Improving the Effectiveness of Peer Code Review in Identifying Security Defects

Rajshakhar Paul
r.paul@wayne.edu
Wayne State University
Detroit, Michigan, USA

ABSTRACT

Prior studies found peer code review useful in identifying security defects. That is why most of the commercial and open-source software (OSS) projects embraced peer code review and mandated the use of it in the software development life cycle. However, despite conducting mandatory peer code review practices, many security-critical OSS projects such as Chromium, Mozilla, and Qt are reporting a high number of post-release vulnerabilities to the Common Vulnerabilities and Exposures (CVE) database. Practitioners may wonder if there is any missing piece in the puzzle that leads code reviews to miss those security defects. Therefore, the primary objective of this dissertation study is *to improve the effectiveness of peer code review in identifying security defects.*

To meet this goal, I plan to empirically investigate: (i) why security defects escape code reviews, (ii) what are the challenges developers face to conduct effective security code reviews, (iii) how to build effective security code review strategy, and (iv) how to make effective utilization of security experts during code reviews.

CCS CONCEPTS

• **Security and privacy → Software security engineering**; • **Software and its engineering → Empirical software validation**; **Search-based software engineering**.

KEYWORDS

security, code review, vulnerability, software development

1 INTRODUCTION

Security defects belong to a special class of software defects that can be exploited by the attackers to destroy, expose, or alter sensitive information. According to Gary McGraw, adopting peer code review in the software development life-cycle can be effective in eliminating such defects [20]. Prior empirical studies [8, 13, 25] find peer code review useful in identifying security defects which support the recommendation of Gary McGraw. Since the longer it takes to detect and fix a security vulnerability, the more that vulnerability will cost [20], code reviews, which occur almost immediately after the introduction of a vulnerability, can effectively minimize the economic impact of a security vulnerability as well. That is why, most of the popular open-source (OSS) and proprietary software projects mandate peer code review in their software development process [28]. That means each code change is inspected and verified by at least one reviewer. Prior study also finds that developers of those projects spend 10-15% of their working hours conducting peer code review [7].

However, despite investing lots of resources and time on peer code review, a large number of post-release vulnerabilities still have been identified and reported to the Common Vulnerabilities and Exposures (CVE) database[1]. Since security defects are different from other bugs [10], practitioners may wonder if there are any missing pieces that lead peer code reviews to miss those security defects. To fill that gap, the primary objective of this doctoral dissertation is *to improve the effectiveness of peer code review in identifying security defects.* To achieve this goal, I plan to conduct the following three studies.

#1: Identifying why security defects go unnoticed during code reviews
Motivation: The practitioners may want to investigate if there is any flaw or weakness in the contemporary code review process that lead code reviews to miss security defects. For example, practitioners may wonder: (i) whether the code change is too difficult to comprehend, (ii) whether the reviewers have adequate expertise on the code they are reviewing, or (iii) whether the reviewers spending adequate time on reviewing the code.
Objective: *To identify the factors that can differentiate between code reviews that successfully identified security defects and the code reviews that missed such security defects.*

#2: Identifying the challenges in security code review and building effective strategies
Motivation: Understanding the underlying challenges developers face to conduct effective security code review would help practitioners in two ways– (i) they could pinpoint the weakness of code review in identifying security defects, and (ii) they could build an effective strategy/guideline for security code reviews.
Objective: Two objectives– *(i) better understand the challenges developers face to conduct effective security code reviews* and *(ii) identify developers' perspective on security code review effective practices.*

#3: Utilizing security code reviewers effectively
Motivation: Involving security experts to review every code changes would help the project management to ensure better security. However, such security experts are scarce [19] and involving them to review every code change is not feasible. Thus, utilizing security experts to review the appropriate security-critical code changes is crucial.
Objective: *To build a machine learning based model to automatically identify security-critical code change and recommend expert reviewers based on the potential defect type.*

[1] https://cve.mitre.org/

2 BACKGROUND

2.1 Peer Code Review

Peer code review is a software quality assurance practice where developers send their code to their peers to verify that the code achieves adequate quality to get merged into the project main codebase. Compared to traditional formal code inspection approaches, modern peer code review is more informal, lightweight, and tool-based. That is why developers practice peer code review on a regular basis [4]. Studies find peer code review not only useful in identifying missing defects [5, 18], but also effective in spotting security defects [8, 13, 19, 24]. That is why most of the popular OSS and proprietary projects embraced peer code review practices in their software development life-cycle [28].

