Validity Concerns in Software Engineering Research

Hyrum K. Wright Miryung Kim Dewayne E. Perry
Electrical and Computer Engineering
The University of Texas at Austin
Austin, Texas 78712
hyrum_wright@mail.utexas.edu,{miryung, perry}@ece.utexas.edu

ABSTRACT

Empirical studies that use software repository artifacts have become popular in the last decade due to the ready availability of open source project archives. In this paper, we survey empirical studies in the last three years of ICSE and FSE proceedings, and categorize these studies in terms of open source projects vs. proprietary source projects and the diversity of subject programs used in these studies. Our survey has shown that almost half (49%) of recent empirical studies used solely open source projects. Existing studies either draw general conclusions from these results or explicitly disclaim any conclusions that can extend beyond specific subject software.

We conclude that researchers in empirical software engineering must consider the external validity concerns that arise from using only several well-known open source software projects, and that discussion of data source selection is an important discussion topic in software engineering research. Furthermore, we propose a community research infrastructure for software repository benchmarks and sharing the empirical analysis results, in order to address external validity concerns and to raise the bar for empirical software engineering research that analyzes software artifacts.

Categories and Subject Descriptors

D.2.0 [**Software Engineering**]: General

General Terms

Validity

Keywords

empirical study, external validity, open source software

1. INTRODUCTION

Over the past several years, software engineering researchers have taken advantage of the wealth of information available from open source software projects. Researchers have mined open source code repositories, issue trackers, mailing list archives and other artifacts to perform their empirical studies. Ostensibly, these studies are performed to learn more about the state of software development, and how it can be improved.

Many of these studies draw general conclusions about software engineering, while examining strictly *open source software*. While these observations are useful, rarely do the authors of such studies comment on the threats to validity, particularly external validity, that studying only open source software presents. Some researchers allow the pendulum to swing too far in the other direction, by explicitly disclaiming any conclusions beyond the specific projects studied. Neither approach is useful in advancing the state of the art or practice in software engineering.

In this paper, we explore potential threats to validity in software engineering research that use software artifact archives as a data set. While discussing validity generally, we specifically focus on the external validity issues relating to the choice of using open source data as a primary research data source. We survey several years worth of past FSE and ICSE conference proceedings to determine the prevalence of the perceived problem, pose questions we feel are pertinent to improving the state of empirical software engineering research, and offer suggestions as to ways in which this knowledge can improve the research methods going forward.

2. EXPERIMENTAL VALIDITY

For experiments of any type to make convincing arguments, they must possess a high degree of validity. While a complete treatment of experimental validity is better left to other sources (see [17] and [21]), this section briefly addresses challenges to internal, external, and construct validity in software engineering research.

2.1 Construct validity

Construct validity refers to whether specific measurements indeed model independent and dependent variables from which the hypothesized theory is constructed. In other words, an empirical study with high construct validity would ensure the studied parameters are relevant to the research questions.

2.2 Internal validity

Confounding factors represent a major threat to the internal validity in empirical studies. As our survey shows, selection bias is a prevalent problem in software engineering research, and limits the validity of these studies. Internal

FoSER 2010, November 7–8, 2010, Santa Fe, New Mexico, USA.

validity can be difficult to counter, since changes in the variable under observation may be attributed to the existence or variations in the degree of other variables, which are related to the manipulated variable but not explicitly modeled variables.

2.3 External validity

Generally, external validity refers to the applicability of study or experimental results to realms beyond those under immediate observation. A study is said to have a high degree of external validity if the conclusions hold throughout the study domain. In most scientific disciplines, researchers prize studies with external validity, since the results can be widely applied to other scenarios.

External validity for a given study has several aspects:

- whether the study generalizes to other subjects in the domain
- whether there exist enough evidence and arguments to support the claimed generalizability
- whether the study outcomes validate predicted theories

Mitigating construct and internal validity concerns is often more important than addressing threats to external validity as addressing the first two is a pre-requisite before considering the generalizability of the studied results beyond the subject domain in which the study was conducted. However, every software engineering study should strive for a high degree of external validity as the world of software is simply too large and too complex to study comprehensively. It is important that researchers choose representative projects to study and then generalize the results from.

