Issue Starvation in Software Development: A Case Study on the Redmine Issue Tracking System Dataset

Md Shamsur Rahim, AZM Ehtesham Chowdhury, Dip Nandi and Mashiour Rahman
Department of Computer Science, American International University-Bangladesh, Dhaka, Bangladesh.
shamsur@aiub.edu

***Abstract*—In computing, starvation refers to the scenario when a process does not get required resources to complete its work. This mainly happens due to very simple priority based scheduling algorithms. Issues in software development require resources too and which issue will get the required resources depend on its priority. So the question is: Does starvation occur in Software Development too? The authors tried to answer the question with the help of their prepared dataset named as "Redmine Dataset". Redmine is one of the popular web-based project management tool as well as an Issue Tracking Systems which also provide role-based access control. Currently, the Redmine ITS has more than 13000 issues and the number of issues is increasing time to time being. The authors have analyzed the Redmine dataset and found that starvation also occurred for issues in Software Development. The authors believe that this finding will steer the Software Engineering community for conducting research on advanced prioritization techniques which will resolve Issue starvation. Furthermore, the authors have provided few future research directions where this dataset can be used.**

***Index Terms*—Dataset; Redmine; Issue Report; Issue Starvation; Mining Software Repository.**

I. INTRODUCTION

Operating systems use various prioritization techniques to allocate the resources for processes efficiently. For poor prioritization techniques, it has found that several low prior processes never get the resources which is referred as process starvation [1]. Similar to the operating system, in our real life, sometimes people also need to wait indefinitely for required resources to get a job done. In software development, different development issues such as bugs, features, patches, customer requests are being tracked and managed by Issue Tracking Systems (ITS). These issues are resolved on priority basis. Higher prior issues are resolved earlier and issues with lower priority resolved later. Hence, starvation may also arise in the software development but to the best of authors' knowledge no evidence has been found till now to support this statement. So, the authors make a hypothesis that issues also suffer from starvation in software development. The authors proposed it as Issue Starvation. To verify the hypothesis, the authors have prepared a dataset which contains the issues from the archive of Redmine ITS and performs analysis to get the valuable insights from the dataset.

The mining software repositories community is playing a noteworthy role by sharing robust and valuable datasets and research outcomes with the software industry. Software practitioners use these results in order to improve their development process. Most of the research have done till now are mainly defect centric analysis by mining software repositories and defect dataset [2][3][4]. But along with software repositories, Issue Tracking Systems (ITSs) have become an integral part of software development.

An ITS is a special software that manages and tracks list of issues like bugs, features, patches and customer requests. The consistent usage of ITS is considered as one of the "hallmarks of a good software team" [5]. As a consequence, the usage of ITS has gained significant popularity among software development practitioners. Due to the enormous popularity, these ITSs have become a great source of data for testing hypotheses regarding maintenance, building prediction models [3].

In this paper, we emphasis on all types of issues reported in Redmine, an open source, cross-platform and cross-database project management web application. This dataset contains all the issues reported in Redmine ITS from its inception to till now (almost a decade). One of the major features of Redmine is: it provides a flexible issue tracking system. Each issue in the system contains several metadata which allow us to investigate the complete life cycle of a reported issue. We believe that rather than focusing only on reported bugs, the focus on whole lifetime of all types of issues can be more effective for verifying the hypothesis. The main contributions of this research are:

i. Verification of the hypothesis proposed by the authors.
ii. The accumulation of different types of reported issues from the issue tracking system of Redmine, regarding itself.
iii. The future research direction in the related field using the dataset.

The following sections discussed in this paper are structured as follows: background study and related works in Section II, how the dataset is obtained and processed in Section III, description of data in section IV, an overview of data in Section V, analysis of the dataset & the proof of concept in Section VI, future research direction and conclusion in Section VII.

II. BACKGROUND STUDY AND RELATED WORK

In order to understand the issue starvation, it is required to understand the life cycle of an issue first. As an issue can be a bug or feature or patch or simply a support ticket, so it may have its own life cycle. To provide a general idea, the authors have discussed the life cycle of a bug in this section along with some issue prioritization methods.

In software development, a defect or bug needs to go through a life cycle to be closed or resolved. A specific life cycle ensures the standard of the bug fixing process. The life cycle contains several stages which are shown in Figure 1 [8].

