INVESTIGATING THE DEFECT PATTERNS DURING THE SOFTWARE DEVELOPMENT PROJECTS

A Paper
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Abhaya Nath Poudyal

In Partial Fulfillment of Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Computer Science

November 2014

Fargo, North Dakota

North Dakota State University
Graduate School

Title

INVESTIGATING THE DEFECT PATTERNS DURING THE SOFTWARE DEVELOPMENT PROJECTS

By

ABHAYA NATH POUDYAL

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Dr. Gursimran Walia
Chair

Dr. Simone Ludwig

Dr. David A Rogers

Approved:

11/12/2014	Dr. Brian M. Slator
Date	Department Chair

ABSTRACT

With the growing software industry and our dependency on complex software applications, it is vital that the software is free of faults. The objective of my study is to, identify and classify the most common type of defects that are committed during the course of a software development project in a real industrial setting. The defects data are collected from two live projects and are categorized into 4 different categories as per the nature of the faults. The data has then been analyzed to see which defect categories are committed most during the software development life cycle.

The result from this study indicates that the category ‘Implementation Defects’ was significantly higher followed by ‘Incorrect/Extra functionality’, ‘Missing Information’ and ‘Ambiguous Information’. ‘Implementation defects’ were further categorized to analyze the main area that led to increased faults and established that ‘Semantic bugs’ were the common faults committed during implementation of an application.

ACKNOWLEDGEMENTS

This research paper would not have been accomplished if it was not for my advisor Dr. Walia. It was only through his constant support that I was able to successfully complete this research. Dr. Walia was not just prompt in his feedback but also provided a logical reasoning with his comments which helped me understand and analyze the topic from various different aspects. His comments and suggestions motivated me to study and gather more information on the topic. He has been very kind and extremely patient throughout the span of research. His interest and concern towards my paper is greatly appreciated. Thank you Dr. Walia for all your effort and time that you put in to make this document a research paper.

I'm very grateful to Dr. Simone Ludwig for taking out the time to be a member of my supervisory committee. I really appreciate her time and kindness.

I would like to thank Dr. David Rogers, my first advisor when I first joined the North Dakota State University for the Bachelors of Science in Electrical Engineering degree (B.S.E.E.) in 2007. His advice have also been valuable and assisted me in planning my course of study to successfully graduate with a B.S.E.E. Thank you so much Dr. David Rogers for setting time aside to be a member of my supervisory committee.

Last but not the least, I'm thankful to the Computer Science Department and all the faculty members and staff who have always been very supportive and responsive to any questions or concerns that I had. It is because of their assistance I could complete my Master's in Software Engineering program from North Dakota State University.

TABLE OF CONTENTS

ABSTRACT iii

ACKNOWLEDGEMENTS iv

LIST OF FIGURES vi

1. INTRODUCTION 1

2. BACKGROUND AND RELATED WORK 4

3. DEFECT DATA COLLECTION 7

    3.1. Selection of the software projects 7

    3.2. Data collection 8

    3.3. Nature of the testing process used in the organization 8

4. DEFECT DATA CATEGORIZATION 10

5. RESULTS 12

6. DISCUSSION 16

7. CONCLUSION AND FUTURE WORK 18

8. REFERENCES 19

LIST OF FIGURES

Figure Page

1: Defect distribution for Project 1 13

2: Further defect distribution for ‘Implementation Defects’ – Project 1 14

3: Defect distribution for Project 2 14

4: Further defect distribution for ‘Implementation Defects’ – Project 2 15

1. INTRODUCTION

Defects can be described as faults/failures that arise in a software application and produce unexpected results. In simple words, defects cause the software application not to behave as is expected. Technology and innovation have reframed our daily activities. The dependency of on these software applications is growing each day, and it certainly becomes very crucial that these applications are free of faults. During the process of application development, faults are inevitable, but, to have a successful application, a thorough and detailed testing of that particular application is necessary with no faults. Companies today spend lot of money to ensure that the application is tested and is defect-free, keeping in mind all possible situations. Finding and fixing defects improves reliability of the software applications. The testing process is complicated due to the fact that the applications which are developed now are complex and testing this complex software becomes a very tedious and stressful task. So to make this complex process run smoothly and correctly, there is a need to find a solution to minimize the number of faults which are even missed with good quality analysis.

