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

- Background briefly explain the context of the project problem
- Problem statement and justification. Clearly state the problem to be addressed in your forthcoming project. Explain why it would be worthwhile to solve this problem.
- Research questions and contributions
- Structure of the rest of the document

2 Background Survey

2.1 Study Design

In order to conduct a thorough background survey, several topics needed to be explored, among which the most important were:

- data quality and metrics;
- issue/ticket quality;
- waste in software projects;
- sentiment analysis.

Firstly, we will deal with a large volume of tickets and information extracted from software projects and the tools that revolve around them, such as issue trackers and version control systems. Thus, *data quality* is an important topic that needed to be explored in order to infer appropriate conclusions.

We analyzed data quality mainly in software organizations and looked at what metrics are most useful when trying to perform measurements. We also tried to find different methodologies that either measure data quality or calculate the impact of poor data quality on the software project.

Secondly, probably the topic of most interest to our research is *issue quality*. This field is mainly concerned with software tickets and their characteristics: how important are issue trackers to a software project, what makes for a good bug report, what impact does bad triaging have etc. As we are interested in finding out what features are most important in improving the quality of a ticket, the issue quality literature I have reviewed proved to be beneficial and provided valuable insight into how other researchers tried to approach this problem.

The third topic was waste in software projects and how it impacts them. We needed to look into various communication and tool-related waste in software projects because having a project with a considerable amount of waste might affect our results and compromise the evaluation process. We investigated researches that looked into what are the main types of waste in software projects and how it can be avoided so that we could apply the findings into selecting the projects.

Last but not least, we did a literature review on sentiment analysis as well because we will inspect, among other components of a ticket, comments - we want to analyze the textual representation of comments and see if the overall sentiment influences the time it takes to close a ticket (i.e. maybe a negative conversation in comments might increase the time it takes to solve the issue).

The main sources of information when searching for papers were Google Scholar, ACM digital library and IEEE Xplore digital library. The first step in the literature review process was finding relevant papers that could aid us in our research and splitting them into the categories mentioned above. There were a couple of key search phrases that I started with (along with slight variations):

- software data quality;
- software data quality metrics;
- data quality in software projects;
- issue quality in software projects;
- ticket quality software;
- sentiment analysis;
- software project communication waste.

Then, after reading the papers retrieved when searching for the above-mentioned phrases, I also applied the snowballing research technique as per Dr. Storer's advice. This extra step was helpful and allowed me to easily find linked papers and correlate them in this literature review.

Moreover, when reading papers, one key aspect for selecting and including it in the literature review was citation count. Thus, after finding the most cited papers, I then checked to see the journal where it was published so that we would only have material from the most important journals.

Even though the literature review contains around 40 papers, there were around 80 papers that were reviewed but roughly half of them were excluded because they eventually proved to not be so relevant to our research.

2.2 Data Quality and Metrics

Several authors have investigated the *meaning* of data quality and what characteristics define it.

Bachmann and Bernstein [2009] conducted a thorough investigation of several software projects, both open source (5 projects) as well as closed source (1 project), in order to infer what determines if it has quality. They selected various sources of information, among which bug tracking databases and version control systems logs, and examined SVN logs, CVS logs and the content of the bug tracker databases, in the end trying to link the logs with the bug tracker contents as they are not integrated by default. On the other hand, Strong et al. [1997] conducted a qualitative analysis instead of quantitative by collecting data from 42 data quality projects by interviewing custodians, customers and managers. They analyzed each project using the data quality dimensions as content analysis codes.

The first study came to several conclusions, among which:

- closed source software projects usually exhibit better data quality (i.e. better average number of attachments, better average status changes, better average number of commits per bug report);
- reduced traceability between bug reports and version control logs due to scarce linkage between the two;
- open source projects exhibit reduced quality in change logs as, for example, the Eclipse project has over 20% empty commit messages.

However, the second study reached the conclusion that representational data quality dimensions are underlying causes of accessibility data quality problem patterns. The authors also found out that three underlying causes for users' complaints regarding data not supporting their tasks are incomplete data, inadequately defined or measured data and data that could not be appropriately aggregated.

After defining what data quality is and what are its characteristics, the next step would be how to measure data or information quality in a project.

There are several research papers that try to find the answer, among which the work of Lee et al. [2002]. The authors tried to define an overall model along with an accompanying assessment instrument for quantifying information quality in an organization. The methodology has 3 main steps that need to be followed in order to apply it successfully:

- 2×2 model of what information quality means to managers;
- questionnaire for measuring information quality along the dimensions found in first step;

- two analysis techniques for interpreting the assessments captured by the questionnaire.

After developing the technique, they applied it at 5 different organizations and found that the tool proved to be practical.

