#ifdef Directives and Program Comprehension: The Dilemma between Correctness and Preference

Wolfram Fenske[1] Jacob Krüger[2] Maria Kanyshkova[2] Sandro Schulze[2]

Abstract: In this extended abstract, we summarize our paper with the homonymous title published at the International Conference on Software Maintenance and Evolution (ICSME) 2020 [Fe20].

Keywords: Configurable Systems; Preprocessors; Program Comprehension; Refactoring; Empirical Study

The C PreProcessor (CPP) is a simple, yet effective tool to implement configuration options in a software system. For this purpose, the CPP provides text-based directives to enable conditional compilation, following the annotate-and-remove paradigm. Each directive is associated with a macro (i.e., the configuration option), which controls the presence or absence of the source code surrounded by its opening (e.g., #ifdef) and closing (e.g., #endif) directives. Due to its simplicity, the CPP is widely used in industrial and open-source systems from various domains—prominent examples being the Linux Kernel with over 26 million lines of code and more than 15 thousand configuration options, Hewlett-Packard's printer firmware, and the Apache web server. The CPP allows developers to customize such systems to specific customer requirements, safety regulations, resource restrictions, or non-functional properties.

While the CPP is established in practice, it is also heavily criticized for several issues perceived as problematic. For instance, researchers suspect that the CPP impedes program comprehension, fosters code scattering as well as tangling, harms maintainability, and increases fault proneness. The most prominent issue that has been investigated in greater detail are *undisciplined CPP directives*, that is, directives that are not aligned with syntactic units in the source code. However, some studies on the CPP led to contradicting results and most studies are limited in their validity, for example, because they exclusively rely on automated repository mining or because controlled experiments involved mostly a smaller number of students. Only two previous experiments (one on undisciplined directives, one on faults) involved a larger number of experienced practitioners.

In our paper, we present a large-scale empirical study on the impact of refactoring CPP directives to be more comprehensible. We selected five real-world code example from `Emacs` and `Vim` that previous work indicates to be particularly "smelly" (i.e., hard to comprehend). Building on findings of previous studies, we employed three types of refactoring to improve

[1] pure-systems GmbH, Magdeburg, Germany, Email: wolfram.fenske@pure-systems.com
[2] Otto-von-Guericke-University Magdeburg, Germany, Email: jkrueger@ovgu.de, sanschul@ovgu.de

doi:10.18420/SE2021_06

the comprehensibility of the source code by reducing the complexity of the present CPP directives. We then designed an online study that comprised an experiment and a survey, combining objective and subjective empirical data for the same code examples—which has not been done in previous work that always focused on either experimental or survey data. Our study was split into two different versions, both sharing one code example to assure that we could compare between the versions. Moreover, each version comprised two original and two refactored code examples. For each example, our participants solved two program comprehension tasks (i.e., the experiment) during which we measured their objective correctness. Afterwards, we asked them to assess the quality of the code and CPP directives (i.e., the survey) to elicit their subjective opinion. We sent our study to 7,791 C developers of open-source projects hosted on GitHub who made their data publicly available. Overall, we received 521 responses with an almost even split between the two study versions (i.e., 260 to 261). Using this methodology, we considerably extend previous studies by combining objective and subjective measurements in a large-scale study.

The core findings we derive from our results are:

- Our participants *performed slightly worse on refactored code* in terms of correctly solving the defined program-comprehension tasks, despite existing evidence suggesting that the refactoring should have improved their program comprehension.
- Our participants *preferred the refactored CPP directives* over those in the original code examples, aligning with existing evidence.
- Most interestingly, our *participants' objective comprehension performance and their subjective preferences contradict each other and existing evidence* on the comprehensibility and refactoring of CPP directives.
- Refactoring CPP directives may result in developers perceiving the overall code quality as worse, indicating a *trade-off between the quality of the CPP directives and the quality of the underlying source code.*

Overall, our results imply a surprising dilemma not covered by previous studies, challenging common beliefs in the context of program comprehension of CPP directives. In our future work, we will investigate this dilemma between objective performance and subjective preference in more detail using further empirical research methods.

Bibliography

[Fe20] Fenske, Wolfram; Krüger, Jacob; Kanyshkova, Maria; Schulze, Sandro: #ifdef Directives and Program Comprehension: The Dilemma between Correctness and Preference. In: International Conference on Software Maintenance and Evolution. ICSME. IEEE, pp. 255–266, 2020.