2.2 Security Vulnerabilities

A software security vulnerability is a weakness or flaw in the software component which can be exploited by an attacker to execute unauthorized actions such as exposing, destroying, or altering confidential data. Writing buggy code, violating security protocol while coding, inappropriate system design, or poor implementation quality are the preliminary reasons behind introducing vulnerability. There can be hundreds of types of security defects occurred in the code. The security communities follow Common Weakness Enumeration (CWE) [1] as a formal categorization of those security defects.

2.3 Related Works

Researchers have justified the positive impact of peer code reviews in identifying security defects [8, 12, 29]. However, still a significant number of post-release security defects are being reported in the CVE database following an increasing trend. Prior studies of Edmundson *et* al. [13] and Beller *et* al. [5] raised questions about the effectiveness of peer code review in identifying security defects. However, they do not conduct any investigation to identify why peer code reviews fail to ensure that effectiveness. Over the years, researchers have introduced different code review attributes that can be useful to identify security defects [3, 21, 22]. Meenely and Williams find that engagement of too many developers on a single source file has higher probability to make that file vulnerable [23]. Munaiah *et* al. apply natural language processing techniques on code review comments of Chromium project and find that code reviews that have discussions with higher sentiment, lower inquisitiveness, and lower syntactical complexity have more potential to miss security vulnerabilities. However, none of these studies investigate the underlying reasons behind why developers miss security defects during code reviews.

The study of Woon and Kankanhalli suggests that developers' lack of intention to practice secure software development is the main reason behind introducing vulnerabilities [30]. Xie et al. find a disconnection between developers' security knowledge and their mindset towards secure programming [31]. However, they conclude their results based on interviewing 15 developers which is relatively a small sample size.

To identify security defects, we need to conduct effective security code review. Howard provides a guideline of conducting security code review based on his experience about what he follows in professional life [16]. He maintains a checklist of ten tactics while conducting security code reviews. Ionescu suggests checking for OWASP Top 10[2] and MITRE Top 25[3] while conducting security code review [17]. Chapman *et* al. also provide a checklist of nine security defects in the OWASP Code Review Guide that need to be investigated while conducting security code reviews [11]. Although, those guidelines provide several ideas about conducting security code review, none of those are empirically verified.

In this dissertation proposal, I empirically investigate the underlying reasons behind why security defects escape code reviews. In addition, I plan to develop an empirically verified strategy to conduct effective security code reviews.

3 PROPOSED RESEARCH

In the following subsections, I describe the research approaches of my three studies.

3.1 Study1: Identifying Why Security Defects Go Unnoticed during Code Reviews

To investigate the underlying reasons behind why security defects are escaping code reviews, I construct two research questions.

Identifying security defects depends on a reviewer's knowledge and understanding about the project architecture and potential security implications. That is why a reviewer may find some security defects challenging to identify during code reviews. Identification of such security defects that are harder to identify during code reviews could help the project management in two ways– (i) the management could educate the developers on those specific security defect categories, and (ii) the management could understand which testing tools they need to leverage to protect against those defects. That is why, I construct my first research question as:

RQ1: *Which categories of security defects have higher probability of being escaped during code reviews?*

In addition, identifying if any attributes apart from the defect category play significant role in differentiating code reviews that identify security defects from code reviews where security defects escape could help the practitioners to find out the missing piece of the puzzle. On this goal, my second research question is:

RQ2: *Which attributes plays significant role in identifying security defects during code review?*

3.1.1 Research Methodology: To answer the two research questions, we conduct a case study of Chromium OS project since Chromium is a crucial OSS project with significant security implications and it has been subject to prior studies related to security investigation [10, 21, 24]. The Chromium project also provides a publicly accessible REST API which allow us to mine all publicly available code reviews.

