …]
- #ifdefs may be associated with faults [e.g., Medeiros et al., GPCE’13 & ECOOP’15; GPCE’15; Ferreira et al., SPLC’16]
- #ifdefs hurt program comprehension [e.g., Walkingshaw et al., VL/HCC’11; Melo et al., ICSE’16]
- Programmers don’t like undisciplined #ifdefs [e.g., Medeiros et al., TSE’17; Malaquias et al., ICPC’17]

Variability-Aware Code Smells

- Inappropriate use of #ifdefs may be a code smell
  [Fenske & Schulze, VaMoS'15]
- Variability-aware code smells can be detected with metrics
  [Fenske et al., SCAM'15]
- More #ifdef use doesn't affect change-proneness (much)
  [Fenske et al., GPCE'17]

Metrics of #ifdef Use

Feature locations (number of #ifdefs)

Negation & number of #else branches

Undisciplined annotations

Featu

Lines of annotated code (lines of code within #ifdefs)

Do metrics of #ifdef use reflect how developers perceive the code?

```
1     char_u  *
2  fix_fname(fname)
3     char_u  *fname;
4  {
5  #ifdef UNIX
6     return FullName_save(fname, TRUE);
7  #else
8     if (!vim_isAbsName(fname)
9             || strstr((char *)fname, "..") != NULL
10            || strstr((char *)fname, "//") != NULL
11 # ifdef BACKSLASH_IN_FILENAME
12            || strstr((char *)fname, "\\\\") != NULL
13 # endif
14 # if defined(MSWIN) || defined(DJGPP)
                                                  NULL
                                                  ;
24    if (USE_LONG_FNAME)
25 #  endif
26    {
27        if (fname != NULL)
28            fname_case(fname, 0);
29    }
30 # endif
31
32    return fname;
33 #endif
34 }
```

Heavily annotated function from Vim

Methodology

- Online questionnaire
- Five heavily annotated C functions
- 1/2 of participants received refactored functions with simplified annotations

Methodology
Original vs. Refactored Function

```
1     char_u  *
2 fix_fname(fname)
3     char_u  *fname;
4 {
5 #ifdef UNIX
6     return FullName_save(fname, TRUE);
7 #else
8     if (!vim_isAbsName(fname)
9             || strstr((char *)fname, "..") != NULL
10            || strstr((char *)fname, "//") != NULL
11 # ifdef BACKSLASH_IN_FILENAME
12            || strstr((char *)fname, "\\\\") != NULL
13 # endif
14 # if defined(MSWIN) || defined(DJGPP)
15            || vim_strchr(fname, '~') != NULL
16 # endif
17            )
18         return FullName_save(fname, FALSE);
19
20     fname = vim_strsave(fname);
21
22 # ifdef USE_FNAME_CASE
23 #  ifdef USE_LONG_FNAME
24     if (USE_LONG_FNAME)
25 #  endif
26     {
27         if (fname != NULL)
28             fname_case(fname, 0);
29     }
30 # endif
31
32     return fname;
33 #endif
34 }
```

```
1  #ifdef UNIX
2
3      char_u  *
4  fix_fname(fname)
5      char_u  *fname;
6  {
7      return FullName_save(fname, TRUE);
8  }
9
10 #else /* !UNIX */
11
12     char_u  *
13 fix_fname(fname)
14     char_u  *fname;
15 {
16     int is_rel_name = !vim_isAbsName(fname)
17                             || strstr((char *)fname, "..") != NULL
18                             || strstr((char *)fname, "//") != NULL;
19 # ifdef BACKSLASH_IN_FILENAME
20     is_rel_name = is_rel_name || strstr((char *)fname, "\\\\") != NULL;
21 # endif
22 # if defined(MSWIN) || defined(DJGPP)
23     is_rel_name = is_rel_name || vim_strchr(fname, '~') != NULL;
24 # endif
25
26     if (is_rel_name)
27         return FullName_save(fname, FALSE);
28
29     fname = vim_strsave(fname);
30
31 # ifdef USE_FNAME_CASE
32 #  if !defined(USE_LONG_FNAME) || USE_LONG_FNAME
33     if (fname != NULL)
34         fname_case(fname, 0);
35 #  endif
36 # endif
37
38     return fname;
39 }
40 #endif
```

Methodology
Tasks

- Questionnaire sent to ~6000 GitHub developers
- Tasks for each function
  1. Two comprehension tasks
  2. Rate general code quality
     (e.g., regarding understandability)
  3. Specifically rate quality of #ifdef use
  4. Optionally explain why #ifdef use was inappropriate
- Received ~1000 responses

