ORIGINAL ARTICLE

Journal Section

How do Programmers Maintain Concurrent Code

Feiyue Yu[1]* | **Hao Zhong**[2]* | **Beijun Shen**[1]†

[1]School of Software, Shanghai Jiao Tong University, Shanghai, 200240, China

[2]Department of Computer Science, Shanghai Jiao Tong University, Shanghai, 200240, China

Correspondence
Feiyue Yu, School of Software, Shanghai Jiao Tong University, Shanghai, 200240, China
Email: yufeiyue@sjtu.edu.cn

Funding information
Funder One, Funder One Department, Grant/Award Number: 123456, 123457 and 123458; Funder Two, Funder Two Department, Grant/Award Number: 123459

Concurrent programming is pervasive in nowadays software development. Many programmers believe that concurrent programming is difficult, and maintaining concurrency code is error-prone. Although researchers have conducted empirical studies to understand concurrent programming, they still rarely study how programmers maintain concurrent code. To the best of our knowledge, only a recent study explored the modifications on critical sections, and many related questions are still open. In this paper, we conduct an empirical study to explore how programmers maintain concurrent code. We analyze more concurrency-related commits and explore more issues such as the change patterns of maintaining concurrent code than the previous study. We summarize five change patterns according to our analysis on 696 concurrency-related commits. We apply our change patterns to three open source projects, and synthesize three pull requests. Until now, two of them have been accepted. Our results can be useful for programmers to maintain concurrent code and for researchers to implement treating techniques.

KEYWORDS
keyword 1, *keyword 2*, keyword 3, keyword 4, keyword 5, keyword 6, keyword 7

Abbreviations: ABC, a black cat; DEF, doesn't ever fret; GHI, goes home immediately.

* Equally contributing authors.

1 | INTRODUCTION

Many practitioners and researchers believe that the software maintenance phase is one of the most expensive phases, in the life cycle of a software system. Some reports (*e.g.*, [**?**]) claim that the software maintenance phase accounts for almost 80% of the whole budget. With the maintenance of software, many revision histories are accumulated [**?**]. Based on such revision histories, researchers have conducted various empirical studies to understand how programmers maintain code (*e.g.*, evolution of design patterns [**?**], fine-grained modifications [**?**], and the evolution of APIs [**?**]). These empirical studies deepen our understanding on software maintenance, and provide valuable insights on how to maintain code in future development.

In recent years, to fully leverage the potential of multi-core CPUs, concurrent programming has become increasingly popular [**?**]. For example, Pinto *et al.* [**?**] investigated 2,227 projects, and their results show that more than 75% of these projects employ some concurrency control mechanism. Despite of its popularity, many programmers find that concurrent programming is difficult [**?**], and often introduce relevant bugs in their code [**?**]. As it is difficult to maintain concurrent code, there is a strong need for a thorough empirical study on how programmers maintain such code. Despite of its importance, the topic is still rarely explored. To the best of our knowledge, only a recent study [**?**] was conducted to understand how programmers maintain concurrent code. Although the study is insightful and explores many aspects of concurrent programming, it is still incomplete. For example, their study sampled only 25 concurrency-related commits, and focuses on limited topics such as over synchronization and how concurrency bugs originate. As a result, many relevant questions are still open. For example, are there any patterns, when programmers maintain concurrent code? Indeed, such patterns are useful for programmers when they maintain code. For example, Santos *et al.* [**?**] have explored the change patterns during software maintenance, and their results show that extracted change patterns can be applied to new code locations. However, their study does not touch the change patterns of concurrent code. A more detailed analysis can have the following benefits:

Benefit 1. The results can deepen the knowledge on how to maintain concurrent code. Due to the complexity of concurrent programming, we find that even experienced developers can be confused when they maintain relevant code. For example, Mark Thomas is a member of the Apache Tomcat Project Management Committee[1], and senior software engineer at the Covalent division of SpringSource[2]. He contributed more than 10,000 commits to Tomcat. In a commit message, he left the complaint as follow:

commit a6092d771ec50cf9aa434c75455b842f3ac6c628 Threading / initialisation issues. Not all were valid. Make them volatile anyway so FindBugs doesn't complain.