Many studies have been performed to minimize the software faults in a software development life cycle. For example researchers have come up with several categorizations of defects that might occur in a software lifecycle and the ways to minimize these faults. Schneider [11] introduced two classes of requirement defects to use when reviewing a user requirement document (missing information and wrong information). Similarly, in 2007, Walia, et al. [13] emphasized the importance of defect classification in requirements and introduced various categorizations to consider when reviewing a requirements document. The categorizations were:

- General
- Missing functionality

- Missing performance
- Missing interface
- Missing environment
- Ambiguous information
- Inconsistent information
- Incorrect or extra functionality
- Wrong section
- Other faults

One paper in software defects showed the importance of categorizing defects based on development faults. The development faults such as semantic bugs, memory bugs, GUI bugs concurrency bugs are the most committed defects found in the software industry [6].

Software defects are of many kinds and can vary with severity or priority such as critical, major, minor and priority 1, 2, 3 and so on. The study conducted by Doolan [15] introduced 3 defects types based on the severity ‘Minor’, ‘Major’ and ‘Super Major’, in order to compare the benefits of these categorization towards cost savings for time invested in the inspection in early stage.

This paper identifies the most common type of defects committed in real industrial settings. The goal is that, if most common defect types are profiled, then effort can be spent to find and fix those problems early in the development cycle which in turn would improve the software quality and reliability.

To accomplish this goal, we collected the defect data from two live projects with duration of 9 and 5 months, respectively. To avoid any biased result, both the chosen projects involved two separate teams from project’s inception to implementation so that there are fewer chances of committing similar kinds of defects by the same team. The data was categorized in 4 categories based on the categories used by different researchers and industrial practitioners [2, 6, 10, 11, 13]. The data was then analyzed to show what kind of defects is most common in industry.

The remainder of the document is organized as follows. Section 2 discusses the details of the previous studies, the type of defect categorization that motivated this study. Section 3 describes the type of project selection and the nature of testing procedure used in the Organization. Section 4 describes defect data categorization used for both of the projects. Section 5 and Section 6 talk about the results and the significance of the results. Section 7 discusses the threats to validity while Section 8 contains the conclusion and suggestion for future work.

2. BACKGROUND AND RELATED WORK

The main crux of this paper is to group defects into categories and analyze them to understand the kind of defects that are committed during software development in a real industrial setting. The categorization of defects can vary on the basis of severity such as critical, major and minor or on the basis of priority. As per the nature of the data collected, our data is categorized after performing an in-depth analysis of different classification schemes presented by many researchers

In 1987, Basili [12], reported insights into the defects found in code and identified two defect classification schemes. In this research, four programs were taken under consideration. These programs were coded with two different languages Simpl-T and Fortran. The defect categorization was done using two categories. The first category included omission and commission. The former was raised as a result of the developer forgetting to include some entity in a module. The later was a result of an incorrect segment in the existing code. The second fault categorization was subcategorized into 6 subcategories:

- Initialization (improper initialization of data structures)
- Computation (incorrect calculation of a value)
- Control (wrong control flow path in a program to be taken for some input)
- Interface (passing an incorrect argument to a procedure)
- Data (incorrect use of a data structure)
- Cosmetic (clerical mistakes when entering a program)

In 1992, Schneider [11] introduced two classes of requirement's defects to utilize them when reviewing user requirement document. In this study, 9 teams with 3 members each from Computer Science Department at the University of Minnesota participated. Each team was provided with 22 pages User Requirement Document. Two classes of defects: Class 1 and Class 2 faults were introduced in the document. Class1 was missing information which consisted of missing functionality, missing features, missing interface, missing performance or missing environment whereas Class 2 faults were wrong information (ambiguous and inconsistent information).