3. OPEN SOURCE DATA IN RESEARCH

With the recent explosion of open source software development and data, researchers have turned to these sources for easy access to development data and artifacts [2, 4, 1]. This occurs, even in spite of the difficulties, understood or not, in doing so [12]. Obtaining a balanced set of data from open source repositories has been an issue in the open source research community for some time, and several collections of data have grown to attempt to solve this problem [11, 13, 20].

However, open source data can differ from proprietary software data in a number of ways. First, when using open source data sets, researchers often have access to the *artifacts* of the software engineering process, while when examining proprietary software, researchers can often get a more complete view of the software system and the environment under which the software systems are constructed and maintained. Second, further biasing the source data is the domain of open source software. Many of the most mature open source projects are systems-domain software, which may lead to its own set of development biases. Software development targeted toward other domains can have unique concerns which are not accurately captured by a systems-specific software bias.

Third, a large body of work exists to determine the social structure of open source projects [8]. Proprietary projects, meanwhile, already have established organization structure, and by studying them, researchers may forgo the time-intensive process of discovering social relationships. Moreover, Conway's Law posits that the design and structure of the software, whether open or proprietary, may be further biased by the structure of the organization [7]. Since the organizational structure of open source projects is often spontaneously formed, the types of research questions that can be answered about communication and organization structure using open source project data may not generalize to closed source projects where the structure is defined explicitly.

Finally, using open source project data further complicates the validation of research methods, as it is difficult to find the right personnel (developers, managers and testers) and have them validate the results of automated software analysis results as the roles of contributors are often implicit and their work is often based on voluntary commitment.

3.1 Survey of open source data in research

A brief survey of several past conferences helps illustrate our point. This survey, while not completely representative of all software engineering research, does show what the prevailing trends are at the major software engineering conferences. In this survey, we have investigated the extent of empirical studies that use only open source software artifacts (OSS) vs. propriety source software artifacts (PSS).

While the operational definition of *open source* can be somewhat fluid, we decided to use the definition of "readily available source code and development artifacts" to distinguish *open* projects from *closed* ones. In addition, open source projects exist along a continuum of *open* development practices and licenses, so this classification is of necessity subjective. In classifying the papers, we looked for papers which *used* open source data, not just those that built an open source tool or provided their tool under an open source license. Neither did we classify such papers as *open* when the authors implemented their tool as a part of an open source framework by distinguishing the data source from the various other uses or mention of open source projects discussed in the various papers. Table 1 illustrates the results of our survey regarding the use of OSS vs. PSS.

The results of our survey are illuminating. Of the recent papers at ICSE or FSE which use software projects as study subjects, nearly half use OSS data exclusively, while another 23% use PSS data. Only 15% of the papers used any combination of OSS, PSS, or custom data (which includes manufactured examples and benchmarks). We hope that the difference between OSS and PSS is not as drastic as believed, lest the validity of a large amount of software engineering research comes into question.

In doing this survey, we noticed some interesting phenomena. Phases such as "production-level code" or "real-world application" seemed to abound when speaking of open source projects. Many papers readily admitted to using projects from SourceForge as their sole source of input data, while being blissfully ignorant of the problems associated with doing so [12].

We also noticed that many authors choose to test their algorithm or process on data which they manufactured, such as sample programs and benchmarks. While such testing is an important step in the research process, results thus obtained do not lend much confidence to their external validity. (See Section 5 for a discussion of benchmarking.)

Anecdotally, the implementation languages of the subject data sources seemed quite skewed as well. For example, many study tools were written in Java, so the authors chose Java programs as their data sources. Nowhere were standard "industrial" languages found, such as Visual Basic or COBOL. While these languages are certainly dated and may not be

Conference	Total papers	Papers using data	Data source			
			open	closed	custom	combination
ICSE '07	49	39	18	9	10	2
ESEC/FSE '07	42	23	12	5	2	4
ICSE '08	56	36	17	9	5	7
FSE '08	31	19	7	5	2	5
ICSE '09	50	38	22	7	3	6
ESEC/FSE '09	38	20	10	5	2	3
Total	266	175	86	40	24	27
As percent of total with data	—	100%	49%	23%	14%	15%

Table 1: Use of open source as data sources in research papers

on the cutting edge of software engineering research, large volumes of existing systems still run on these types of platforms, and software engineering practitioners still interface and use such languages on a regular basis. Researchers interested in improving the state of software engineering practice would do well to consider projects written in languages such as these as candidates for study.