I collect two types of code review data from Chromium OS project– (i) code review that successfully identified security defects, and (ii) code review where security defects escaped. To build the dataset of code review that identified security defects, I adopt a keyword-based mining approach as Bosu *et* al. [8]. To build the

[2] https://owasp.org/www-project-top-ten/
[3] https://cwe.mitre.org/top25/archive/2020/2020_cwe_top25.html

dataset of code reviews where security defects escaped, I utilize the custom search feature of Monorail-based bug tracking system of Chromium OS project[4] and adopt a modified SZZ-based algorithm [6]. In total, I collect 516 code reviews that identified security defects and 374 code reviews where security defects escaped. I classify each security defect under a Common Weakness Enumeration (CWE) class [1] which is a standard classification scheme of software weakness types.

3.1.2 Research Progress: For each code review, I collect 18 attributes as described in the work of Paul *et* al. [27] and build a logistic regression model followed by the approach of Harrell Jr [15]. Our model achieves an AUC score[14] of 0.914 and an adjusted R^2 value of 0.6375 which is considered as a good fit [26]. In addition, Our attributes explain 47.21% variances of the model which can be considered as a significant improvement over the NULL model.

I find that some security defects are significantly ($\chi^2 = 491.69$, $p < 0.001$) less likely to be identified during code reviews. Security defects related to exposure of resource to wrong sphere (CWE 668), improper access control (CWE 284), improper input validation (CWE 20), and incorrect type conversion/cast (CWE 704) have very high probability of being escaped during code reviews. On the other hand, security defects related to use of potentially unsafe methods (CWE 676), resource leak (CWE 404), improper synchronization of multi-threaded applications (CWE 662), potential race condition (CWE 362), and incorrect calculation of array/buffer/integer size (CWE 682) have higher probability of being identified during code reviews.

The regression model also finds that if the code review has higher review time, higher number of mutual reviews between the code author and reviewer, if the reviewer has prior review experience with the source code file that is under review, or if the code change is labeled as bug-fix of a prior defect, the code review has higher probability that it will identify the security defects. On the other hand, if the code change touches multiple directories, if the source file has higher number of prior commits, or if the number of review comments gets higher during the review cycle, the security defect is more likely to be escaped. My paper on these findings has been accepted to the 43^{rd} International Conference on Software Engineering (ICSE'21) [27].

3.1.3 Contributions: The main contributions of this study are:

- An empirically built and validated dataset of code reviews that either missed or identified security defects.
- An empirical investigation of security defects that are identified during code review Vs. security defects that escape.
- A logistic regression model that can identify attributes that play significant role in identifying security defects during code reviews.
- A publicly accessible code and dataset.

[4] https://crbug.com

3.2 Study2: Identifying the Challenges in Security Code Review and Building Effective Strategies

Since my first study identifies a set of security defects that have higher probability of being escaped during code reviews and a set of attributes that play significant role behind that escape, in my second study, I aim to investigate what challenges developers face to conduct effective security code reviews. To identify such challenges, I plan to investigate developers' perspective on this matter. In addition, based on the developers' feedback, I plan to develop a guideline that help conducting effective security code reviews. So, the overall study is divided into two parts.

(Part 1) Identifying challenges in security code review: To understand the challenges that developers face to conduct effective security code review, I construct two research questions.

Understanding developers' perspective regarding why they miss security defects during code review would help the project management to identify necessary steps that need to be undertaken. That is why, I construct my first research question as:

RQ1.1: *Why do developers miss security defects during code reviews?*

Understanding which security defects developers find difficult to spot during code reviews would help the project management more to understand what additional testing tools need to be leveraged and how they can provide training to the developers to spot those security defects. It will also help to cross-check with the findings of my first study. For that, my second research questions is:

RQ1.2: *Which categories of security defects are more difficult to identify during code reviews?*

(Part 2) Building effective security code review strategy: There are different approaches and guidelines about how to conduct security code review [11, 16, 17]. However, there is no empirically verified evidence that can answer what approaches are effective in identifying security defects during code reviews and how to build expertise in conducting security code review. To investigate that, I construct another three research questions.