In 2007, Walia et al. [13], researched the importance of defect classification in requirements. The participants of this study were software engineering students, and they were asked to review System Requirement Specification document using the defect checklist. The review was repeated again after the participants were trained in the error abstraction process. It was concluded that more defects were found in the SRS document using error abstraction technique. In this study several fault classes were introduced:

- General
- Missing functionality
- Missing performance
- Missing interface
- Missing environment
- Ambiguous information
- Inconsistent information
- Incorrect or extra functionality
- Wrong section
- Other faults

In 2011, Margarido [2], emphasized on the importance of defects classification on requirements to find the root causes of problems and to take necessary steps to reduce the risks associated with requirements. The authors introduced 9 different defect categories in requirement documents:

- Missing or Incomplete
- Incorrect
- Inconsistent
- Ambiguous or unclear
- Misplaced
- Infeasible or non-verifiable information
- Redundant or duplicate Typo or formatting
- Not relevant or Extraneous

In 2010, Cavalcanti [10] provided an insight on duplicate defects and compared the possible factors that caused defect duplication and its effect on software development. The study was conducted on 8 open source projects and one private project. It was concluded that all 9 projects were being affected by the bug report duplication problem which in turn affected the productivity of the team.

These defect classifications were categorized in light of the project settings in our research and were used to arrive at our defect classification scheme. More details are provided in later sections.

3. DEFECT DATA COLLECTION

This section describes the process of selecting the projects, data collection and the testing process used in the software organization from which the defect data was collected and further analyzed.

3.1. Selection of the software projects

The defect data from two live software development projects were collected and analyzed. Both of the selected projects were '*Web based*' as most of the software companies today prefer a generic interface that allows global access and is more compatible across different platforms. These projects were initiated in a large American multinational corporation with 40,000+ employees. To avoid any chance of having a biased result, these projects were worked upon by two separate teams right from the project's inception to the implementation. Therefore, the nature of the defects made by different people working on these two projects mitigates the likelihood for any bias in the defect sampling. A brief description of the two projects is provided follows:

Project 1 provided monitoring and management solutions for multiple storage systems. There were approximately 449 requirements. The project involved 10 developers and 3 testers. The scheduled level of effort was 9 months.

Project 2 focused on providing backup solutions for a single storage system. The project included 162 requirements and involved 4 developers and 2 testers and was scheduled for duration of 5 months.

3.2. Data collection

In this paper, the defect data from two live software development projects were extracted from the defect tracking tool. The extracted data included: summary of the defect, root cause of the fault provided by the development team, and name of the person to which the fault is assigned. The root cause information is extracted as it is the key to know the cause of the fault and helps analyze the category it belongs to. The fault assignee name is extracted for the defects where the root causes are not clear enough to perform categorization. For such defects the assignee is contacted to know more details about the fault so that the categorization is done correctly. The 'duplicate bugs', 'won't fix bugs' and the 'invalid bugs' were discarded from the dataset and are not considered for further data analysis as these faults are not actual defects**.**

3.3. Nature of the testing process used in the organization

Defect data for both projects was collected and analyzed separately. The collected data was comprised of defects that were raised by the developers and testers during the course of the testing process (i.e., unit, regression and integration) used in the organization. Duplicate bugs, invalid bugs and won't fix bugs are discarded from the data since these are not actual defects. A brief description of the development process and the testing used in the selected projects follows.

Once a certain set of requirements are finalized, the developer implements that functionality and then performs 'unit testing'. The defects are logged by the developer in an internal software defect tracker and are categorized as per the defect severity (i.e., critical, major and minor). The critical defects (i.e., quality assurance show stopper) are fixed while major and minor defects are the ones that are tagged as 'known issues'. After the developer executes the test, it is then ready for the quality analyst to test in a specific environment. A similar procedure is followed by the quality analyst to perform 'regression' and 'integration' testing. Any defect that is found by the quality analyst is also logged in the defect tracking application. Each defect is also assigned a specific developer who had implemented that functionality. Once the developer fixes the defect, the status of the defect changes to ‘ready to test’. The quality analyst is then responsible to retest the same functionality and ensure the previously identified defect is now resolved and no new defects in that functionality are identified.

