A Systematic Literature Review on the Use of Patterns Related to Software Bugs

S M Sohan
Department of Computer Science
University of Calgary
Calgary, AB T2N 1N4
Email: sohan39@gmail.com

***Abstract*—Patterns are used as a shared vocabulary to quickly identify and communicate recurring problems and their solutions in the software engineering discipline. Accordingly, researchers and practitioners have used various patterns related to bugs so that bugs can be easily identified, communicated, fixed and analyzed. Given the variety of these bug patterns and their use cases, the problem is to find an effective and efficient way to use the bug patterns. In this paper, results of a systematic literature review on bug patterns is presented to answer three research questions about how and where to use the bug patterns and their known impact on software projects. The implication of this research is two-fold: it helps practitioners to learn about the different bug patterns and their impacts, and researchers can use this as a groundwork for conducting future research on the use of bug patterns.**

I. Introduction

II. Research Questions

- **RQ1 -** Where patterns in software engineering have been used related to bugs?
- **RQ2 -** How these bug patterns were established?
- **RQ3 -** What is known about the impact of these bug patterns?

III. Research Method

A. Selection Criteria

- Search term
- Year
- Data Source
- Exclusion Criteria
- Inclusion Criteria
- Keywording

B. Selected Papers

IV. Results

Bug patterns have been used primarily on three artifacts as follows: source code (e.g. Java Code of Spring), commit logs on version control systems (e.g. SVN) and bug tracking databases (e.g. JIRA). Looking at different combinations of these artifacts, the patterns are used on the following life-cycle stages of bugs: bug detection, reporting, fixing and analysis. The remainder of these section presents the results of our analysis on RQ1-3 based on these life cycle stages and artifacts that are used by the selected papers.

A. RQ1: Where patterns in software engineering have been used related to bugs?

1) Bug Detection: Detection of bugs has been the primary topic of interest in (x/y) selected papers. Several papers focused on the detection of bugs based on patterns found on the source code exclusively. Allen listed 13 general purpose bug patterns based on Java code as follows: The Rogue Tile, Null Pointers Everywhere!, The Dangling Composite, The Null Flag, The Double Descent, The Liar View Saboteur Data, The Broken Dispatch, The Impostor Type, The Split Cleaner, The Fictitious Implementation, The Orphaned Thread and The Run-On Initialization[1]. Zhang et al. identified 6 bug patterns that are specific to Aspect Oriented Programming based on AspectJ as follows: Infinite loop, Scope of Advice, Multiple Advice Invocation, Unmatched join point, Misuse of getTarget, Introduction interference [2]. These patterns are identified based on the authors' experience of using the programming languages.

Several bug detector tools are built to detect potential bugs based on static analysis of the source code or it's compiled binary output. Bug patterns are coded as rules, and the bug detector tools can find fragments of code by matching the manually coded rules against the given source code or its compiled binary output. FindBugs[1] analyzes Java byte-code to detect bugs against 424 known bug patterns at the time of writing this paper. Al-Ameen et al. identified 8 bug patterns that cannot be identified by FindBugs because some information is lost when source code is translated to byte-code [3]. They identified the following bug patterns cannot be detected on the byte-code representation of Java source code: zero or negative length arrays, divide by zero, integer overflow, out of bounds array, probable out of bounds, never executed for loop, unexpected loop behaviour. Primary evaluation from Al-Ameen et al. showed a reduction of false negatives from 50% to 15% using their bug detector compared to FindBugs. PMD[2] is tool that analyzes source code of multiple languages: PLSQL, Apache Velocity, XML, XSL. Both FindBugs and PMD allow users to configure and add custom bug patterns.

[1] http://findbugs.sourceforge.net/
[2] http://pmd.sourceforge.net/

Several papers discussed alternative bug tracking tools that look at artifacts other than source code alone. For example, Yu et al. proposed BugDetector as tool to detect bugs based on static analysis of Java source code and existing bug reports data [4]. Yu manually analyzed 200 bug reports and categorized them into 42 categories mapping each category to one or more bug patterns as follows: EqualsMayReturnNullBug, NullPointerBug, OverrideEqualsOrHashcodeBug, EqualsWithObjectBug, MaliciousCodeBug, SwitchClauseWithoutDefaultClauseBug, SwitchSubClauseWithoutBreakClauseBug, NewStringInstanceBug,NewIntegerInstanceBug,SelfAssignmentBug, BlockNotContainAnyClauseInIfElseClauseBug. These patterns are coded in Semantic Web Rule Language (SWRL) to develop an ontology. Then a program detects potential bugs by comparing the abstract syntax tree representation of a Java program against the SWRL coded bug patterns. In a case study, they showed BugDetector was able to detect more bugs compared to FindBugs (443 vs. 341) for 3 out of the 4 projects studied while taking more time (approx. 5x) in the process. Jiang et al. analyzed bug patterns by developers and found that personalizing the bug detection process based on past records of developers produced an improved classification of buggy vs. clean code [5].