It can require multiple sources of knowledge to become a security expert. Identifying those sources would help developers to build expertise in conducting effective security code review. For that, I construct my first research question as:

RQ2.1: *What helps developers to build expertise in conducting security code review?*

A project management may want to investigate if their followed approach of security code review has any weakness or gap. Identifying what approaches do developers find most effective while conducting security code review would enlighten the management to fill that gap. For that, my second research question is:

RQ2.2: *Which approaches developers find most effective while conducting security code review?*

Finally, understanding which initiatives the developers think that the project management should take to improve security code reviewing would help the management to better understand the necessary steps the project should adopt. For that why my third and final research questions is:

RQ2.3: *Which initiatives a project can take to improve security code reviews?*

3.2.1 Methodology: Since I am planning to investigate developers' perspective to identify the existing challenges and potential approaches for effective strategy, I choose questionnaire survey as my research instrument. I design a survey with 25 questions where each question either belongs to respondents' demographic or any of the five research questions that I construct in the previous subsection. I sent the survey to 5,697 developers of 37 different OSS projects and got reply from 315.

Since apart from *RQ1.2*, all the questions related to the RQs were open-ended, I follow qualitative analysis approach. I with one of my peers manually classify each response to a particular category following the similar approach of Bosu *et* al. [9].

3.2.2 Research Progress: The data collection and analysis is completed. Two manuscripts of this study are currently under review.

3.2.3 Expected Contributions: The expected contributions of this study are:

- A better understanding about the challenges developers face while conducting security code review.
- A better understanding about the security defects that developers find difficult to spot during code review.
- A better understanding about the knowledge source that help developers building expertise in security.
- A better understanding the approaches modern developers find effective while conducting security code review.
- A list of recommendations a project management can adopt to produce better security code reviewers.

3.3 Study3: Utilizing Security Code Reviewers Effectively

To utilize security code reviewers effectively, we will need to identify which code reviews raise security concern. On this goal, I plan to build a machine learning based model to automatically identify a code review as soon as that triggers a security concern. I will be leveraging both the source code and code review discussion. In a code review, if any security concern appears, developers including code author and reviewers may engage in a discussion to resolve that concern (terming as 'security discussion' hereinafter). Developers often discuss the origin of the problem and potential solution to that. However, some security concerns may go unnoticed. Or even if noticed, the lack of involvement of security experts may mislead to a fragile solution which can lead further security vulnerability. An additional scrutiny from a security expert before merging the code can reduce the probability of introducing further security vulnerability.

3.3.1 Methodology: I will be using code review data of Chromium OS project. Chromium use *Gerrit*[5] for tool-based code review. I will be collecting code review comments from *Gerrit*. For each code review comment, I will also collect the corresponding source code file. We can extract the line/s of source code that trigger the discussion. After collecting data, I with one of my peers will label the code review comment whether the comment is related to security concern or not.

[5] https://chromium-review.googlesource.com/

Once the data labelling is done, I will be developing a machine learning based model to predict whether a code review comment is security-related or not. Once the model will find a security-related comment, it will label the entire discussion as a security discussion. The input to the model will be code review comments written in natural English and the code context that triggered the discussion. I will be leveraging Natural Language Processing (NLP) techniques for both comments and code. For comments, I will perform necessary pre-processing steps before building the model. For code context, I will be extracting the entire function/method from the source code where the line/s that trigger discussion belong. Since, taking only the line/s that trigger discussion will be too less to extract context, and taking the entire code can be too much to extract context or can be out of context, I consider that the appropriate code context for a certain line can be extracted from the function body where that line belongs. Once I extract the function body, I will utilize the Code2Vec model [2] to extract the code context. Then I will build the model using pre-processed text and code context. The model will be able to identify security discussion in real-time.

In addition, I will be developing a reviewer recommendation system which will automatically assign the best-fitted expert reviewer to scrutinize the code change that triggers security discussion.

3.3.2 Research Progress: The data collection and labelling processes are ongoing. The detailed approach to build the system is yet to be finalized.

3.3.3 Expected Contributions: The expected contributions are:

- An empirically developed and validated dataset of code review comments that either belong to security discussion or do not.
- A machine learning based model to automatically identify code reviews that trigger security concern.
- A reviewer recommendation system that will automatically recommend best possible security expert to scrutinize the code change.