4. DEFECT DATA CATEGORIZATION

To be able to understand the major types of defects committed during the software development in a real industrial setting, the defect data (collected from two live projects) were categorized into distinct classes. To perform this categorization, a background literature search was conducted to determine the "*defect characterizations*" used by different researchers and industrial practitioners. Analysis of different defect classification schemes (e.g., problem vs. textual specific, interface defects, implementation defects) used by software organizations [2, 6, 10, 11, 13], helped us develop the defect characterization scheme most relevant to our defects (since our defects were defined in defect report forms).

The following five major defect categories were used in our research to categorize different defects (for each project) based on the nature of the defect. The reason of selecting the following defect categorization was to ensure that all phases of the software development life cycle are included from inception through implementation. The defect categorizations performed by different researchers mentioned in previous work were limited to either requirement phase or just the development phase.

1. *Incorrect or Extra Functionality (IF):* A lot of times, information transformed during the software development lead to an incorrect specification of a software system to be developed. This may not pertain to any grammatical mistakes but rather could be an error in the functionality stated in the requirements document. Also, there are instances when the requirement document contains information that should have actually been removed or needs to be updated. Example: The requirement states that the "upon clicking the 'assign' button", a task, would be assigned. However, after further discussion with the business team, this functionally was changed to automate the task assignment process so

that no manual work to assign is required. Hence, this requirement should have been removed or updated in the requirement document.

2. *Missing or Incomplete Information (MI):* These defects arise when the necessary details that are required for a requirement to be successfully implemented are not stated or are missing in the document leading to an incomplete requirement. For example: If the details about the environment in which the application should work is missing. A developer in this case may not have developed the application as MAC OS friendly and hence the application would not function in the MAC OS, giving rise to a defect.
3. *Ambiguous Information (AI):* Ambiguous information is another leading cause of defects as well. The requirements mentioned can be very unclear and lead to more than one interpretation [2]. For example, a requirement states 'The system shall generate the report'. This requirement is ambiguous because the developer is unaware of the: 1) frequency of report generation, 2) how long should it take to generate a report, 3) are all the users authorized to generate the report, and 4) Number of reports that can be generated.
4. *Implementation defects (ID):* Implementation defects are the most common defects found in software industries. The cause of these kinds of defects is developer's ignorance. Semantic bugs, memory bugs and the GUI bugs are the reasons that cause system crashes, data corruption, and unexpected behavior of the system, improper exceptional handling, and corner cases etc. For instance, an erroneous pop up window of NULL pointer exception is displayed due to improper error handling. This can also occur due to errors in developer's logic while implementing.

5. RESULTS

For these two projects, data was collected from the defect tracking system maintained by the software organization. On the basis of the nature of the defect, the data was then analyzed and categorized into different defect types described in the previous section.

Each defect that is listed in the defect tracking system comes with a detailed note that is mentioned by the tester. The tester notes included: the steps that caused the defect so that the developer can reproduce them to fix the defect, the expected result (i.e. the anticipated behavior of the system as per the requirement) and the actual result which is the response received as an outcome of executing the test plan. The actual result may or may not be same as the expected result.

In response, a similar explanation is then provided by the developer who is working on fixing the defect and trying to understand the root cause. This explanation provided by the developer is the key to analyze and classify the defects into various categories. In a few instances an explanations provided by the developer could be: the functionality is not listed in the design document or the design document does not mention any detail about how the process should work.

For defects where the root cause or explanation was not mentioned by the developer or was unclear in the defect tracking tool, for such faults the defect categorization was done only after reaching out to the concerned person and analyzing the explanation provided by them to ensure categorization is done accurately.

After categorization of the defects, the average of categorized faults for each project was calculated and the numbers were compared between both the projects. The brief average summary for both the projects are as follows:

Project #1: Figure 1 shows that 54.33% of defects fall into the category of 'Implementation defects'. This is followed by 'Incorrect/Extra Functionality' 20.67% and 'Missing Information' 14.54% while 'Ambiguous Information' are at a significantly lower percentage than rest of the defects having 10.63%.