4. OPEN QUESTIONS

Given the above background information, we believe the empirical software engineering community must address the following questions to increase the external validity of studies performed primarily using open source data.

- *What threats to validity arise from relying so heavily on open source data?*
- *Do open source and proprietary development practices differ in meaningful ways?*
- *Is there sufficient variety within open source data to allow results to be generalizable?*
- *Even when validity issues exist, do authors recognize these threats, and address them?*

Program committees, reviewers and researchers can and should work together to answer these questions and increase the validity of software engineering research. By recognizing the benefits and biases of various data sources, authors can better improve the quality of their research, and address the issues of validity given the differences between proprietary and open source software development.

For instance, historically, open source teams have worked in ways that are fundamentally different from proprietary software teams [14]. Members come and go frequently, they are widely geographically distributed, and by necessity they leave persistent communication artifacts, such as email archives. These characteristics are especially relevant to the large projects often studied by software engineering researchers. In contrast, proprietary teams operate differently, and may leave much less information of use to researchers. Casual conversations, meetings, and other communication often goes unrecorded. In some settings, these interactions can comprise 75 minutes of the typical day [15].

The one resource that proprietary software development does have that open source does not, is access to the people involved. Gathering this information, while providing a much richer set of data, is also much more expensive and time consuming than simply applying automatic mining techniques to the large corpus of open source data.

Additionally, the success of open source has not been lost on proprietary software vendors. Many companies have leveraged the possibilities of increased globalization to geographically distribute teams in an effort to maximize productivity [5]. While the source code may not be available under an open source license, the development practices begin to mirror those of the open source. At the same time, successful open source projects are often sponsored, adopted, or outright bought by commercial companies. In such cases, the license may remain open, but the development structure begins to become more rigid and approach that of traditional software development processes.

5. BENCHMARKING

One final point worth noting is the use of benchmarks in software engineering research. Ours is not the first suggestion to use benchmarks in software engineering [18, 3], but previous pleas have largely been ignored, as our survey data indicates. The threats to validity are increasing, and a proper set of empirical benchmarks, along with other measures will strengthen research validity in the software engineering community.

While the software testing community has begun to use the Software-artifact Infrastructure Repository (SIR) to evaluate various testing methods [10], and many of the papers we surveyed referenced artifacts from this repository, these papers rarely took a comprehensive look at the entire suite since the SIR repository is restricted by housing software history with regression tests and faults data. SIR is a good start, but researchers need a more complete set of data for strong experimental validity.

Constructing a representative set of benchmark data is not easy, but results from other fields are encouraging. For example, the Dacapo benchmark for compiler optimization research has been adopted by both academic and industry researchers and has continued to evolve [6]. While benchmarks do have their own sets of difficulties (researchers can and will optimize research to fit the characteristics of the benchmark), we believe the benefits of these benchmarks largely outweigh the costs. The maintenance of the benchmark suite could potentially be time intensive, given the rapidly changing state of the software world.

Despite these difficulties, a proper benchmark suite should be considered a prerequisite for creating a proper baseline for valid experiments in software engineering research. Furthermore, such benchmark will facilitate comparative evaluation

of research approaches. Currently, the lack of shared benchmarks and shared analysis results on those benchmarks often forces researchers to re-implement others' prior approaches and compare their approach on the same set of subject software [9, 16], unnecessarily incurring a high cost of research method validation.

6. CONCLUSION

Software engineering research has progressed much since the early days of ICSE and FSE and the bar for empirical validation using real software systems has been raised significantly [19]. In looking at recent ICSE and FSE conferences, we have presented our concerns that the data being examined is not sufficient to warrant the implicitly-claimed generalizability of results, and that authors do not sufficiently acknowledge this fact. As a result, we have outlined a series of questions which we hope will encourage researchers to better address these issues of external validity.