Compared to the methodology proposed by Lee et al. [2002], the solution found by Heinrich et al. [2007] is quite different. Even though the main goals were the same, the authors used a single metric, and that is the metric for timeliness (i.e. whether the values of attributes still correspond to the current state of their real world counterparts and whether they are out of date). Thus, they applied the metric at a major German mobile services provider. Due to some Data Quality issues the company was having, they had lower mailing campaign success rates, but after applying the metrics, the company was able to establish a direct connection between the results of measuring data quality and the success rates of campaigns.

Nelson et al. [2005] proposed another technique for measuring data quality in an organization by, firstly, setting 2 main research questions:

- identify a set of antecedents that both drive information and system quality, as well as define the nature of the IT artifact;
- explore data warehousing in general, especially analytical tools, predefined reports and ad hoc queries.

Then, the authors set up a model to define a tree-structured representation of system quality and data quality, as follows:

- data quality - defined by completeness, accuracy, format and currency;
- system quality - defined by reliability, flexibility, integration, response time and accessibility;
- then, data and systems available are evaluated to infer data satisfaction and system satisfaction (coming from customers), which in turn will compute the final satisfaction score of the product.

After conducting a cross-functional survey to test the model, they learned that the features they selected were a good indicator of overall information/data and system quality. They also found that accuracy is the dominant determinant across all three data warehouse technologies, followed by completeness and format. Last but not least, another discovery was that more attention needs to be given to differences across varying technologies.

Another important area of data quality is what methodologies or techniques can be applied by organizations in order to improve data quality. There are several studies that tried to propose such methodologies. A general overview of such techniques was presented by

Pipino et al. [2002] - they wanted to present ways of developing usable data quality methods that organizations can implement in their own internal processes. After reviewing various techniques of assessing data quality in information systems, they reached the conclusion that there is no universal approach to assess data quality as it heavily depends on the context where it is analysed.

One such methodology examined by Pipino et al. [2002] is the technique proposed by Wang [1998] called Total Data Quality Management (abbreviated TDQM). Through this method, the authors wanted to help organizations deliver high quality information products to information consumers by following the TDQM cycle - define, measure, analyse and improve information quality continuously. In order to do that, the research proposes 4 steps: clearly articulate the information product, establish several roles that would be in charge of the information product management, teach information quality assessment and management skills to all information products constituencies and, finally, institutionalize continuous information product improvement. After developing the methodology, the authors conclude on a confident note that their technique can fill a gap in the information quality environment.

However, there are other aspects that can be investigated when trying to improve data quality. For example, Prybutok et al. [2008] tried to find if leadership and information quality can lead to positive outcomes in an e-government's data quality. They conducted a web survey to gather data and test their hypotheses. Afterwards, they assessed the City of Denton's e-government initiatives, including current plans and implementations. They learned that the MBNQA leadership triad (leadership, strategic planning and customer/market focus) had a positive impact on the IT quality triad (information, system and service quality). Moreover, they found out that both leadership and IT quality improved the benefits.

Finally, after defining, measuring and improving data quality, we also need to be able to assess the impact poor DQ has on a company. In this area, there are several authors that investigated various implications of low indices of data quality.

One such research is the one done by Gorla et al. [2010]. They wanted to examine what influences do system quality, service quality and data quality have on an organization, as well as what effect does system quality have on data quality. After conducting a construct measurement for the information systems quality dimensions [Swanson, 1997] and collecting data through empirical testing, they learned that service quality has the greatest impact of all three quality constructs. Moreover, they found out that there is a linkage between system and information quality, fact that previous research did not reveal. Last but not least, their results indicate that the above-mentioned quality dimensions have a significant positive influence on organizational impact either directly or indirectly.

On the other hand, Redman [1998] had different findings. Even though they did not analyze multiple types of quality, but only data quality, they learned that there are three main issues across most enterprises: inaccurate data, inconsistencies across databases and unavailable data necessary for certain operations. Furthermore, they presented a number of impacts that poor data quality has on enterprises: lowered customer satisfaction, increased cost,

lowered employee satisfaction, it becomes more difficult to set strategy, as well as worse decision making.

Another research, the one of Lee and Strong [2003], investigated another aspect: whether knowing-why affects work performance and whether knowledge held by people in different roles affect the overall work performance. After conducting various surveys with 6 companies that served as data collection points, they reached the conclusion that there is vast complexity of knowledge at work in the data production process and if these gaps are not bridged it might lead to decrased data quality.

2.3 Issue Quality

Issue quality is the main topic of this literature review as it revolves around software tickets and what are the characteristics of a well-written and informative bug report, how efficient are bug tracking systems, what defines an efficient bug triaging process etc.

