Diagnosing and Treating Code-Duplication Problems in Bioinformatics Libraries

Mohammad Shabbir Hasan
Dept. of Computer Science
Virginia Tech,
Blacksburg, VA 24061, USA
Email: shabbir5@cs.vt.edu

Saima Sultana Tithi
Dept. of Computer Science
Virginia Tech,
Blacksburg, VA 24061, USA
Email: saima5@cs.vt.edu

Eli Tilevich
Software Innovations Lab
Virginia Tech,
Blacksburg, VA 24061, USA
Email: tilevich@cs.vt.edu

Liqing Zhang
Dept. of Computer Science
Virginia Tech,
Blacksburg, VA 24061, USA
Email: lqzhang@cs.vt.edu

***Abstract*—As computing is an enabling tool of bioinformatics, software quality can influence not only the efficiency of the research process, but also the degree of confidence in scientific findings. As we discovered, popular bioinformatics C++ libraries suffer from problems that make their code hard to maintain, fine-tune, and extend. In particular, code duplication caused by the ubiquitous copy-and-paste development practice, substantially complicates software maintenance and evolution. The presence of multiple clones of the same code snippet multiples the amount of effort required to modify or extend it. In this paper, we present the results of a systematic study we have conducted to understand the code quality of popular bioinformatics libraries. Based on the results of our study, we developed an automated tool that systematically identifies and consolidates duplicated code blocks. Here we describe our tool—*ReBio*[1]—and the results of applying it to improve the quality of several commonly used C++ libraries, including SeqAn, BEDtools, and NCBI C++ Toolkit. Our results reveal that these libraries indeed suffer from poor maintainability, and that our automated tool can effectively improve their quality.**

***Keywords*—Software Maintenance; Code Clones/Duplication; Program Analysis; Refactoring; Bioinformatics Libraries**

I. Introduction

Bioinformatics software is a powerful tool in our quest for scientific discovery. Therefore, software quality is paramount to ensuring the efficiency of the research process and our confidence in the obtained findings. Recognizing the importance of this problem, the US National Science Foundation has recently committed $35 million to improve the quality of scientific software [1]. Nevertheless, the quality of bioinformatics software is often not treated with the degree of attention this issue deserves. As a result, the bioinformatics community has to contend with a shared scientific discovery tool, whose quality has never been the focus of targeted quality assessment and improvement efforts. The goal of our work is to address this shortcoming.

A software development phenomenon, known as *code duplication*, is known to complicate software maintenance efforts. Several reasons push code duplication to the top of the factors that complicate software maintenance. Modifying a code block, copied and pasted in multiple source files, requires modifying all the code block's clones, a tedious, costly, and error-prone programming task especially for large scale codebases. Moreover, copying and pasting code fragments often introduces bugs. Therefore, the ability to detect code clones effectively is an essential step to keeping software bug free as well as facilitating various evolutionary modifications. To alleviate the process of harmful code duplication, software refactoring has been introduced as a systematic approach that can also be supported by automated program transformation tools.

Bioinformaticians rely on various libraries to create their own software solutions. Our analysis of some popular libraries, including SeqAn [2], BEDtools [3], and NCBI C++ Toolkit [4], revealed that these libraries have 34%, 17.40%, and 37% of duplicated lines respectively and can benefit from undergoing a refactoring to consolidate the duplications. The preferred implementation language for bioinformatics libraries—C++—complicates such a refactoring undertaking and automated refactoring support for C++ remains scarce. Despite the proven benefits of systematic refactoring [5], this software development process has not been applied systematically to the numerous bioinformatics software tools and libraries.

In this paper, we report on our experiences of designing, creating, and evaluating an automated tool, built with the overriding goal of improving the quality of bioinformatics C++ libraries. Our reference implementation, named ReBio, automate many of the tedious and error-prone tasks required to identify and consolidate duplicated code blocks. As our evaluation of the effectiveness of ReBio, we applied it to the BEDtools library, discovering that up to 13,083 lines of duplicated code can be removed from the library while preserving its external behavior.

II. Materials and Methods

ReBio works in two steps: in the beginning, the user selects the source directory that contains a library that needs to undergo a refactoring. ReBio then applies the PMD-CPD [6] and CCFinderX [7] tools in sequence to each source file contained in the selected directory. Although both of these third-party tools find many code blocks that have been duplicated in various fashions, not all of these blocks can be refactored. The main added value of ReBio lies in its ability to sift through the found duplicates and select only those ones

[1]ReBio is written in Java and freely available at `https://sourceforge.net/projects/rebio/`.