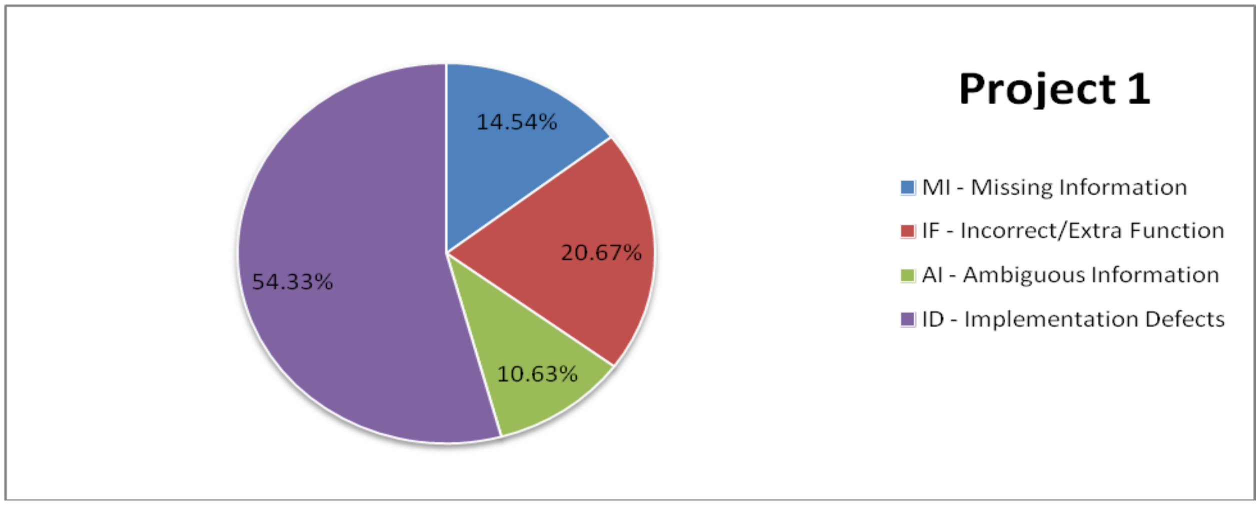

Figure 1: Defect distribution for Project 1

'Implementation Defects' are further classified so as to understand which sub categories lead to more faults (Figure 2). The semantic bugs are significantly higher than other type of faults with 70.23%. This is followed by 'GUI bugs' with 22.99% while 'Memory bugs' shows a significantly lower percentage (6.78%) than other categories.