The first and most important sub-topic of issue quality, which is the one we will address in our own research as well, is the quality of bug reports. There are several authors that have tried to defined what makes for a good bug report and what components actually improve the overall quality.

Bettenburg et al. [2008a] have tried, through their work, to analyze what makes for a good bug report through qualitative analysis. They interviewed over 450 developers and asked them what are the most important features for them in a bug report that help them solve the issue quicker. They reached the conclusion that stack traces and steps to reproduce increased the quality of a bug report the most, followed by well-written, grammar error free summaries and descriptions. Last but not least, they also created a tool called CueZilla that could with an accuracy rate between 31 and 48% predict the quality of a bug report.

Strengthening the argument that readability matters considerably in bug report quality is the work of Hooimeijer and Weimer [2007]. They ran an analysis over 27000 bug reports from the Mozilla Firefox project, looking at self-reported severity, readability, daily load, submitter reputation and changes over time. After running the evaluation, they not only found out about the importance of readability in bug reports, but also that attachment and comment counts are valuable for faster triaging and that patch count, for example, does not contribute in the same manner as the previous two.

Another research that agrees that stack traces are helpful in solving bug reports faster is the one performed by Schroter et al. [2010]. They conducted the whole experiment on the Eclipse project. They first extracted the stack traces using the infoZilla tool([Bettenburg et al., 2008c]), followed by linking the stack traces to changes in the source code (change logs) by mining the version repository of the Eclipse project. The results were that around 60% of the bugs that contained stack traces in their reports were fixed in one of the methods in the frame, with 40% being fixed in exactly the first stack frame.

Following on the conclusions of Bettenburg et al. [2008a] we have found out that the investigations undertaken by Bettenburg et al. [2007] reveal very similar results. They performed the same type of evaluation method (i.e. interviews with developers) and confirmed that steps to reproduce and stack traces are the most important features in a bug report that help developers solve the issue quicker.

However, in order to model the quality of a bug report, one needs to be able to successfully extract various types of information from such a report and, if possible, link them to the source code of the project (i.e. source code fragments in bug report discussions should be linked to the corresponding sections in the actual code) or other software artifacts. There are several researches that tried to analyze such techniques and one of them is the work of Bettenburg et al. [2012]. The authors created a tool that could parse a bug report (using fuzzy code search) and extract source code that could then be matched with exact locations in the source code, in the end producing a traceability link (i.e. tuple containing a clone group ID and file paths corresponding to the source code files). Evaluation showed an increase of roughly 20% in total traceability links between issue reports and source code when compared to the current state-of-the-art technique, change log analysis.

Bettenburg et al. [2012] also made use of a tool developed by Bettenburg et al. [2008c] called infoZilla. This application can parse bug reports and correctly extract patches, stack traces, source code and enumerations. When evaluating it on over 160.000 Eclipse bug reports, it proved to have a very high rate of over 97% accuracy.

Predicting the severity of a reported bug Lamkanfi et al. [2010]:

- **What were the goals?** The main goal of the research was to determine if, using textual analysis, one can predict the severity of a bug. This main research question was further divided into 4 RQs:
  - which terms in the textual description could serve as good indicators for the bug severity;
  - what types of textual fields (e.g. summary, description) serve indicate the bug severity best;
  - how many sample does one need to train an accurate predictor;
  - is it better to have a specialized predictor for each component or combine bug reports over multiple components.
- **What was the method?** The authors conducted their research on the Mozilla, Eclipse and GNOME project. The approach can be divided into 5 steps:
  - extract and organize bug reports;
  - preprocess bug reports (i.e. tokenization, stop word removal and stemming);
  - train the classifier on multiple datasets of bug reports (they used a ratio of 70% - training, 30% - evaluation);
  - apply the trained classifier on the evaluation set.

Then, they used precision and recall to evaluate their results.

- **What did they learn?** The authors managed to answer all their 4 research questions:
  - terms such as deadlock, hand, crash, memory or segfault usually indicate a severe bug; however, there are other terms that can indicate the opposite (i.e. non-severe bugs), such as typo;
  - they switched from analysing only the one-line summary to analysing the whole textual description as it contains more detailed information, thus the model can be trained better; however, when they analysed the results and noticed that they had a large number of false positives, which made them switch back to one-line summaries;
  - the classifier needs a large number of datasets in order to predict better on the evaluation set;
  - it mainly depends on the type of project and how it is structured; thus, in Mozilla and GNOME, which share problem-specific characteristics over different components, using a predictor across components sees improvements; however, the dataset needs to be even larger, otherwise the performance starts to decrease.
- **Relevance to our work**: The authors' findings could help us in better understanding textual description analysis and how it can indicate various aspects about a ticket, such as severity. Thus, we can apply the technique described in order to differentiate between severe and non-severe bug reports so that we can focus on a specific category in our own research, if we want that.

