# Bare Demo of IEEEtran.cls for IEEE Conferences

Michael Shell
School of Electrical and
Computer Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332–0250
Email: http://www.michaelshell.org/contact.html

Homer Simpson
Twentieth Century Fox
Springfield, USA
Email: homer@thesimpsons.com

James Kirk
and Montgomery Scott
Starfleet Academy
San Francisco, California 96678–2391
Telephone: (800) 555–1212
Fax: (888) 555–1212

***Abstract*—The abstract goes here.**

## I. Introduction

Concurrent programs are pervasive in nowadays software development activities. Using concurrency rightly in programs can exploit the calculation ability better with the rapid development of multi-core system. However, concurrent programs are known hard to write correctly for multiple threads accessing objects simultaneously or depending on each other usually need complex synchronization and hard to debug for the uncertainty of thread interleaving which makes it difficult to reproduce the bug. Developers often struggle with various of synchronization methods and subtle concurrent bugs. There are many researches of concurrent programming in the literature such as data race detection, atomicity violation detection or deadlock detection.

Software projects evolves during years because of new functionalities, bugs, reorganization of code. A few of open source software platforms like github has been more and more popular in recent years. They hold a huge amount of software projects and their historic versions. Researchers have shown that software evolution history can provide much useful information for today's software development activities. Many studies focus on topics of software evolution such as refactoring, transformation patterns. Gustavo Santos studied system specific, source code transformations. Rui Gu studied change history of thread synchronization. Gustavo Pinto did a large-scale study on the usage of Javas concurrent programming constructs.

We studied concurrent programs from a perspective of software evolution history and found many change patterns about concurrent programming.

However, this work has to face several challenges:

1. The scale of open source software is increasing explosively as a result of some open source code platforms have become more and more popular. The change history of the open source software is also vast. Our interest is concurrent related commit, but they are hidden in the massive commit history. It requires much time and effort to identify whether a commit is concurrent related or not if doing it manually. We would like to adopt some automatic methods.

2. The changes of code usually have complex relationship with the context not only in the file where change happens but also other files. Some change patterns have implicit dependency on the existing code. This raises a challenge to identify real change patterns which can be applied to other context correctly.

Our main contributions are:

1. We use machine learning algorithm to select concurrent related commits effectively.

2. We identify and classify change patterns in concurrent code and find some interesting findings.

3. We give some inspirations to concurrent program or library developers and analysis tool developers.

The rest of paper is organized as follows: Section 2 presents the methodology of our study. Section 3 presents our result and discussion. Section 4 presents related work. Section 5 presents future work and Section 6 concludes.

## II. Methodology

This section presents the project sources of our study, research questions and methods of doing the study. We have developed a tool supporting the empirical study.

### A. Data set

We investigate 8 Java open-source projects from Github including Hadoop, Tomcat, Cassandra, Lucene-solr, Netty, Flink, Guava and Mahout as shown in Table 1. They are all popular, large-scale, active, representative Java open-source projects and cover different areas like distributed computing, web server, database, information retrieval, I/O and machine learning. The Hadoop project develops open-source software for reliable, scalable, distributed computing and has become one of the most famous Java open-source software for many years. Tomcat is the most popular implementation of the Java Servlet, JavaServer Pages, Java Expression Language and Java WebSocket technologies. Cassandra is a database system which can Manage massive amounts of data, fast, without losing sleep. Lucene-solr is two projects together in one respository in Github. Lucene is a search engine library and solr is a search engine server which uses lucene. Netty is an event-driven asynchronous network application framework.

TABLE I
PROJECTS INFORMATION (LOC AND #FILES ARE BOTH OF JAVA FILES)

| Project | LOC | #Files | #Commits |
|---|---|---|---|
| Hadoop | 1202764 | 7701 | 14930 |
| Tomcat | 301173 | 2192 | 17731 |
| Cassandra | 387980 | 2143 | 21982 |
| Lucene-solr | 918398 | 6310 | 26152 |
| Netty | 218131 | 2054 | 7759 |
| Flink | 414264 | 4068 | 9771 |
| Guava | 251205 | 1672 | 3850 |
| Mahout | 109584 | 1215 | 3703 |

Flink is an open source stream processing framework with powerful stream- and batch-processing capabilities. Mahout is a machine learning project. Table 1 shows the lines of code in Java, the number of Java files and the number of commits of each project. All the projects are checked out for our study in December 2016.

### B. Research questions

In order to understand the evolution of concurrent code better, we proposed 4 research questions:

**RQ1.** How many change patterns in concurrent programming?

Change patterns

**RQ2.** How frequent do concurrent related code modification appear in different kinds of Java open-source projects?

Concurrent programming is very popular in today's Java development with the rapid developments of multi-core techniques which help exploit the power of concurrent programming. Java programming language provides convenient built-in concurrent libraries and users can also invoke third-party libraries. Although developers can use their own concurrent related classes or third-party libraries, they are always using the facilities provided by JDK by default. We want to know how frequent do concurrent related code modification in software projects. What are the differences of frequency in different kinds of software projects.

**RQ3.** What is the trend of concurrent programming construct usage statistically?

Java programming language offers many handy facilities for building concurrent programs. For example, language level constructs like synchronized and volatile are keywords of Java. There are also API level constructs like notify method of an object and some concurrent related convenient classes such as the java.util.concurrent package. There always are more than one ways to finish a task in Java and the preferences of developers evolves fast. We are interested in the trend of some common concurrent related constructs and the possible reasons hidden behind the phenomenon.

**RQ4.** How can these change patterns in history guide the development?

### C. Tool support

We have developed a tool to collect and analyze data. The tool have the following functionalities.

*1) Collecting commits:* All the projects of our study are under git which is one of the most popular version control systems in the world. Some projects of the study used svn or some other version control systems before because they have long histories, but they all support git now. We employ JGit, a lightweight, pure java library implementation of git, to retrieve all the commit logs in projects' histories. A typical commit log contains commit id which is a 20-character-long string uniquely identifying a commit, author, date and message. Once we get a commit id, we use "git show" command to show the log message and textual diff. The diff result contains one or more change files which contain one or more change hunks.

*2) Classification:* Most of the commits selected by the last step are not our targets since they usually add or modify functionalities which are specific although they contain some concurrent keywords. The target commits to be analysed occupy a very small proportion of all the commits even we have already filtered them. So, we need some automatic methods to help finish the job.

We use the SVM[**?**] method to classify commits as concurrent-related or not. SVM is a supervised classification algorithm which needs both positive and negative training data. We extracted 12 features from each commit and labeled 65 instances manually as a training data set. Testing on the training set itself has an accuracy of 98.46%. The classifier selects 96 positive instances from 9891 instances.

## III. RESULTS

### A. RQ1. How many change patterns in concurrent programming?

### B. RQ2. How frequent do concurrent related code modification appear in different kinds of Java open-source projects?

### C. RQ3. What is the trend of concurrent programming construct usage statistically?

### D. RQ4. How can these change patterns in history guide the future development?

## IV. DISCUSSION

## V. RELATED WORK

## VI. CONCLUSION

## ACKNOWLEDGMENT

The authors would like to thank...

## REFERENCES

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.