4 CONCLUSION

In my doctoral research, I focus on improving the effectiveness of peer code review in identifying security defects. To reach my goal, I construct three novel study approaches. I believe, successful completion of those studies will help the practitioners to understand– (i) why contemporary peer code reviews miss security defects, (ii) what are the challenges developers face to conduct effective security code review, (iii) what should project managements adopt for effective security code review, (iv) how to build an effective security code review strategy, and (v) how to utilize the security experts efficiently during code review.

REFERENCES

[1] [n.d.]. Common Weakness Enumeration, A Community-Developed List of Software & Hardware Weakness Types. https://cwe.mitre.org/about/index.html. Accessed: 2021-07-08.

[2] Uri Alon, Meital Zilberstein, Omer Levy, and Eran Yahav. 2019. code2vec: Learning distributed representations of code. *Proceedings of the ACM on Programming Languages* 3, POPL (2019), 1–29. https://doi.org/10.1145/3290353

[3] Henrique Alves, Baldoino Fonseca, and Nuno Antunes. 2016. Software metrics and security vulnerabilities: dataset and exploratory study. In *2016 12th European Dependable Computing Conference (EDCC)*. IEEE, 37–44. https://doi.org/10.1109/EDCC.2016.34

[4] Alberto Bacchelli and Christian Bird. 2013. Expectations, outcomes, and challenges of modern code review. In *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 712–721. https://doi.org/10.1109/ICSE.2013.6606617

[5] Moritz Beller, Alberto Bacchelli, Andy Zaidman, and Elmar Juergens. 2014. Modern code reviews in open-source projects: Which problems do they fix?. In *Proceedings of the 11th working conference on mining software repositories*. 202–211. https://doi.org/10.1145/2597073.2597082

[6] Markus Borg, Oscar Svensson, Kristian Berg, and Daniel Hansson. 2019. SZZ unleashed: an open implementation of the SZZ algorithm-featuring example usage in a study of just-in-time bug prediction for the jenkins project. In *Proceedings of the 3rd ACM SIGSOFT International Workshop on Machine Learning Techniques for Software Quality Evaluation*. 7–12. https://doi.org/10.1145/3340482.3342742

[7] Amiangshu Bosu, Jeffrey C Carver, Christian Bird, Jonathan Orbeck, and Christopher Chockley. 2016. Process aspects and social dynamics of contemporary code review: Insights from open source development and industrial practice at microsoft. *IEEE Transactions on Software Engineering* 43, 1 (2016), 56–75. https://doi.org/10.1109/TSE.2016.2576451

[8] Amiangshu Bosu, Jeffrey C Carver, Munawar Hafiz, Patrick Hilley, and Derek Janni. 2014. Identifying the characteristics of vulnerable code changes: An empirical study. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. 257–268. https://doi.org/10.1145/2635868.2635880

[9] Amiangshu Bosu, Anindya Iqbal, Rifat Shahriyar, and Partha Chakraborty. 2019. Understanding the motivations, challenges and needs of blockchain software developers: A survey. *Empirical Software Engineering* 24, 4 (2019), 2636–2673. https://doi.org/10.1007/s10664-019-09708-7

[10] Felivel Camilo, Andrew Meneely, and Meiyappan Nagappan. 2015. Do bugs foreshadow vulnerabilities? a study of the chromium project. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 269–279. https://doi.org/10.1109/MSR.2015.32

[11] Jenelle Chapman, Andrew Van der Stock, Eoin Keary, Paolo Perego, David Lowry, David Rook, Dinis Cruz, and Jeff Williams. 2017. OWASP Code Review Guide v2. *OWASP Foundation, November* (2017).