Where should we fix this bug?Kim et al. [2013]:

- **What were the goals?** The main goal of the paper was to aid developers in easily finding where to apply the necessary changes in order to fix a bug. The authors wanted to bridge this gap by creating a two-phase recommendation model that would locate the locations where the fixes need to be applied based on the corresponding bug reports.
- **What was the method?** They divided the work into multiple stages:
  - firstly, they did a feature extraction on the bug reports (e.g. extract description, metadata, summary);
  - in order to predict the locations of the necessary fixes, the one-phase recommendation model was put into place - this model would be trained on collected bug reports and then, when given a new bug report, it would try to localize the files that need to be changed automatically;
  - last step was to implement the two-phase recommendation model which basically has two classification phases - binary and multiclass:

    - binary phase filters out uninformative bug reports before predicting the files to fix;
    - multiclass phase where the previously filtered bug reports are used as training data and only after that new bug reports can be analyzed and files to be changed recommended.

- **What did they learn?** Evaluating the model yielded a 70% accuracy in pointing to the correct files to be fixed, as long as the algorithm actually recommended any locations. However, the authors acknowledge that the approach needs extra work, especially in fine-grained defect localization, performance improvement and bug triaging.

- **Relevance to our work**: The research shows interesting points regarding time-to-fix-bug and bug report contents, thus it can give us an insight into how ticket quality can affect that time to fix the bug (in the case of this paper it would mean to successfully pass the first phase of the recommendation model so that fix locations could be proposed).

Automatic Bug Report Assignment Anvik [2006]:

- **What were the goals?** The authors main goal was to determine a way through which bug triaging can be made easier using automatization approaches.

- **What was the method?** The researchs output was an algorithm that can automatically assign bugs to developers based on various types of information fed. The author looked exclusively at ML approaches, examining 8 main types of information (textual description of bug, bug component, OS, hardware, software version, developer who owns the code, current workload of developers, list of devs actively contributing). Moreover, the author considers launching a Firefox extension to test the app even further through human evaluation as well.

- **What did they learn?** The research brings 3 main contributions: methodology for creating semi-automated bug triaging algorithms, a characterization of performance of different approaches, as well as an actual implementation for such a semi-automated big triaget. The author discovered that with further improvements and tweaking, we can reach an even further bug triager.

- **Relevance to our work**: Another project that we can use to learn about bug triaging; also we can contact maybe the author and receive either an executable or the source code for the application.

Reducing the effort of bug report triage Anvik and Murphy [2011]:

- **What were the goals?** The authors had one main goal, and that was to find out if costs related to bug triaging can be reduced if ML algorithms were to be introduced.

This hypothesis was backed by 2 research questions, which were if the approach would create recommenders that make accurate recommendations and if the humans can make use of the information provided by the approach.

- **What was the method?** They created a bug triager which can present a list of developers from which someone can choose who is the most suitable to fix that specific issue. Also, they improved a previous research the same authors worked on, thus increasing the precision and recall of the algorithm. They employed machine learning techniques to create the algorithm. Then, in order to test it, they implemented a proxy to the actual web service thats providing the issue repository, thus visualizing results helped conduct the evaluation process.

- **What did they learn?** The authors learned that their approach could actually be used in software projects as its accuracy is reasonable. They also learned that the impact of poor software project management, including bug triaging, on software projects can be quite high.

- **Relevance to our work**: The research presented can help us in understanding better if statistical expetiments are helpful in analyzing tickets, as well as give an insight into how important ticket quality and proper triaging are for any software projct.

Effects of process maturity on quality, cycle time and effort in software product development Harter et al. [2000]:

- **What were the goals?** The research tries to investigate the relationship between process maturity, quality, cycle time and effort in software projects. The authors are mainly trying to prove/reject a couple of hypotheses: higher levels of process maturity lead to higher product quality in software projects, higher levels of process maturity are associated with increased cycle time in software products, higher product quality is associated with lower cycle time, higher levels of process maturity lead to increased development efforts in the project and higher product quality is associated with lower development effort.

- **What was the method?** In order to test their hypotheses, the authors examined data coming from 30 projects belonging to the systems integration division from a large IT company. The process improvement data was collected via external divisions and by government agencies to provide independent assessments of the firms software development processes. Then, they analyzed a couple of key variables and see how they interact: process maturity, product quality, cycle time, development effort, product size, domain/data/decision complexity and requirements ambiguity.

