Software Ticket Quality

Andrei-Mihai Nicolae

School of Computing Science
Sir Alwyn Williams Building
University of Glasgow
G12 8QQ

Masters project proposal

December 18th 2017

Contents

1 Introduction **2**

1.1 A subsection 2

2 Statement of Problem **2**

3 Background Survey **2**

3.1 Introduction 2

3.2 What was covered and why 2

3.3 How papers were found; what terms were searched for + snowballing . . . 2

3.4 Data Quality Metrics 2

3.5 Issue Quality 2

3.6 Measuring Cost and Waste in Software Projects 4

3.7 Sentiment Analysis 5

3.8 Conclusion 5

4 Proposed Approach **5**

5 Work Plan **6**

1 Introduction

briefly explain the context of the project problem

1.1 A subsection

Please note your proposal need not follow the included section headings; this is only a suggested structure. Also add subsections etc as required

2 Statement of Problem

Clearly state the problem to be addressed in your forthcoming project. Explain why it would be worthwhile to solve this problem.

3 Background Survey

3.1 Introduction

3.2 What was covered and why

3.3 How papers were found; what terms were searched for + snowballing

3.4 Data Quality Metrics

3.5 Issue Quality

Secret Life of Bugs[2]:

- What were the goals? The paper tries to understand and analyze common bug fixing coordination activities. Another goal of the paper was to analyze the reliability of repositories in terms of software projects coordination and propose different directions on how to implement proper tools.
- What was the method? They executed a field study which was split into two parts:
  - firstly, they did an exploratory case study of bug repos histories;

  - secondly, they conducted a survey with professionals (i.e. testers, developers).

  All data and interviews were conducted using Microsoft bug repositories and employees.

- What did they learn? They learned that there are multiple factors which influence the coordination activities that revolve around bug fixing, such as organizational, social and technical knowledge, thus one cannot infer any conclusions only by automatic analysis of the bug repositories. Also, through surverying the professionals, they reached the conclusion that there are 8 main goals which can be used for better tools and practices:
  - probing for ownership;
  - summit;
  - probing for expertise;
  - code review;
  - triaging;
  - rapid-fire emailing;
  - infrequent, direct email;
  - shotgun emails.
- Relevance to our work: this paper is one of the key papers for our research paper.

What makes a good bug report[3]:

- **What were the goals?** The main goal of the paper was to investigate the **quality of bug reports** from a developer's point of view, based on the typical information found in such a report (e.g. stack traces, screenshots).
- **What was the method?** The authors conducted a massive survey with over 450 respondents. The survey was online and it targeted developers from Mozilla, Apache and Eclipse.
- **What did they learn?** The main conclusion of this research paper was that well written bug reports will more likely get the attention of the developers. Thus, including steps to reproduce the bugs or stack traces proved to increase the quality of the bug report. Also, an important achievement reached by the authors was the development of a prototype tool called Cuezilla that could estimate, with an accuracy rate of 3148%, the quality of a bug report.
- **Relevance to our work**: it is relevant to our research in that it provides valuable insight into what makes for a good bug report based on actual professionals' opinions.

Who should fix this bug?[1]:

- **What were the goals?** The authors aimed to create a tool that could automatically assign the bug report to a specific developer based on his/her suitability for that specific task.
- **What was the method?** They applied a supervised machine learning algorithm on the repositories to learn which developers were best suited for specific tasks, thus when a new bug report would come in, a small set of people would be selected. In order to train the algorithm, they looked at Bugzilla repositories and selected the free text form of tickets, trying to label similar ones based on textual similarities. Once the tickets were labeled and grouped for specific developers, the algorithm would then be able to present the triager the set of developers suitable to fix the bug.
- **What did they learn?** The most important lesson learned was that collecting data from bug reports and CVS logs was quite challenging. One of the major reasons why they found this aspect hard was that not all CVS comments referenced the specific bug report id.
- **Relevance to our work** The paper taught us that bug triaging is hard and that there is almost no automated tool that can choose the perfect developers to work on the task. Moreover, the method applied by the authors could prove useful as a learning aid when working with the open source repositories chosen as data sets.

3.6 Measuring Cost and Waste in Software Projects

Waste Identification[4]:

- **What were the goals?** The paper had two main goals:
  - a means to identify communication waste in agile software projects environments;
  - types of communication waste in agile projects.
- **What was the method?** The authors collaborated with a medium-sized American software company and conducted a series of observations, informal discussions, documents provided by the organization, as well as semi-structured interviews. Moreover, the data collection for waste identification was split into 2 parts:
  - **pre-development**: occured before the actual implementation begun (e.g. backlog creation);
  - **development**: happened throughout the implementation process (e.g. throughout sprints, retrospectives, sprint reviews, communication media).
- **What did they learn?** They realized the communication waste can be divided into 5 main categories:
  - lack of involvement;

  - lack of shared understanding;
  - outdated information;
  - restricted access to information;
  - scattered information.

  Also, they learned that their way of identifying these types of waste was quite efficient and they even recommend it to companies if they'd like to conduct such processes internally.

- **Relevance to our work**: the waste identification process can be applied to our work so that we can identify possible causes to poor quality tickets.

Software Development Waste[5]:

- **What were the goals?** The main goal of the paper was to identify main types of waste in software development projects.
- **What was the method?** They conducted a participant-observation study over a long period of time at Pivotal, a consultancy software development company. They also interviewed multiple engineers and balanced theoretical sampling with analysis to achieve the conclusions.
- **What did they learn?** They found out there are nine main types of waste in software projects:
  - building the wrong feature or product;
  - mismanaging backlog;
  - extraneous cognitive load;
  - rework;
  - ineffective communication;
  - waiting/multitasking;
  - solutions too complex;
  - psychological distress.
  - **Relevance to our work**: this paper complements the previous one on waste identification[4].

3.7 Sentiment Analysis

3.8 Conclusion

4 Proposed Approach

state how you propose to solve the software development problem. Show that your proposed approach is feasible, but identify any risks.

5 Work Plan

show how you plan to organize your work, identifying intermediate deliverables and dates.

References

[1] John Anvik, Lyndon Hiew, and Gail C Murphy. Who should fix this bug? pages 361–370, 2006.

[2] Jorge Aranda and Gina Venolia. The secret life of bugs: Going past the errors and omissions in software repositories. pages 298–308, 2009.

[3] Nicolas Bettenburg, Sascha Just, Adrian Schröter, Cathrin Weiss, Rahul Premraj, and Thomas Zimmermann. What makes a good bug report? pages 308–318, 2008.

[4] Mikko Korkala and Frank Maurer. Waste identification as the means for improving communication in globally distributed agile software development. *The Journal of Systems and Software*, 95:122–140, 2014.

[5] Todd Sedano, Paul Ralph, and Cécile Péraire. Software development waste. pages 130–140, 2017.