In this example, we find that even experienced programmers can have problems in understanding their own code changes, when they maintain concurrent code. Our results can resolve such confusions.

Benefit 2. The results can be useful to improve existing tools. For example, Meng *et al.* [**?**] proposed an approach that applies changes systematically based on a given example. With extensions, it can be feasible to apply our extract change patterns to update concurrent code. We further discuss this issue in Section **??**.

However, to fulfill the above benefits, we have to overcome the following challenges:

Challenge 1. To ensure the reliability of our result, we have to collect many code changes that are related to concurrent programming. It is tedious to manually collect many related code changes for analysis. Tian *et al.* [**?**] worked on a similar research problem. They proposed an approach that identifies bug fixes from commits. Their results show that even advanced techniques can fail to identify many desirable commits.

Challenge 2. The changes on concurrent code can be complicated. A recent study [**?**] showed that only 38% commits

[1] `http://tomcat.apache.org/whoweare.html`

[2] `https://sourceforge.net/projects/covalent/`

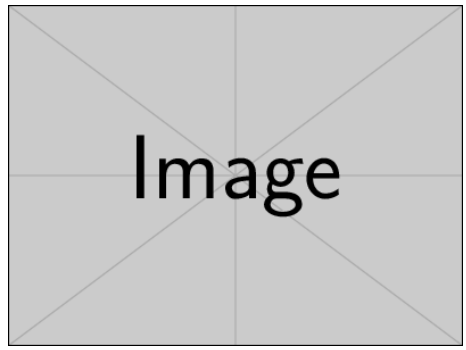


FIGURE 1 Although we encourage authors to send us the highest-quality figures possible, for peer-review purposes we are can accept a wide variety of formats, sizes, and resolutions. Legends should be concise but comprehensive – the figure and its legend must be understandable without reference to the text. Include definitions of any symbols used and define/explain all abbreviations and units of measurement.

are compilable. To analyze code that is not compilable, researchers typically use partial-code tools such as PPA [**?**] and ChangeDistiller [**?**] to analyze commits. However, as partial programs lose information, partial-code tools are imprecise and typically do not support advanced analysis. Furthermore, as we do not know what patterns can be followed, it is difficult to implement an automatic tool. As a result, it is inevitable to take much human effort when we conduct the empirical study.

In this paper, we conduct an empirical study on 98,325 commits that are collected from six popular open-source projects. To reduce the effort of manual inspection, we implement a set of tools that collect and identify concurrency-related commits automatically (see Section **??** for details). With its support, in total, we identified 11,868 concurrency-related commits, and manually analyzed 696 such commits. Based on our results, this paper makes the following contributions:

- The first analysis on the change patterns of maintaining concurrent programs. Based on our results, we summarize five change patterns, and we present their examples for explanation. We find that following such change patterns, during software maintenance, programmers can modify concurrent code to repair bugs, improve performance, and change functions of their code. Furthermore, we find that maintaining concurrent code is not a one-direction migration. Due to various considerations, programmers can apply seemingly contradictory changes, and even revert their changes. Sometimes, programmers can even make changes, before they fully understand the consequences of their changes.
- An application of our change patterns in real code. In particular, we search the latest versions of three projects for chances to apply our change patterns, and synthesize three pull requests according to our change patterns. Two of our pull requests are already confirmed and accepted by their programmers. However, our results also reveal that it needs much experience and understanding to leverage our change patterns.

2 | INTRODUCTION

Please lay out your article using the section headings and example objects below, and remember to delete all help text prior to submitting your article to the journal.

2.1 | Second Level Heading

If data, scripts or other artefacts used to generate the analyses presented in the article are available via a publicly available data repository, please include a reference to the location of the material within the article.