- **What did they learn?** They learned that the main features they inspected are additively separable and linear. Moreover, they found out that higher levels of process maturity are associated with significantly higher quality, but also with increased cycle times and development efforts. On the other hand, the reductions in cycle time and

effort resulting from improved quality outweigh the marginal increases from achieving more process maturity.

- **Relevance to our work**: It can aid our research through better understanding of how various factors can influence the software development processes, thus giving more value to our own research (i.e. being able to automatically determine issue quality could improve better issue writing guidelines/better rules to be enforced, thus reduced triaging and development costs).

Where should the bugs be fixed?-more accurate information retrieval-based bug localization based on bug reports Zhou et al. [2012]:

- **What were the goals?** The main goal of the paper was to implement a tool that, based on a bug report, could accurately select a number of files where the developer needs to make the necessary changes to fix the issue. They tried to answer 4 main research questions: how many bugs can be successfully lovated by BugLocator, does the revised VSM improve bug localization accuracy, does considering similar bugs improve localization accuracy, can bugLocator outperform other similar methods.
- **What was the method?** The tool firstly performed a textual analysis, looking for similarities between the bug report description and the source code files. Then, it analyzed previous bugs in the repository to find the most similar ones, thus being able to find which files ought to be changed. Lastly, it assigned scores to similar files, the ones with bigger sizes obtaining higher scores as they are more likely to contain bugs. They collected the necessary data from Bugzilla projects and then performed the evaluation.
- **What did they learn?** The buglocator can locate a large percentage of bugs analysing just a small set of source code files. Secondly, the revised VSM outperforms the standard VSM. Moreover, similar bugs can improve the localization accuracy only to a certain extent, while the locator outperformed every other competitor on a multitude of projects.
- **Relevance to our work**: We can try to use BugLocator and even include it in our research as it gives a direct correlation between bug reports and development effort (thus reduced costs). Thus, we can infer even more possible hypotheses regarding our issues quality.

Analyzing and Relating Bug Report Data for Feature Tracking Fischer et al. [2003]:

- **What were the goals?** The main goal of the paper was to analyze the proximity of software features based on modification and problem report data.
- **What was the method?** They employed a method to track features by analyzing and relating bug report data filtered from a release history database. Features are instrumented and tracked, relationships of modification and problem reports to

these features are established, and tracked features are visualized to illustrate their otherwise hidden dependencies.

- **What did they learn?** The authors approach suggest first to instrument and track features, then establish the relationships of modification and problem reports to these features and lastly visualize the tracked features for illustrating their non apparent dependencies.
- **Relevance to our work**: This paper can give us insight into how bug reports can influence feature implementation throughout the lifetime of the ticket.

Duplicate Bug Reports Considered Harmful... Really?Bettenburg et al. [2008b]:

- **What were the goals?** The authors wanted to test two main hypotheses, and these were: duplicate bug reports provide developers with information that was not present in the original report and the information in bug duplicates can improve automated triaging techniques.
- **What was the method?** They collected big amounts of data from the Eclipse open source project in XML form. Then, they ran different kinds of textual and statistical analysis on the data to find answers to their research questions.
- **What did they learn?** They reached the conclusion that bug duplicates contain information that is not present in the master reports. This additional data can be helpful for developers and it can also aid automated triaging techniques (e.g. decide who to assign a bug to).
- **Relevance to our work**: The findings presented in this paper could potentially make us consider duplicate tickets in a different way (i.e. do not treat it as disposable but analyze it and see if it maybe adds more information to the original/master report).

Summarizing Software Artifacts: A Case Study of Bug Reports Rastkar et al. [2010]:

- **What were the goals?** The main goal of the research was to determine if software artifacts could be summarized effectively and automatically so that developers would need only to analyze summaries instead of full software artifacts (i.e. in our case, bug reports).
- **What was the method?** Firstly, they collected data to analyze from 4 different open source projects (Eclipse, Mozilla, Gnome, KDE). Then, they asked the volunteers (i.e. university students) to annotate the bug reports - write a summary of maximum 250 words in their own sentences. These human-produced annotations were then used by algorithms to learn how to effectively summarize a bug report. Afterwards, the authors asked the end users of these bug reports, the software developers, to rate the summaries against the original bug reports.

- **What did they learn?** They learned that existing conversation-based extractive summary generators trained on bug reports produce the best results.
- **Relevance to our work**: As the paper revolves around conversations that are attached to a bug (i.e. comments) instead of the actual description of the bug, we can apply the technique described here maybe to generate summaries and analyze them as well in the overall context of ticket quality (maybe together with some sentiment analysis).