Furthermore, looking forward, we presented our rationale on why we believe a properly implemented benchmark suite and shared analysis results on the suite will improve the validity measure of software engineering research. Lastly, reviewers and members of program committees bear the responsibility of policing validity concerns in the software engineering community. These prominent members of the research community must insist on a discussion of validity in the papers they review for publication. A concerted effort in this area will rise the tide which lifts all boats toward higher validity standards.

7. ACKNOWLEDGMENTS

The authors thank Christine Julien for insightful discussion on this topic, and the anonymous reviewers for their constructive criticism. Parts of this work were funded by grant NSF-CCF 1043810 and NASA grant NNX08AC48G.

8. REFERENCES

[1] Open source software engineering workshop series. In *WOSSE*, 2001-2005.

[2] Working conference on mining software repositories. In *MSR*, 2004-2010.

[3] Working conference on mining software repositories: Mining challenges. In *MSR Challenge Track*, 2006-2010.

[4] Workshop on emerging trends in free/libre/open source software research and development. In *FLOSS*, 2010.

[5] P. J. Agerfalk, B. Fitzgerald, H. H. Olsson, and E. O. Conchuir. Benefits of global software development: the known and unknown. In *ICSP 08: Proceedings of the Software process, 2008 international conference on Making globally distributed software development a success story*, pages 1–9, Berlin, Heidelberg, 2008. Springer-Verlag.

[6] S. Blackburn, R. Garner, C. Hoffmann, A. Khang, K. McKinley, R. Bentzur, A. Diwan, D. Feinberg, D. Frampton, S. Guyer, et al. The DaCapo benchmarks: Java benchmarking development and analysis. In *Proceedings of the 21st annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, page 190. ACM, 2006.

[7] M. Conway. How do committees invent. *Datamation*, 14(4):28–31, 1968.

[8] K. Crowston and J. Howison. The social structure of free and open source software development. *First Monday*, 10(2), 2005.

[9] M. D'Ambros, M. Lanza, and R. Robbes. An extensive comparison of bug prediction approaches. In *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*, pages 31 –41, 2-3 2010.

[10] H. Do, S. Elbaum, and G. Rothermel. Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact. *Empirical Software Engineering*, 10(4):405–435, 2005.

[11] I. Herraiz, D. Izquierdo-Cortazar, F. Rivas-Hernández, J. Gonzalez-Barahona, G. Robles, S. nas Dominguez, C. Garcia-Campos, J. Gato, and L. Tovar. FLOSSMetrics: Free/libre/open source software metrics. In *Proceedings of the 13th European Conference on Software Maintenance and Reengineering (CSMR). IEEE Computer Society*, 2009.

[12] J. Howison and K. Crowston. The perils and pitfalls of mining SourceForge. *Proceedings of the International Workshop on Mining Software Repositories (MSR 2004)*, pages 7–11, 2004.

[13] J. Howison and K. Crowston. FLOSSmole: A collaborative repository for FLOSS research data and analyses. *Int. J. of Information Technology and Web Engineering*, 1(3):17–26, 2006.

[14] J. Howison, K. Inoue, and K. Crowston. Social dynamics of free and open source team communications. *International Federation for Information Processing Digital Library*, 203(1), 2009.

[15] D. Perry, N. Staudenmayer, and L. Votta. People, organizations, and process improvement. *IEEE SOFTWARE*, pages 36–45, 1994.

[16] R. Robbes, D. Pollet, and M. Lanza. Replaying ide interactions to evaluate and improve change prediction approaches. In *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*, pages 161 –170, 2-3 2010.

[17] R. Rosenthal and R. Rosnow. *Essentials of behavioural research*. McGraw, 1991.

[18] S. Sim, S. Easterbrook, and R. Holt. Using benchmarking to advance research: A challenge to software engineering. In *Proceedings of the 25th International Conference on Software Engineering*, page 83. IEEE Computer Society, 2003.

[19] W. F. Tichy. Should computer scientists experiment more? *IEEE Computer*, 31(5):32–40, 1998.

[20] M. Van Antwerp and G. Madey. Advances in the sourceforge research data archive (srda). In *Fourth International Conference on Open Source Systems, IFIP 2.13 (WoPDaSD 2008)*, Milan, Italy, September 2008.

[21] R. Yin. *Case study research: Design and methods*. Sage Pubns, 2008.