This is an equation, numbered

$$\int_0^{+\infty} e^{-x^2} dx = \frac{\sqrt{\pi}}{2} \tag{1}$$

And one that is not numbered

$$e^{i\pi} = -1$$

2.2 | Adding Citations and a References List

Please use a `.bib` file to store your references. When using Overleaf to prepare your manuscript, you can upload a `.bib` file or import your Mendeley, CiteULike or Zotero library directly as a `.bib` file[3]. You can then cite entries from it, like this: [1]. Just remember to specify a bibliography style, as well as the filename of the `.bib`.

You can find a video tutorial here to learn more about BibTeX: `https://www.overleaf.com/help/97-how-to-include-a-bibliography-using-bibtex`.

This template provides two options for the citation and reference list style:

Numerical style Use `\documentclass[...,num-refs]{wiley-article}`
Author-year style Use `\documentclass[...,alpha-refs]{wiley-article}`

2.2.1 | Third Level Heading

Supporting information will be included with the published article. For submission any supporting information should be supplied as separate files but referred to in the text.

Appendices will be published after the references. For submission they should be supplied as separate files but referred to in the text.

Fourth Level Heading

> *The significant problems we have cannot be solved at the same level of thinking with which we created them.*[1]
>
> *Anyone who has never made a mistake has never tried anything new.*
>
> *Albert Einstein*

Fifth level heading Measurements should be given in SI or SI-derived units. Chemical substances should be referred to by the generic name only. Trade names should not be used. Drugs should be referred to by their generic names. If proprietary drugs have been used in the study, refer to these by their generic name, mentioning the proprietary name, and the name and location of the manufacturer, in parentheses.

[3] see `https://www.overleaf.com/blog/184`

TABLE 1 This is a table. Tables should be self-contained and complement, but not duplicate, information contained in the text. They should be not be provided as images. Legends should be concise but comprehensive – the table, legend and footnotes must be understandable without reference to the text. All abbreviations must be defined in footnotes.

Variables	JKL ($n = 30$)	Control ($n = 40$)	MN	t (68)
Age at testing	38	58	504.48	58 ms
Age at testing	38	58	504.48	58 ms
Age at testing	38	58	504.48	58 ms
Age at testing	38	58	504.48	58 ms
stop alternating row colors from here onwards				
Age at testing	38	58	504.48	58 ms
Age at testing	38	58	504.48	58 ms

JKL, just keep laughing; MN, merry noise.

ACKNOWLEDGEMENTS

Acknowledgements should include contributions from anyone who does not meet the criteria for authorship (for example, to recognize contributions from people who provided technical help, collation of data, writing assistance, acquisition of funding, or a department chairperson who provided general support), as well as any funding or other support information.

CONFLICT OF INTEREST

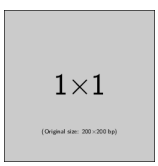
You may be asked to provide a conflict of interest statement during the submission process. Please check the journal's author guidelines for details on what to include in this section. Please ensure you liaise with all co-authors to confirm agreement with the final statement.

ENDNOTES

[1] Albert Einstein said this.

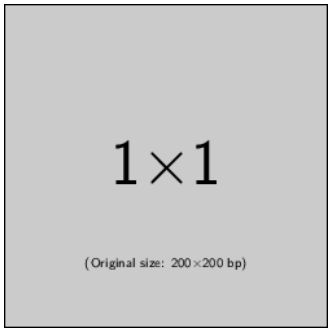
REFERENCES

[1] Lees-Miller J, Hammersley J, Wilson R. Theoretical maximum capacity as benchmark for empty vehicle redistribution in personal rapid transit. Transportation Research Record: Journal of the Transportation Research Board 2010;(2146):76–83.



A. ONE Please check with the journal's author guidelines whether author biographies are required. They are usually only included for review-type articles, and typically require photos and brief biographies (up to 75 words) for each author.

GRAPHICAL ABSTRACT



Please check the journal's author guildines for whether a graphical abstract, key points, new findings, or other items are required for display in the Table of Contents.