Improving Bug Triage with Bug Tossing Graphs Jeong et al. [2009]:

- **What were the goals?** The two main goals of the authors, through their bug tossing graph idea (based on the Markov property), were to discover developer networks and team structures, as well as help to better assign developers to bug reports.
- **What was the method?** They analyzed 145.000 bug reports from Eclipse and 300.000 from Mozilla. Then, using statistical analysis on the bug reports in order to find evolution histories and changes throughout the lifetime of the reports, they created the bug tossing graph.
- **What did they learn?** They learned that it takes a long time to assign and toss bugs. Additionally, they learned that their model reduces tossing steps by up to 72% and improved the automatic bug assignment by up to 23%.
- **Relevance to our work**: First paper on importance of tossing and its impact on the software development process. It might prove useful as we shall also inspect tossing and bug assignment in our own research in order to determine ticket quality (i.e. maybe tossing actually reduces the quality of the ticket?).

Towards the Next Generation of Bug Tracking Systems Just et al. [2008]:

- **What were the goals?** The authors wanted to determine what makes for good bug reports and, more specifically, how the reporting environment can be changed to best suit the developers needs (and everyone else involved in the software project).
- **What was the method?** They first launched a survey to developers from Eclipse, Mozilla and Apache open source projects from which they received 175 comments back. Then, they applied a card sort in order to organize the comments into hierarchies to deduce a higher level of abstraction and identify common patterns.
- **What did they learn?** They found 7 main topics of improvement for bug tracking systems: provide tool support for users to collect and prepare information that developers need, find volunteers to translate bug reports filed in foreign languages, provide different user interfaces for each user level and give cues to inexperienced reporters on what to report and best practices, reward reporters when they do a good job, integrate reputation into user profiles to mark experienced reporters, provide powerful and easy to use tools to search bug reports, encourage users to submit additional details (provide tools for merging bugs).

- **Relevance to our work**: This is an important paper to our research as it presents, from open source developers perspective, how bug reporting and bug tracking systems should behave. This research gives us various topics to concentrate on when analyzing the quality of tickets.

Bug Report Assignee Recommendation using Activity Profiles Naguib et al. [2013]:

- **What were the goals?** The main goal of the paper is to automatically detect who should be the developers who are most suitable to solve a specific issue.
- **What was the method?** They employed an algorithm using activity profiles (i.e. assign, review, resolve activity of the developer) such that, after detecting the prior experience, developer's role, and involvement in the project, it could recommend who should fix a specific bug. The average accuracy was around 88%, much higher than the LDA-SVM technique.
- **What did they learn?** They found out that their approach was more accurate than previous work done on assignee recommendation, but they think that it might even be improved further by combining their technique with the classic LDA-SVM one.
- **Relevance to our work**: this work complements the research presented in Anvik et al. Anvik et al. [2006] and it will provide us with valuable insight into how automatic triaging should be performed.

Secret Life of Bugs Aranda and Venolia [2009]:

- 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.

Who should fix this bug?Anvik et al. [2006]:

- **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.

Software Quality The Elusive Target Kitchenham and Pfleeger [1996]:

- **What were the goals?** The main goal of the paper is to determine what makes for a good quality software project, as well as who are the people in charge of this aspect and how should they approach achieving it.
- **What was the method?** They tried to define quality in software projects and analyze techniques that measure such quality by looking at other models proposed in different other papers (e.g. McCall's quality model, ISO 9126).

- **What did they learn?** They learned that quality is very hard to define and there are various factors which need to be taken into consideration, such as the business model of the company, the type of the software project (e.g. safety critical, financial) or the actors which are involved and how they coordinate the software activities.
- **Relevance to our work**: this paper is a key paper on software quality and it can prove beneficial in our research by giving valuable insights into how software quality can be modeled, thus helping us in selecting good quality open source repositories to work with (i.e. ticket selection and analysis).

Code Quality Analysis in OSS Stamelos et al. [2002]:

- **What were the goals?** The article tries to discuss and examine the quality of the source code delivered by open source projects.
- **What was the method?** They used a set of tools that could automatically inspect various aspects of source code. The authors analyzed the 6th release of the OpenSUSE project and examined only the components, which are defined by C functions in the programs.
- **What did they learn?** The research's results show that Linux applications have high quality code standards that one might expect in an open source repository, but the quality is lower than the one implied by the standard. More than half of the components were in a high state of quality, but on the other hand, most lower quality components cannot be improved only by applying some corrective actions. Thus, even though not all the source code was in an industrial standards shape, there is definitely room for further improvement and open source repositories proved to be of good quality.
- **Relevance to our work**: this work completes the previous paper on software quality in general by looking specifically at quality in open source projects, which will be our main points for data collection.