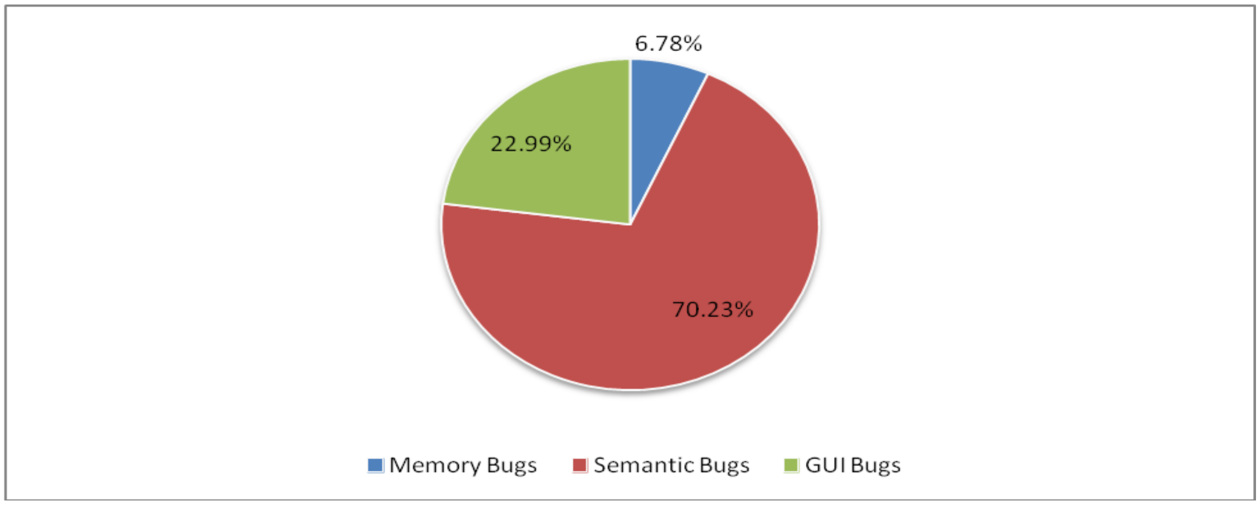

Figure 2: Further defect distribution for ‘Implementation Defects’ – Project 1

Project #2: Figure 3 shows that 70.67% of defects fall into the category of ‘Incorrect Implementation’. This is followed by ’Incorrect/Extra Functionality’, 13.33%, and ’Missing Information’, 9.33%, while ’Ambiguous Information’ carries 6.67%.

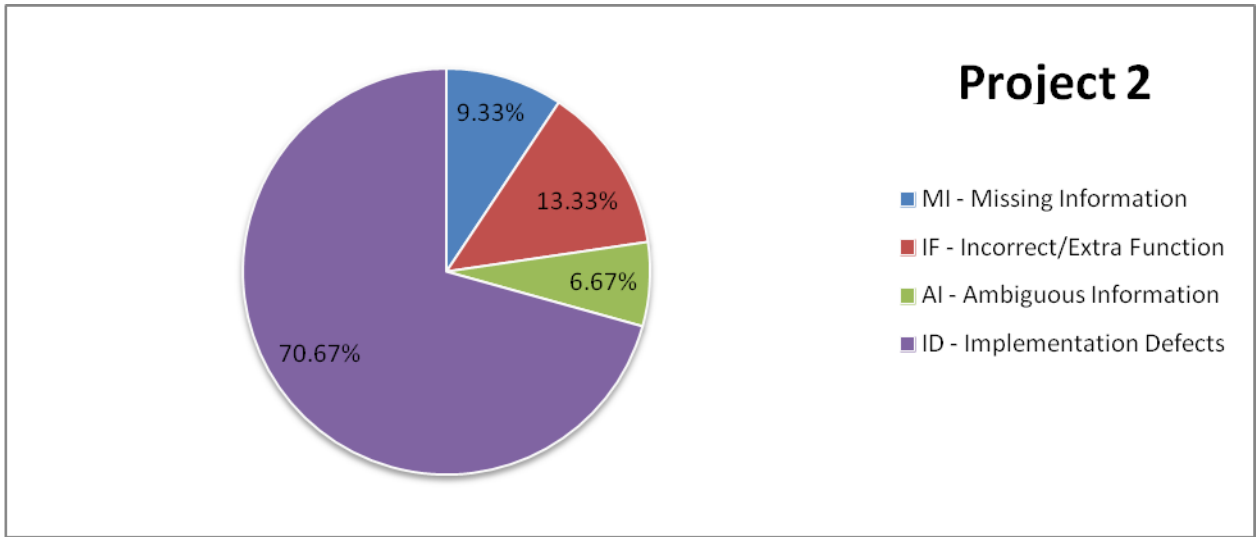

Figure 3: Defect distribution for Project 2

‘Implementation Defects’ are further classified (Figure 4). The semantic bugs are significantly higher than other type of faults with 65.23%. This is followed by GUI bugs with 25.07% while Memory bugs shows significantly lesser percentage, 9.07%, than other catogories.

