Flávio Medeiros, Christian Kästner, Márcio Ribeiro, Sarah Nadi, and Rohit Gheyi

The Love/Hate Relationship with the C Preprocessor: An Interview Study
29th European Conference on Object-Oriented Programming (ECOOP 2015)

Valentin Rothberg
https://www4.cs.fau.de/~vrothberg/
The Authors

Flávio Medeiros\textsuperscript{1}, Christian Kästner\textsuperscript{2}, Márcio Ribeiro\textsuperscript{3}, Sarah Nadi\textsuperscript{4}, and Rohit Gheyi\textsuperscript{1}

\textsuperscript{1} Federal University of Campina Grande, Brazil
\textsuperscript{2} Carnegie Mellon University, USA
\textsuperscript{3} Federal University of Alagoas, Brazil
\textsuperscript{4} Technische Universität Darmstadt, Germany
European Conference on Object-Oriented Programming

- Annual conference since 1986 (Paris)
- OOP systems, languages and applications
- Wide range of topics, tracks, workshops etc.
- Sister conference in North America: OOPSLA

“Historically ECOOP has combined the presentation of academic papers with comparatively practical experience reports, panels, workshops and tutorials.”
ECOOP 2016: The Workshop Armada

11 workshops on various topics:

- Context-Oriented Programming
- Formal Techniques for Java-like Programs
- The Grace Programming Language
- Implementation, Compilation, Optimization of Object-Oriented Languages, Programs and Systems
- Aliasing, Capabilities and Ownership
- Tools for JavaScript Analysis
- Live Programming Systems
- Programming Models and Languages for Distributed Computing
- Programming Experience
- Script To Program Evolution
- Runtime Verification
Paper Overview

Problem statement:

- The C Preprocessor (CPP) has received strong criticism
  - Lack of separation of concerns
  - Error proneness
  - Obfuscation of source code
- Academia proposed alternatives
  - Syntactical preprocessors
  - Aspect-oriented programming
- Developers are continuously using CPP

Core question:

- How do practitioners (i.e., “real world”) perceive the CPP?
- In other words: Are we (i.e., academia) on the right track?
Before discussion, let’s summarize the paper
An Interview Study

Study setup:
- Interview of 40 developers
- Cross validation with
  - a survey among 202 developers
  - results mined from software repositories
  - prior studies

“Our study is designed to elicit the perception of developers by talking to them.”
RQ1: Why is the CPP still widely used in practice?

- **Portability**
  support multiple platforms and systems

- **Variability**
  alternative or optional implementations, features, modules

- **Code optimizations**
  highly compiler-dependent

- **Code evolution**
  grace period for deprecated code

- **Language limitations**
  include guards
RQ2: What do developers consider as alternatives to CPP directives? (1)

Guidelines for structuring code

- Too diverse answers in interview
- Question has been moved to survey

```c
void function (){ 
#define OS1 
  /* Code 1 here.. */
#endif

#define OS2
  /* Code 2 here.. */
#endif
}

(a)

// FILE: OS1.c
void function (){ 
  /* Code 1 here.. */
}

// FILE: OS2.c
void function (){ 
  /* Code 2 here.. */
}

(b)
```

In (b), only OS1.c or OS2.c is compiled depending on the platform. It is controlled at makefile level.

Survey Results

- Strongly prefer (a) 9%
- Prefer (a) 21%
- It does not matter 10%
- Prefer (b) 35%
- Strongly prefer (b) 25%
RQ2: What do developers consider as alternatives to CPP directives? (2)

In-Language runtime mechanisms:

- Use runtime bindings (e.g., if statements)
- Diverse opinions:
  - As much as possible
  - As few as possible (scaling)

“Surprising” results?

```
1. if (*Y_AXIS.label.text) {
2.   #ifdef PM3D
3.     if (rot_x <= 90){
4.       double step = (end - x);
5.       // lines of code..
6.     } if (map)
7.     *t = text_angle;
8.   } #endif
9.   // lines of code..
10. }
```

```
1. int PM3D_RT = 0;
2. #ifdef PM3D
3.   PM3D_RT = 1;
4. #endif
5. if (*Y_AXIS.label.text) {
6.   if (PM3D_RT && rot_x <= 90){
7.     double step = (end - x);
8.     // lines of code..
9.   } if (map)
10.   *t = text_angle;
11. } // lines of code..
12. }
```

Survey Results

<table>
<thead>
<tr>
<th></th>
<th>Strongly prefer (a)</th>
<th>Prefer (a)</th>
<th>It does not matter</th>
<th>Prefer (b)</th>
<th>Strongly prefer (b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Survey Results</td>
<td>43%</td>
<td>32%</td>
<td>6%</td>
<td>14%</td>
<td>5%</td>
</tr>
</tbody>
</table>
RQ2: What do developers consider as alternatives to CPP directives? (3)

No alternative or general replacement:

- Sometimes code **must** be removed
- Alternatives would end-up as a CPP
- Using CPP is **portable**
RQ3: What are common problems of using CPP directives in practice? (1)

**Preprocessor related bugs:**

- Incorrect macro expansion
- Misspelled macro names
- Missing/undefined variables and functions
- Syntax and linking errors
- Behavioral changes due to macro interactions
- Memory and resources leaks
- Memory corruption and race conditions, ...

“[...] code that does not compile is easy to deal with, but the runtime bugs are the harder ones to detect.”
RQ3: What are common problems of using CPP directives in practice? (2)

Combinatorial testing:

“[...] code that does not compile is easy to deal with, but the runtime bugs are the harder ones to detect.”

- Finding the configuration(s) is not trivial
- “Combinatorial explosion”
- The more macros, the bigger the testing matrix

Solution?

- Check only a few configurations
- Check only a default configuration (optionals activated)
- Different compilers on different platforms
RQ3: What are common problems of using CPP directives in practice? (3)

**Code comprehension:**

- Harder to read and understand
- Mix of languages:
  - C/C++ (if, else, for, while, switch)
  - CPP (#ifdef, #ifndef, #elif)
- Deep nesting of #ifdef blocks
RQ4: Do developers care about the discipline of preprocessor annotations?

- **Yes**: impact on code quality
- Some (would) use it but document their intentions
- Refactoring: “I am not going to touch that” :-)

Repository mining:

- 21 (7%) of 299 developers introduced 85% of undisciplined annotations
- “[...] some got defensive and excused”
Conclusion and Implications for Practitioners and Researchers

Guidelines and enforcement:
- It’s done for good reasons
- Only few tools to enforce CPP related guidelines

Quality assurance:
- Configurations are rarely tested systematically or even exhaustively
- Systematic sampling and family-based analyses are promising directions

Tool design and technology transfer:
- CPP’s portability makes alternatives hard to establish
- Research should communicate better
What we should keep in mind (imho):

- Love it or hate it: the CPP will be around
- Plenty of research has happened, but we’re not done (yet)
- Investing into new research/tools will pay off
- Alternatives to CPP will be hard to establish
Paul A. Karger, and Roger R. Schell

Thirty Years Later: Lessons from the Multics Security Evaluation
Proceedings of the 18th Annual Computer Security Applications Conference (ACSAC ’02)

by
Christian Dietrich
https://www4.cs.fau.de/~stettberger/