Analyzing the Discipline of Preprocessor Annotations in 30 Million Lines of C Code

Jörg Liebig
University of Passau
joliebig@fim.uni-passau.de

Christian Kästner
Philipps-University Marburg
kaestner@informatik.uni-marburg.de

Sven Apel
University of Passau
apel@fim.uni-passau.de

ABSTRACT

The C preprocessor *cpp* is a widely used tool for implementing variable software. It enables programmers to express variable code (which may even crosscut the entire implementation) with conditional compilation. The C preprocessor relies on simple text processing and is independent of the host language (C, C++, Java, and so on). Language-independent text processing is powerful and expressive—programmers can make all kinds of annotations in the form of `#ifdef`s—but can render unpreprocessed code difficult to process automatically by tools, such as refactoring, concern management, and variability-aware type checking. We distinguish between disciplined annotations, which align with the underlying source-code structure, and undisciplined annotations, which do not align with the structure and hence complicate tool development. This distinction raises the question of how frequently programmers use undisciplined annotations and whether it is feasible to change them to disciplined annotations to simplify tool development and to enable programmers to use a wide variety of tools in the first place. By means of an analysis of 40 medium-sized to large-sized C programs, we show empirically that programmers use *cpp* mostly in a disciplined way: about 84 % of all annotations respect the underlying source-code structure. Furthermore, we analyze the remaining undisciplined annotations, identify patterns, and discuss how to transform them into a disciplined form.

Categories and Subject Descriptors

D.2.3 [**Software Engineering**]: Coding Tools and Techniques; D.2.8 [**Software Engineering**]: Metrics; D.3.4 [**Programming Languages**]: Processors—*Preprocessors*

General Terms

Languages

Keywords

preprocessor, ifdef, conditional compilation, virtual separation of concerns, crosscutting concerns

AOSD'11, March 21–25, 2011, Pernambuco, Brazil.

1. INTRODUCTION

The preprocessor *cpp* is a text processing tool that extends the programming language C by lightweight metaprogramming facilities [24]. It was originally designed for the programming language C and is nowadays also part of C++ and used with several other languages such as Fortran. The preprocessor provides three capabilities: file inclusion, textual substitution (macro substitution), and conditional inclusion (a.k.a. conditional compilation). Here, we concentrate on conditional inclusion and problems related to this capability [8, 26].

Conditional inclusion allows programmers to selectively include source code. To this end, a programmer *annotates* source code using the preprocessor directives `#ifdef`, `#ifndef`, and so on, which wrap lines of source code to make them optional. Programmers influence the inclusion of annotated code with configuration files or compiler flags and, to this end, generate different program variants, some of which include certain code fragments and some not. The application of conditional inclusion is not limited to a single file. Programmers use it for the implementation of features (end-user visible concerns) that often crosscuts the entire code base [26].

In academia, contemporary textual preprocessors are heavily criticized as error prone and as rendering code hard to read and maintain [1, 11, 10, 27, 35]. Instead of separating concerns, with preprocessors, developers often implement concerns with many small annotated code fragments scattered across the code base. There are two common suggestions to deal with this situation: The first is to *refactor concerns* and replace conditional compilation my means of contemporary language concepts that support crosscutting implementations, such as aspects, mixin layers, or feature modules; for example, a large body of research addresses the potential of refactoring `#ifdef` directives into aspects [1, 5, 6, 27, 32]. The second suggestion—called *concern management* or *virtual separation of concerns*—is to keep but explicitly manage scattered preprocessor implementations, often with additional tool support, for example, in the form of views on selected concerns, visualizations, and preprocessor-aware type systems [11, 15, 16, 17, 21, 30, 34].

Both approaches, concern refactoring and concern management, rely on an integrated analysis of the source-code structure and the effect of preprocessor directives. However, parsing and analyzing the unprocessed representation of the source code (pre-cpp) is known to be hard, because the preprocessor *cpp* is token-based and as such oblivious to the underlying source-code structure. Developers may annotate