Analysis of Software Cost Estimation Grimstad and Jørgensen [2006]:

- **What were the goals?** The authors are trying to show that poor estimation analysis techniques in software projects will lead to wrong conclusions regarding cost estimation accuracy. Moreover, they also propose a framework for better analysis of software cost estimation error.
- **What was the method?** They approached a real-world company where they conducted analysis on their cost estimation techniques.
- **What did they learn?** They learned that regular, straight-forward types of cost estimation analysis techniques error lead them to wrong conclusions.
- **Relevance to our work**: it showed us that we need to be careful when selecting techniques for cost estimation in our own research.

2.4 Measuring Cost and Waste in Software Projects

Waste Identification Korkala and Maurer [2014]:

- **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 Sedano et al. [2017]:

- **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 identificationKorkala and Maurer [2014].

Waste in Kanban Projects Ikonen et al. [2010]:

- **What were the goals?** The authors are trying to find the main sources of waste in Kanban software development projects and categorize/rank them based on severity.
- **What was the method?** A controlled case study research was employed in a company called Software Factory. They conducted semi-structured interviews with 5 of the team members both in the beginning in order to collect data as well as at the end of the whole process to categorize the seven types of waste found. Moreover, they also measured the overall success of the project based on Shenar's techniques (first-second-third-fourth; project efficiency-imapct on the customer-business success-preparing for the future).
- **What did they learn?** They reached two main findings:
  - they found 7 types of waste throughout the project at various development stages:
    * partially done work;
    * extra processes;
    * extra features;
    * task switching;
    * waiting;
    * motion;
    * defects.
  - they reached the conclusion that they couldn't explain the success of the project even though waste was found.
- **Relevance to our work**: this work completes the findings from the previous work presented as most of the projects we will work with will be Agile, thus Kanban-based in terms of issue management.

2.5 Sentiment Analysis

Thumbs Up or Thumbs Down Turney [2002]:

- **What were the goals?** The main goal of the paper is to detect the overall sentiment transmitted through reviews of various types.
- **What was the method?** The author created an unsupervised machine learning algorithm that was evaluated on more than 400 reviews on Epinions on various kinds of markets (e.g. automobiles, movie). The algorithm implementation was divided into three steps:
  - extract phrases containing adjectives or adverbs;
  - estimate the semantic orientation of the phrases;
  - classify the review as recommended or not recommended based on the semantic orientation calculated at previous step.
- **What did they learn?** One thing the author learned that different categories will yield different results. For example, the automobile section on Epinions ranked much higher, 84%, compared to movie reviews, which had an accuracy of 65.83%. Moreover, most pitfalls of the algorithm could be attributed to multiple factors, such as not using a supervised learning system or limitations of PMI-IR.
- **Relevance to our work**: the method can be applied for extracting sentiments from the tickets (description and comments) we will use in our own research.

Recognizing Contextual Polarity Wilson et al. [2005]:

- **What were the goals?** The paper's main goal is to find efficient ways to distinguish between contextual and prior polarity.
- **What was the method?** They used a two step method that used machine learning and a variety of features. The first step classified each phrase which had a clue as either neutral or polar, followed by taking all phrases marked in the previous step and giving them a contextual polarity (e.g. positive, negative, both, neutral).
- **What did they learn?** Through the method the authors employed, they managed to automatically identify the contextual polarity. As most papers were only looking at the sentiment extracted from the overall document, they managed to get valuable results from looking at specific words and phrases.
- **Relevance to our work**: when analyzing the description and comments of the ticket, we can use their method for infering the sentiment transmitted probably more accurately than the technique used in Turney's paperTurney [2002].

2.6 Conclusion

3 Proposed Approach

What are we actually going to do?

Research questions, outline experimental design, independent variables, dependent variables.

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

4 Work Plan

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

References

John Anvik. Automating bug report assignment. pages 937–940, 2006.

John Anvik and Gail C Murphy. Reducing the effort of bug report triage: Recommenders for development-oriented decisions. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 20(3):10, 2011.

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

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

Adrian Bachmann and Abraham Bernstein. Software process data quality and characteristics: a historical view on open and closed source projects. pages 119–128, 2009.

Nicolas Bettenburg, Sascha Just, Adrian Schröter, Cathrin Weiß, Rahul Premraj, and Thomas Zimmermann. Quality of bug reports in eclipse. pages 21–25, 2007.

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

Nicolas Bettenburg, Rahul Premraj, Thomas Zimmermann, and Sunghun Kim. Duplicate bug reports considered harmful really? pages 337–345, 2008b.

Nicolas Bettenburg, Rahul Premraj, Thomas Zimmermann, and Sunghun Kim. Extracting structural information from bug reports. pages 27–30, 2008c.