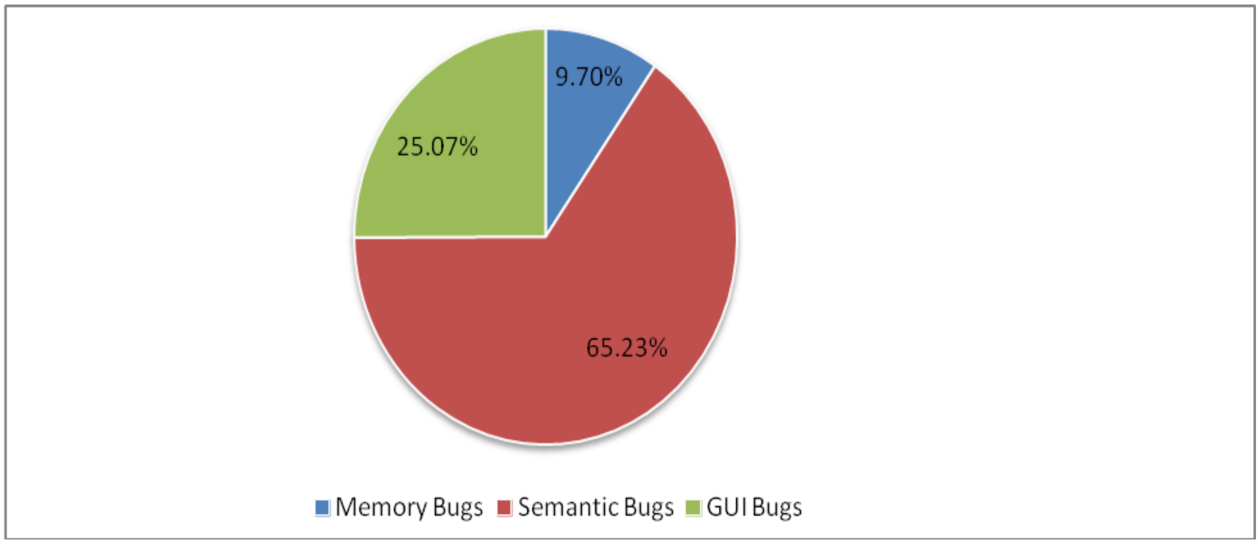

Figure 4: Further defect distribution for ‘Implementation Defects’ – Project 2

6. DISCUSSION

As per the results of the analysis of defects according to their types, ‘Implementation Defects’ occurs significantly higher than other defects categories with 54.33% and 70.67% in Project 1 and Project 2, respectively. This could be due to semantic bugs, for instance using the variable assignment incorrectly, which in return give rises to system availability issues and GUI bugs [6]. ‘Implementation defects’ are further categorized into 3 categories ‘Semantic bugs’, ’Memory bugs’ and ‘GUI bugs’ (Figure 2 and Figure 4). These figure show that Semantic bugs were significantly higher with 70.23% and 65.23% for Project 1 and Project 2, respectively. This could be due to the developer’s lack of understanding of the system. Also semantic bugs are application specific and are difficult to automatically detect by using any tools [6]. ‘GUI bugs’ were 22.99% and 25.07% in Project 1 and Project 2, respectively. The main root cause of GUI bugs is semantic errors [6]. For instance: The semantic errors such as passing incorrect parameters to a stored procedure. This will result in displaying exception pop up window in the GUI hence adding to GUI bugs. Last but not the least, memory bugs count significantly less with 6.79% and 9.70% in Project 1 and Project 2, respectively. The reason behind this could be availability of multiple debugging tools to automatically detect memory bugs such as Purify, Valgrind and Coverity [16].

The defect type ‘Incorrect/Extra functionality’ was approximately 20.67% and 13.33% in Project 1 and Project 2, respectively. This could be due to the lack of domain knowledge while writing or reviewing requirement documents. Lack of detailed domain knowledge can be risky at the development and testing phase. It is a good practice to include subject matter experts while reviewing requirement documents as they are known to have complete domain knowledge hence minimizing the risk of incorrect design, which in turn leads to lesser defects caused by Incorrect or Extra functionality. [15]

Whereas the defect type ‘Missing information’ and ‘Ambiguous information’ was approximately 14.54% and 9.33% in project 1 and project 2, respectively, it is evident that incomplete or unclear requirement can lead to a vague system design document which is not detailed or clear enough for developing any feature, hence creating faults in software application [7, 8, 9].

The main menace of this study are the ‘duplicate’, ‘won’t fix’ and ‘invalid’ filed bugs, during the course of software life cycle. Including these defects might lead to inaccurate results. Another threat to this study could be not filing bugs for small issues. This type of threat might be common in the development phase where a developer fixes any small fault without logging it to the defect tracking system, and thus these defects will not be considered for analysis as there is no proof of any documentation. Moreover, logging bugs and assigning them to the wrong team could also affect the result. For instance, a P4 priority, minor fault (which has less or no impact on the client side) has been logged and assign to the wrong team or component, and the fault had never been looked at.