Preliminary Results
Program Comprehension Tasks

```
1     char_u  *
2 fix_fname(fname)
3     char_u  *fname;
4 {
5 #ifdef UNIX
6     return FullName_save(fname, TRUE);
7 #else
8     if (!vim_isAbsName(fname)
9             || strstr((char *)fname, "..") != NULL
10            || strstr((char *)fname, "//") != NULL
11 # ifdef BACKSLASH_IN_FILENAME
12            || strstr((char *)fname, "\\\\") != NUL
13 # endif
14 # if defined(MSWIN) || defined(DJGPP)
15            || vim_strchr(fname, '~') != NULL
16 # endif
17            )
18        return FullName_save(fname, FALSE);
19
20    fname = vim_strsave(fname);
21
22 # ifdef USE_FNAME_CASE
23 #  ifdef USE_LONG_FNAME
24    if (USE_LONG_FNAME)
25 #  endif
26    {
27        if (fname != NULL)
28            fname_case(fname, 0);
29    }
30 # endif
31
32    return fname;
33 #endif
34 }
```

1. Which of the following statements is true? [CD02]

- If UNIX is not defined, the function will be empty.
- If UNIX is defined, the function will return the result of another function.
- Whether UNIX is defined or not has no effect on the result of the function.
- None of the above is true.
- I don't know

2. When would line 20 be executed? [CD07]

Choose a combination of conditions that would lead to the desired execution.

- [ ] Never
- [ ] Always
- [ ] `UNIX` is defined
- [ ] `UNIX` is undefined
- [ ] `BACKSLASH_IN_FILENAME` is defined and the filename contains \\
- [ ] Either `MSWIN` or `DJGPP` is defined and the filename contains ~
- [ ] The filename contains `..` or `//`
- [ ] The filename does not contain `..` or `//`
- [ ] The function `vim_isAbsName(fname)` returns >0
- [ ] The function `vim_isAbsName(fname)` returns 0
- [ ] I don't know

Fewer correct answers for
more #ifdefs

More correct answers for

- ***more nesting,***
- ***more negation,***
- ***longer functions,***
- ***higher percentage of annotated lines***

No consistent effect:

- number of feature constants

Preliminary Results
Ratings of Code Quality

```
1      char_u  *
2  fix_fname(fname)
3      char_u  *fname;
4  {
5  #ifdef UNIX
6      return FullName_save(fname, TRUE);
7  #else
8      if (!vim_isAbsName(fname)
9              || strstr((char *)fname, "..") != NULL
10             || strstr((char *)fname, "//") != NULL
11 # ifdef BACKSLASH_IN_FILENAME
12             || strstr((char *)fname, "\\\\") != NULL
13 # endif
14 # if defined(MSWIN) || defined(DJGPP)
15             || vim_strchr(fname, '~') != NULL
16 # endif
17             )
18         return FullName_save(fname, FALSE);
19
20     fname = vim_strsave(fname);
21
22 # ifdef USE_FNAME_CASE
23 #  ifdef USE_LONG_FNAME
24     if (USE_LONG_FNAME)
25 #  endif
26     {
27         if (fname != NULL)
28             fname_case(fname, 0);
29     }
30 # endif
31
32     return fname;
33 #endif
34 }
```

4. Do you consider the use of preprocessor annotations in the example appropriate? [CD12]
Preprocessor annotations are directives like #if and #ifdef.

- ○ Yes
- ○ No, because

6. Please rate the presented code regarding the following questions: [CD32]

	very hard	moderately hard	moderately easy	very easy
How easy was it to understand this code?	○	○	○	○
How easy would it be to maintain this code (e.g., to change code or fix bugs)?	○	○	○	○
How easy would it be to extend this code?	○	○	○	○
How easy would it be to detect bugs in this code?	○	○	○	○

No significant correlations
between #ifdef metrics and ratings.

Preliminary Results
Qualitative Assessments

Emacs

- Just ugly, makes me uncomfortable
- Please stop the torture :-)

Vim

- JESUS THAT'S A LOT OF IFDEFS
- I've written a lot of C code in my life,
  and maintained far more,
  and **this is just way messy.**
- I'm now considering giving up using VIM.

Preliminary Results
Were the refactorings beneficial?

Outcome	Significant?	Effect
Correctness: comprehension task 1	not significant	(negative tendency)
Correctness: comprehension task 2	significant ($p<0.002$)	negative: -7% (59% vs. 52% correct answers)
Rating of general quality (understandability, maintainability, …)	significant ($p<0.05$)	negligible positive effect (Cliff's delta ~0.05—0.07)
Is #ifdef use appropriate?	significant ($p<0.001$)	positive: +12% (51% vs. 63% positive ratings)

Comparing refactored to original functions …

1. #ifdef use was rated ***more appropriate,*** but …
2. Performance in comprehension tasks was ***worse!***

Conclusion & Future Work

- Large-scale questionnaire on effect of preprocessor annotations on code comprehension
- Preliminary results are inconclusive
- Possibly interesting: Actual developer performance may not align with subjective ratings.
- Future work
  - Publish data set
  - Finish data cleaning & analysis
  - Follow-up interviews to gain more insights