Ocariza Jr. et al. analyzed reported bugs on JavaScript code and found that 79% of JavaScript bugs are related to DOM manipulation. Ocariza et al. developed AutoFLox is a bug detector tool for JavaScript to automatically detect potential null errors is JavaScript code that are triggered because expected DOM elements are missing on the HTML [**?**].

2) Bug Reporting: Pattern matching techniques have been used to classify bug reports. For example, Chaturvedi et al. used a machine learning technique to automatically infer the potential severity of a bug to help the bug triage process based on artifacts of a single kind, the existing bug reports [7]. They compute important terms for each bug severity level based on existing bugs which are then matched against a new reported bug to automatically infer it's severity. Text mining has also been used to automatically classify bug reports. Limsettho et al. used an unsupervised machined learning technique to classify bug reports into clusters based on the text similarity and also automatically tag each cluster with meaningful label extracted using NLP [8].

Nagwani et al. proposed a technique for automatically assigning expert developers to fix a bug [9]. Using this technique, each developer is mapped against a set of terms that is automatically extracted from previously assigned bugs. This mapping is used against the terms found on new bug report to rank the developers using similarity metrics.

In this cases, since the solutions are based on project specific historical data about bug reports, the technique is reusable across projects, but the specific pattern used on a project is developed and evolves with the project.

ReLink is a tool developed buy Wu et al. to find the missing links between bug reports and their corresponding code commit [10]. ReLink finds explicit links where bug numbers are found on commit logs as well as uses similarity metrics to infer the missing links. To compute similarity, it uses both the text and context. The context comprises of people and timing of the bugs and code commits with the heuristic that similarity in these properties provide additional information to the text. Bissyande performed an evaluation of ReLink with 12,000 bugs on 10 programs showed high precision but low recall for commits where explicit bug references are missing [12]. However, the results were 50% better compared to when only text match is used to infer the link between commit and bug reports.

3) Bug Fixing: Patterns have been used to generate patches for fixing code against known bug patterns. R2Fix auto-generates patches based on past bug patterns and using machine learning techniques [11]. When system generated crash reports are converted into bugs, the bugs include detailed information about the call stack at runtime that triggered the buggy code path. To use R2Fix, past bug reports are manually categorized and patterns of code execution are extracted for each category. R2Fix automatically generated 57 patches with a precision of 71.3% by analyzing bug reports for three large open source projects, with 5 new patches for bugs that were yet to be fixed, and 4 of the 5 auto-generated patches were accepted and merged into the source code.

4) Bug Analysis: Previous bug reports have been analyzed by connecting the different artifacts together so that the relationship among bug reports, code commits and source code can be categorized into bug patterns. Ahsan et al. developed a database comprising of bug reports, commit logs and source code that can be used for analyzing bug and code change patterns [13]. They manually analyzed 3716 code commits and classified the commits as follows: bug introducing (18.2%), fixing (15.8%), bug fix-introducing (37.8%) and clean (28.1%). As seen from their analysis, a total of (18.2+37.8) = 56% commits were introducing new bugs. Ocariza et al. categorized bugs commonly observed in JavaScript code under the following patterns: erroneous input validation, error in writing string literal, neglecting differences in browser behavior, forgetting null/undefined check, and error in syntax [6].

Steff et al. analyzed 1060 historical bug reports and 579 code commits and found that the history of a commit described by its files was correlated to defects [14]. Steff et al. visualized the commits as a graph, where each node represents a commit and each edge represents a file that is changed by a subsequent commit. The graph showed that subsequent bug fixes on the same file were separated by at most two commits. This implies a bug pattern that defect-prone files are likely to create new defect every two commits. Zhang et al. classified code changes into 4 patterns to understand the impact of each pattern on potentially introducing bugs [15]. Zhang identified the code change patterns as follows: concurrent (multiple developers working on the same file at the same time), parallel (multiple files that change together by single developers), extended (files that change over a longer period of time), and interrupted (files that are edited with long interruptions). Zhang categorized changes on 2,140 files and 98 bugs related to these files and found that concurrent and parallel changes introduced 2.46 and 1.67 times more bugs respectively. They also identified that combination of these change patterns introduced more bugs than the individual change patterns.