[12] Marco di Biase, Magiel Bruntink, and Alberto Bacchelli. 2016. A security perspective on code review: The case of chromium. In *2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 21–30. https://doi.org/10.1109/SCAM.2016.30

[13] Anne Edmundson, Brian Holtkamp, Emanuel Rivera, Matthew Finifter, Adrian Mettler, and David Wagner. 2013. An empirical study on the effectiveness of security code review. In *International Symposium on Engineering Secure Software and Systems*. Springer, 197–212. https://doi.org/10.1007/978-3-642-36563-8_14

[14] James A Hanley and Barbara J McNeil. 1982. The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology* 143, 1 (1982), 29–36. https://doi.org/10.1148/radiology.143.1.7063747

[15] Frank E Harrell Jr. 2015. *Regression modeling strategies: with applications to linear models, logistic and ordinal regression, and survival analysis*. Springer. https://doi.org/10.1007/978-3-319-19425-7

[16] Michael A Howard. 2006. A process for performing security code reviews. *IEEE Security & privacy* 4, 4 (2006), 74–79. https://doi.org/10.1109/MSP.2006.84

[17] Paul Ionescu. 2019. Security Code Review 101. https://medium.com/@paul_io/security-code-review-101-a3c593dc6854. Accessed: 2021-07-08.

[18] Mika V Mäntylä and Casper Lassenius. 2008. What types of defects are really discovered in code reviews? *IEEE Transactions on Software Engineering* 35, 3 (2008), 430–448. https://doi.org/10.1109/TSE.2008.71

[19] Gary McGraw. 2006. Software security. *Building security in* (2006).

[20] Gary McGraw, Brian Chess, and Sammy Migues. 2009. Building security in maturity model. *Fortify & Cigital* (2009).

[21] Andrew Meneely, Harshavardhan Srinivasan, Ayemi Musa, Alberto Rodriguez Tejeda, Matthew Mokary, and Brian Spates. 2013. When a patch goes bad: Exploring the properties of vulnerability-contributing commits. In *2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. IEEE, 65–74. https://doi.org/10.1109/ESEM.2013.19

[22] Andrew Meneely, Alberto C Rodriguez Tejeda, Brian Spates, Shannon Trudeau, Danielle Neuberger, Katherine Whitlock, Christopher Ketant, and Kayla Davis. 2014. An empirical investigation of socio-technical code review metrics and security vulnerabilities. In *Proceedings of the 6th International Workshop on Social Software Engineering*. 37–44. https://doi.org/10.1145/2661685.2661687

[23] Andrew Meneely and Laurie Williams. 2009. Secure open source collaboration: an empirical study of linus' law. In *Proceedings of the 16th ACM conference on Computer and communications security*. 453–462. https://doi.org/10.1145/1653662.1653717

[24] Nuthan Munaiah and Andrew Meneely. 2016. Vulnerability severity scoring and bounties: Why the disconnect?. In *Proceedings of the 2nd International Workshop on Software Analytics*. 8–14. https://doi.org/10.1145/2989238.2989239

[25] Nuthan Munaiah, Benjamin S Meyers, Cecilia O Alm, Andrew Meneely, Pradeep K Murukannaiah, Emily Prud'hommeaux, Josephine Wolff, and Yang Yu. 2017. Natural language insights from code reviews that missed a vulnerability. In *International Symposium on Engineering Secure Software and Systems*. Springer, 70–86. https://doi.org/10.1007/978-3-319-62105-0_5

[26] Nico JD Nagelkerke et al. 1991. A note on a general definition of the coefficient of determination. *Biometrika* 78, 3 (1991), 691–692.

[27] Rajshakhar Paul, Asif Kamal Turzo, and Amiangshu Bosu. 2021. Why security defects go unnoticed during code reviews? A case-control study of the chromium os project. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 1373–1385. https://doi.org/10.1109/ICSE43902.2021.00124

[28] Peter C Rigby and Christian Bird. 2013. Convergent contemporary software peer review practices. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. 202–212. https://doi.org/10.1145/2491411.2491444

[29] Christopher Thompson and David Wagner. 2017. A large-scale study of modern code review and security in open source projects. In *Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering*. 83–92. https://doi.org/10.1145/3127005.3127014

[30] Irene MY Woon and Atreyi Kankanhalli. 2007. Investigation of IS professionals' intention to practise secure development of applications. *International Journal of Human-Computer Studies* 65, 1 (2007), 29–41. https://doi.org/10.1016/j.ijhcs.2006.08.003

[31] Jing Xie, Heather Richter Lipford, and Bill Chu. 2011. Why do programmers make security errors?. In *2011 IEEE symposium on visual languages and human-centric computing (VL/HCC)*. IEEE, 161–164. https://doi.org/10.1109/VLHCC.2011.6070393