ASE 2013

Test of Time and Impact Award

Matthew Lease

Improving Bug Localization using Structured Information Retrieval

Ripon K. Saha* Matthew Lease† Sarfraz Khurshid* Dewayne E. Perry*
*Department of Electrical and Computer Engineering
†School of Information
The University of Texas at Austin, USA
ripon@utexas.edu, ml@ischool.utexas.edu, khurshid@ece.utexas.edu, perry@mail.utexas.edu

***Abstract*—Locating bugs is important, difficult, and expensive, particularly for large-scale systems. To address this, natural language information retrieval techniques are increasingly being used to suggest potential faulty source files given bug reports. While these techniques are very scalable, in practice their effectiveness remains low in accurately localizing bugs to a small number of files. Our key insight is that structured information retrieval based on code constructs, such as class and method names, enables more accurate bug localization. We present BLUiR, which embodies this insight, requires only the source code and bug reports, and takes advantage of bug similarity data if available. We build BLUiR on a proven, open source IR toolkit that anyone can use. Our work provides a thorough grounding of IR-based bug localization research in fundamental IR theoretical and empirical knowledge and practice. We evaluate BLUiR on four open source projects with approximately 3,400 bugs. Results show that BLUiR matches or outperforms a current state-of-the-art tool across applications considered, even when BLUiR does not use bug similarity data used by the other tool.**

***Index Terms*—Bug localization, information retrieval, search**

I. Introduction

Frederick Brooks wrote that "Software entities are more complex for their size than perhaps any other human construct because no two parts are alike (at least above the statement level)" [5]. Due to this inherent complexity of software construction, software bugs remain frequent. For a large software system, the number of bugs may range from hundreds to thousands. Generally, bug fixing starts with finding relevant buggy source code. i.e., *bug localization*. However, performing this process manually for many bugs is time consuming and expensive. Therefore, effective methods for locating bugs automatically from bug reports are highly desirable.

There are two general approaches for bug localization: i) dynamically locating the bug via program execution together with such technologies as execution and data monitoring, breakpoints etc. [1]; and ii) statically locating bugs via various forms of analyses using the bug reports together with the code [15]. The dynamic approach is often time consuming and expensive. The ease of the static approach, together with its immediate recommendation, make it appealing.

In recent years, information retrieval (IR) based bug localization techniques have gained significant attention due to their relatively low computational cost and minimal external dependencies (e.g., requiring only source code and bug report in order to operate) [2]. In these IR approaches, each bug report is treated as a *query*, and the source files to be searched comprise the *document collection*. IR techniques then rank the documents by predicted relevance, returning a ranked list of candidate source files which may contain the bug. Lukins et al. [21] proposed a Latent Dirichlet Allocation (LDA) approach, while Rao et al. [29] compared a range of IR techniques: Unigram, Vector Space, Latent Semantic Analysis (LSA), LDA, Cluster Based, and various combinations. Both used a relatively small number of bugs in evaluation. Ngyuen et al. proposed *BugScout* [25], which customized LDA for bug localization. Results on several large-scale datasets showed good performance. Recently, Zhou et al. [46] proposed *BugLocator*, which combined a sophisticated TF.IDF formulation, a modeling heuristic for file length, and knowledge of previously fixed similar bugs. In a large scale evaluation of approximately 3,400 bugs over four open source projects, BugLocator showed even stronger performance than BugScout. Moreover, datasets and BugLocator's executable were made available, providing an invaluable benchmark for testing and comparing alternative IR approaches to bug localization.

Despite the empirical success of prior work, we perceive a gap today between IR community practices and techniques being applied to bug localization. For example, existing IR-based bug localization treats source code as flat text lacking structure. In fact, source code's rich structure distinguishes code constructs such as comments, names of classes, methods, and variables, etc. While ignoring such code structure simplifies the system, it also sacrifices an opportunity to exploit this structural information to improve localization accuracy. While we believe modeling source code structure is novel for bug localization, we also note that the concept of modeling document structure in IR is quite old (e.g., Google in 1998 [4] and more recent BM25F [31]).

Whereas recent prior work devised a heuristic to model program length, we discuss how the importance of length normalization was actually recognized in IR two decades ago [38] and is built-into today's baseline IR models. In the same vein, we discuss how the use of bug similarity data to improve localization is closely related to the established IR use of relevance feedback data [33]. Generalizing from this, we suspect our idea for modeling code structure is only one of the many ways in which IR-based bug location could benefit from greater interaction with the IR community. Beyond our

Saha, R. K., Lease, M., Khurshid, S., & Perry, D. E. (2013). Improving bug localization using structured information retrieval. 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE), 345–355.