TABLE I
COMPARISON BASED ON REFACTORABLE CODE BLOCKS IDENTIFIED BY PMD-CPD AND CCFINDERX

Name of the library	**PMD**		**CCFinderX**	
	# of clone groups	**# of refactorable clone groups**	**# of clone groups**	**# of refactorable clone groups**
BEDtools	48	32	388	266
SeqAn	1,275	81	910	679
NCBI C++ Toolkit	10,819	7,044	14,687	10,796

TABLE II
COMPARISON BASED ON NUMBER OF CLONE GROUPS IDENTIFIED

Name of the library	**# of refactorable clone groups**		
	PMD	CCFinderX	ReBio
BEDtools	32	266	258
SeqAn	81	679	710
NCBI C++ Toolkit	7,044	10,796	17,058

that can be effectively refactored. ReBio considers a duplicated code block (clone segment) as not refactorable if it does not contain any full blocks of code (i.e., it may contain some parts of a code block that is detected as duplicated by PMD-CMD or CCFinderX). If there is no full code block in a clone segment, ReBio discards the whole clone segment. If a clone segment contains one or more full code blocks and a partial block, then ReBio extracts only the full code blocks that are refactorable and discards the partial blocks.

After selecting the refactorable code blocks from the result of PMD-CPD and CCFinderX, ReBio merges the results of these two tools. While merging, ReBio checks if a clone detected by one tool overlaps with the clone detected by the other tool. If one clone overlaps with another, ReBio only keeps one of them, and discard the overlapped one. After discarding the overlapped clones, ReBio gives the output containing all clone groups, in which all clone groups contain only refactorable and non-overlapped clones.

In our experimental evaluation, we applied ReBio to three popular bioinformatics C++ libraries: SeqAn [2], BEDtools [3], and NCBI C++ Toolkit [4]. After ReBio finds the duplicated refactorable code blocks, we manually refactor BEDtools. All the results are reported in the following section.

III. ANALYSIS RESULTS

Since both PMD-CPD and CCFinderX find duplicated code blocks without any regard for whether these blocks are refactorable, the main contribution of this research is determining which ones of them can benefit from a refactoring transformation. Hence, in presenting our results we distinguish between the output produced by these two third-party analysis tools and the further pruning of that output by ReBio. The numbers presented in Table I detail this comparison.

Table II shows the number of refactorable clone groups identified by PMD-CPD, CCFinderX, and ReBio. Although ReBio uses both PMD-CPD and CCFinderX to produce the final clone groups, the number of clone groups identified by ReBio is smaller than the total sum of clone groups identified by PMD-CPD and CCFinderX. The reason is simple: if a refactorable code block identified by one tool is also identified by another tool, ReBio includes that refactorable code block only once, so as to avoid any redundancy.

After identifying the refactorable duplicated code blocks, we then undertook a manual refactoring of the BEDtools code base. We used the well-known refactoring techniques “Extract Method” and “Pull up Method” to systematically remove the duplicated code blocks. Upon completing these refactoring transformation, we assessed the effectiveness of ReBio in its ability to reduce the number of line of source code. Result shows that ReBio can remove 13,083 duplicated lines of code.

Since software refactoring must preserve the external behavior of the refactored source code, we tested both the original and refactored versions of BEDtools. We designed and implemented test cases to check for the behavior preservation properties for the following major modules of BEDtools: bam to bed conversion, intersection of two bed files, and compute the coverage of aligned sequence. For all of these modules, we called the corresponding methods from the original as well as the refactored BEDtools library with the same input file. Then we compared the outputs generated by the original and the refactored versions of the tool. Results show that the outputs remain unchanged, thus indicating that our software refactoring preserves the original behavior with respect to the tested functionality.

IV. CONCLUSION

In this paper, we presented ReBio, a tool for refactoring bioinformatics C++ libraries. Using this tool, we identified duplicated code blocks in the source code that are refactorable. These results indicate that the software quality of these libraries can be substantially improved by systematic refactoring, a software process that can increase the comprehensibility of the libraries’ architecture, design, and implementation, thereby streamlining their maintainability and evolution.

REFERENCES

[1] “NSF commits $35 million to improve scientific software,” http://www.nsf.gov/news/news_summ.jsp?cntn_id=189347&WT.mc_id=USNSF_51&WT.mc_ev=click, [Online; accessed 1-August-2016].

[2] A. Döring, D. Weese, T. Rausch, and K. Reinert, “SeqAn an efficient, generic C++ library for sequence analysis,” *BMC Bioinformatics*, vol. 9, no. 1, p. 11, 2008.

[3] A. R. Quinlan and I. M. Hall, “BEDTools: a flexible suite of utilities for comparing genomic features,” *Bioinformatics*, vol. 26, no. 6, pp. 841–842, 2010.

[4] D. Vakatov, K. Siyan, and J. Ostell, “The NCBI C++ Toolkit [Internet] National Library of Medicine,” *National Center for Biotechnology Information, Bethesda (MD)*, 2003.

[5] J. Archuleta, E. Tilevich, and W.C. Feng, “A Maintainable Software Architecture for Fast and Modular Bioinformatics Sequence Search, ” in *IEEE International Conference on Software Maintenance (ICSM)*, 2007, pp. 144–153.

[6] “PMD-CPD (PMD Copy/Paste Detector) Tool,” https://pmd.github.io/, [Online; accessed 28-May-2016].

[7] Y. Higo, T. Kamiya, S. Kusumoto, and K. Inoue, “ARIES: refactoring support tool for code clone,” *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 4, pp. 1–4, 2005.