Nicolas Bettenburg, Stephen W Thomas, and Ahmed E Hassan. Using fuzzy code search to link code fragments in discussions to source code. pages 319–328, 2012.

Michael Fischer, Martin Pinzger, and Harald Gall. Analyzing and relating bug report data for feature tracking. 3:90, 2003.

Narasimhaiah Gorla, Toni M Somers, and Betty Wong. Organizational impact of system quality, information quality, and service quality. *The Journal of Strategic Information Systems*, 19(3):207–228, 2010.

Stein Grimstad and Magne Jørgensen. A framework for the analysis of software cost estimation accuracy. pages 58–65, 2006.

Donald E Harter, Mayuram S Krishnan, and Sandra A Slaughter. Effects of process maturity on quality, cycle time, and effort in software product development. *Management Science*, 46(4):451–466, 2000.

Bernd Heinrich, Marcus Kaiser, and Mathias Klier. Metrics for measuring data quality foundations for an economic data quality management. pages 87–94, 2007.

Pieter Hooimeijer and Westley Weimer. Modeling bug report quality. pages 34–43, 2007.

Marko Ikonen, Petri Kettunen, Nilay Oza, and Pekka Abrahamsson. Exploring the sources of waste in kanban software development projects. pages 376–381, 2010.

Gaeul Jeong, Sunghun Kim, and Thomas Zimmermann. Improving bug triage with bug tossing graphs. pages 111–120, 2009.

Sascha Just, Rahul Premraj, and Thomas Zimmermann. Towards the next generation of bug tracking systems. pages 82–85, 2008.

Dongsun Kim, Yida Tao, Sunghun Kim, and Andreas Zeller. Where should we fix this bug? a two-phase recommendation model. *IEEE transactions on software Engineering*, 39(11):1597–1610, 2013.

Barbara Kitchenham and Shari Lawrence Pfleeger. Software quality: the elusive target [special issues section]. *IEEE software*, 13(1):12–21, 1996.

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.

Ahmed Lamkanfi, Serge Demeyer, Emanuel Giger, and Bart Goethals. Predicting the severity of a reported bug. pages 1–10, 2010.

Yang W Lee and Diane M Strong. Knowing-why about data processes and data quality. *Journal of Management Information Systems*, 20(3):13–39, 2003.

Yang W Lee, Diane M Strong, Beverly K Kahn, and Richard Y Wang. Aimq: a methodology for information quality assessment. *Information & management*, 40(2):133–146, 2002.

Hoda Naguib, Nitesh Narayan, Bernd Br"ugge, and Dina Helal. Bug report assignee recommendation using activity profiles. pages 22–30, 2013.

R Ryan Nelson, Peter A Todd, and Barbara H Wixom. Antecedents of information and system quality: an empirical examination within the context of data warehousing. *Journal of management information systems*, 21(4):199–235, 2005.

Leo L Pipino, Yang W Lee, and Richard Y Wang. Data quality assessment. *Communications of the ACM*, 45(4):211–218, 2002.

Victor R Prybutok, Xiaoni Zhang, and Sherry D Ryan. Evaluating leadership, it quality, and net benefits in an e-government environment. *Information & Management*, 45(3): 143–152, 2008.

Sarah Rastkar, Gail C Murphy, and Gabriel Murray. Summarizing software artifacts: a case study of bug reports. pages 505–514, 2010.

Thomas C Redman. The impact of poor data quality on the typical enterprise. *Communications of the ACM*, 41(2):79–82, 1998.

Adrian Schroter, Adrian Schröter, Nicolas Bettenburg, and Rahul Premraj. Do stack traces help developers fix bugs? pages 118–121, 2010.

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

Ioannis Stamelos, Lefteris Angelis, Apostolos Oikonomou, and Georgios L Bleris. Code quality analysis in open source software development. *Information Systems Journal*, 12 (1):43–60, 2002.

Diane M Strong, Yang W Lee, and Richard Y Wang. Data quality in context. *Communications of the ACM*, 40(5):103–110, 1997.

E Burton Swanson. Maintaining is quality. *Information and Software Technology*, 39(12): 845–850, 1997.

Peter D Turney. Thumbs up or thumbs down?: semantic orientation applied to unsupervised classification of reviews. pages 417–424, 2002.

Richard Y Wang. A product perspective on total data quality management. *Communications of the ACM*, 41(2):58–65, 1998.

Theresa Wilson, Janyce Wiebe, and Paul Hoffmann. Recognizing contextual polarity in phrase-level sentiment analysis. pages 347–354, 2005.

Jian Zhou, Hongyu Zhang, and David Lo. Where should the bugs be fixed?-more accurate information retrieval-based bug localization based on bug reports. pages 14–24, 2012.