7. CONCLUSION AND FUTURE WORK

The result of this study offers insights into different categories of defects and analyzes the most common type of defect committed in real industrial settings.   The ‘Implementation Defects’ such as semantic bugs, improper error handling, improper passing parameters to the stored procedure is significantly higher, where as other categories (‘Incorrect/Extra functionality’, ‘Missing Information’ and ‘Ambiguous Information’) were comparably lower in both the selected projects. As a future work, this research could be further extended to see the actual time, cost and resources needed to fix all the faults especially for the defect type ‘Implementation Defects’ which carries significantly higher percentage than other categories in this study. The projects in this study have been selected for specific projects that were initiated in a data storage company. Another extended future scope of this project could be to repeat and analyze multiple projects from various other IT industries such as healthcare, financial or mortgage sectors so as to arrive at a better understanding of the kinds of defects that are most common in these different industrial settings.

8. REFERENCES

[1] B. Freimut, et al., "An Industrial Case Study of Implementing and Validating Defect Classification for Process Improvement and Quality Management," presented at the Proceedings of the 11th IEEE International Software Metrics Symposium, 2005.

[2] Lopes Margarido, I. ; Faria, J.P. ; Vidal, R.M. ; Vieira, M, Classification of Defect Types in Requirements Specifications: Literature Review, Proposal and Assessment, presented Information Systems and Technologies (CISTI), 2011 6th Iberian Conference

[3] T. E. Bell and T. A. Thayer, "Software requirements: Are they really a problem?," presented at the Proceedings of the 2nd international conference on Software engineering, San Francisco, California, United States, 1976.

[4] M. Hamill and G.-P. Katerina, "Common Trends in Software Fault and Failure Data," IEEE Trans. Softw. Eng., vol. 35, pp. 484-496, 2009.

[5] J.-C. Chen and S.-J. Huang, "An empirical analysis of the impact of software development problem factors on software maintainability," Journal of Systems and Software, vol. 82, pp. 981-992 June 2009.

[6] Have things changed now?: an empirical study of bug characteristics in modern open source software, October 2006 ASID '06: Proceedings of the 1st workshop on Architectural and system support for improving software dependability

[7] L. Apfelbaum and J. Doyle, "Model based testing," presented at the 10th International Software Quality Week Conference, San Francisco, 1997.

[8] O. Monkevich, "SDL-based specification and testing strategy for communication network protocols," presented at the Proceedings of the 9th SDL Forum, Montreal, Canada, 1999.

[9] G. Mogyorodi, "Requirements-based testing: an overview," presented at the 39th International Conference and Exhibition on Technology of Object-Oriented Languages and Systems (TOOLS39), 2001.

[10] Y. C. Cavalcanti, E. S. Almeida, C. E. Cunha, D. Lucredio, and S. Meira, “An initial study on the bug report duplication problem,” in Proceedings of the European Conference on Software Maintenance and Reengineering (CSMR), 2010,

[11] An experimental study of fault detection in user requirements documents G. Michael Schneider, Johnny Martin, W. T. Tsai, April 1992 Transactions on Software Engineering and Methodology (TOSEM)

[12] Basili, V.R., and Selby, R.W. “Comparing the Effectiveness of Software Testing Strategies.” IEEE Transactions on Software Engineering, SE-12 (7): 1278-1296, Dec. 1987.

[13] G. S. Walia, et al., "Requirement Error Abstraction and Classification: A Control Group Replicated Study," presented at the Proceedings of the The 18th IEEE International Symposium on Software Reliability, 2007.

[14] Search-based duplicate defect detection: an industrial experience Mehdi Amoui, Nilam Kaushik, Abraham Al-Dabbagh, Ladan Tahvildari, Shimin Li, Weining Liu May 2013 MSR '13: Proceedings of the 10th Working Conference on Mining Software Repositories

[15] Doolan, E.P. “Experience with Fagan’s Inspection Method.” Software-Practice and Experience. 22(2): 173-182. Feb. 1992.

[16] Coverity: Automated error prevention and source code analysis. http://www.coverity.com, 2005