Osman et al. analyzed the contents of the bug-fixing code changes and found that most bug fixing changes involve a small change in code [**?**]. They found out of 94,534 bug

fixing changes, 73% of the fixes contained less than 4 lines of code change each. They identified 4 recurrent patterns of bugs in those bug-fixing changes: null checks, missing method invocation, wrong names and undue invocations.

B. RQ2 - How these bug patterns were established?

C. RQ3 - What is known about the impact of these bug patterns?

V. Discussion

A. Threats to Validity

VI. Conclusion

VII. Conclusion

References

[1] Allen, Eric, Bug patterns in Java, 2002

[2] Sai Zhang; Jianjun Zhao, On Identifying Bug Patterns in Aspect-Oriented Programs, Computer Software and Applications Conference, 2007. COMPSAC 2007. 31st Annual International , vol.1, no., pp.431,438, 24-27 July 2007

[3] Al-Ameen, M.N.; Hasan, M.M.; Hamid, A., Making findbugs more powerful, Software Engineering and Service Science (ICSESS), 2011 IEEE 2nd International Conference on , vol., no., pp.705,708, 15-17 July 2011

[4] Lian Yu; Jun Zhou; Yue Yi; Ping Li; Qianxiang Wang, Ontology Model-Based Static Analysis on Java Programs, Computer Software and Applications, 2008. COMPSAC '08. 32nd Annual IEEE International , vol., no., pp.92,99, July 28 2008-Aug. 1 2008

[5] Tian Jiang; Lin Tan; Sunghun Kim, Personalized defect prediction, Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on , vol., no., pp.279,289, 11-15 Nov. 2013 doi: 10.1109/ASE.2013.6693087

[6] Ocariza, F.; Bajaj, K.; Pattabiraman, K.; Mesbah, A., An Empirical Study of Client-Side JavaScript Bugs, Empirical Software Engineering and Measurement, 2013 ACM / IEEE International Symposium on , vol., no., pp.55,64, 10-11 Oct. 2013

[7] Chaturvedi, K.K.; Singh, V.B., Determining Bug severity using machine learning techniques, Software Engineering (CONSEG), 2012 CSI Sixth International Conference on , vol., no., pp.1,6, 5-7 Sept. 2012

[8] Limsettho, N.; Hata, H.; Monden, A.; Matsumoto, K., Automatic Unsupervised Bug Report Categorization, Empirical Software Engineering in Practice (IWESEP), 2014 6th International Workshop on , vol., no., pp.7,12, 12-13 Nov. 2014

[9] Nagwani, N.K.; Verma, S., Predicting expert developers for newly reported bugs using frequent terms similarities of bug attributes, ICT and Knowledge Engineering, 2011 9th International Conference on , vol., no., pp.113,117, 12-13 Jan. 2012

[10] Wu, Rongxin and Zhang, Hongyu and Kim, Sunghun and Cheung, Shing-Chi, ReLink: Recovering Links Between Bugs and Changes, 2011 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, pp.15,25

[11] Chen Liu; Jinqiu Yang; Lin Tan; Hafiz, M., R2Fix: Automatically Generating Bug Fixes from Bug Reports, Software Testing, Verification and Validation (ICST), 2013 IEEE Sixth International Conference on , vol., no., pp.282,291, 18-22 March 2013

[12] Bissyande, T.F.; Thung, F.; Shaowei Wang; Lo, D.; Lingxiao Jiang; Reveillere, L., Empirical Evaluation of Bug Linking, Software Maintenance and Reengineering (CSMR), 2013 17th European Conference on , vol., no., pp.89,98, 5-8 March 2013

[13] Ahsan, S.N.; Ferzund, J.; Wotawa, F., A Database for the Analysis of Program Change Patterns, Networked Computing and Advanced Information Management, 2008. NCM '08. Fourth International Conference on , vol.2, no., pp.32,39, 2-4 Sept. 2008

[14] Steff, M.; Russo, B., Co-evolution of logical couplings and commits for defect estimation, Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on , vol., no., pp.213,216, 2-3 June 2012

[15] Feng Zhang; Khomh, F.; Ying Zou; Hassan, A.E., An Empirical Study of the Effect of File Editing Patterns on Software Quality, Reverse Engineering (WCRE), 2012 19th Working Conference on , vol., no., pp.456,465, 15-18 Oct. 2012

[16] Howden, W.E., Software test selection patterns and elusive bugs, Computer Software and Applications Conference, 2005. COMPSAC 2005. 29th Annual International , vol.2, no., pp.25,32 Vol. 2, 26-28 July 2005

[17] Osman, H.; Lungu, M.; Nierstrasz, O., Mining frequent bug-fix code changes, Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week - IEEE Conference on , vol., no., pp.343,347, 3-6 Feb. 2014