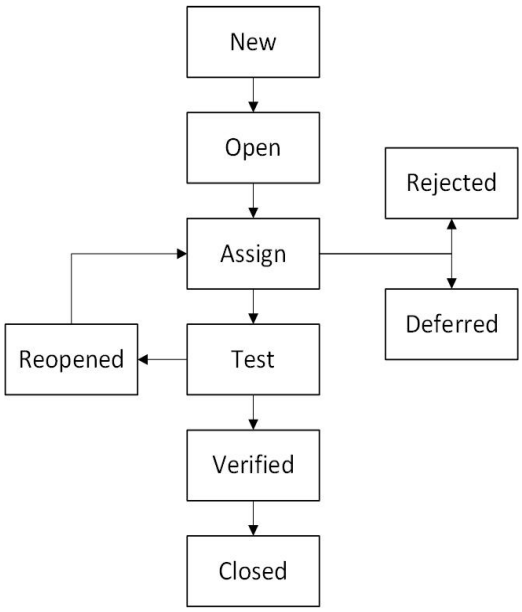

Figure 1: Defect Life Cycle.

When a bug or defect found during the testing phase, the tester/ reviewer first need to check if it is the same bug which has been reported already in the system. If it is not, then the bug is reported as a *New* bug in the IST. After this stage, the status of the bug is assigned as *Open* and a person is assigned to fix the bug. When the assigned person tests the bug then the status of the bug is changed to *Test*. After this stage, the fix of the bug goes through a verification stage. If it is verified, then the status of the bug is changed to *Closed* and that is the end of the bug. But if the testing of the fix fails, the bug reopened again.

From the Figure 1, it is clearly understandable that issue starvation may occur before any stages of the defect life cycle due to the lack of required resources.

The priority of an issue is determined based on some parameters like business value, cost, effort, risk, volatility [9]. Determining priority of an issue is one of the challenging tasks to the practitioners [10]. There are many methods proposed by researchers to prioritize an issue. Some of the common techniques are: Analytical Hierarchy Process (AHP) [11], 100-dollar test [12], Cost-Value Approach [13], Planning Game [14]. Noe of the existing techniques have any mechanisms to detect or handle issue starvation.

III. Data Collection and Processing

Every issue in Redmine has its own URL ending by issue id. The raw-HTML documents of the issues were crawled using the URL and processed later on. The main blocks of the data collection and processing architecture are presented in Figure 2.

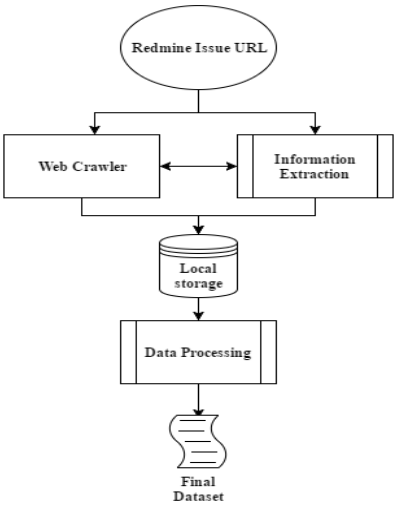

Figure 2: Data Collection and Processing Architecture.

A. *Redmine Issue URL*

The issues in Redmine maintain a common URL pattern, which can be retrieved at http://www.redmine.org/issues/issue_id. This pattern is similar for every type of issue. We find out the total number of issues manually and later on used this number to extract data using web crawler.

B. *Web Crawler*

We have used a web crawler for extracting the HTML pages of Redmine. To create a crawler, we have used the open source tool Selenium WebDriver and programmed it with C#. It is a powerful yet lightweight tool for web automation. For each requested page, the crawler can pass through the Document Object Model (DOM) [6] for searching particular elements.

C. *Information Extraction*

In this process, required information are extracted from the particular elements of the DOM crawled by the web crawler. In addition, we have used Regular Expression (RE) to find out formatted data from the particular elements.

D. *Local Storage*

Initially, we have stored the extracted data into local storage in Comma Separated Values (CSV) format. The main idea behind choosing this format is the ease of writing and processing CSV files.

E. *Data Processing*

The raw data collected in the previous step may contain duplicate, unnecessary, error prone, inaccurate and missing data. In order to increase the accuracy and efficiency of analysis, these inconsistencies need to be removed. So we have used OpenRefine [7], an open source, a powerful tool for processing data.

F. *Final Dataset*

After the data processing step, the final dataset contains 13820 instances with 19 attributes. The final dataset can be retrieved from https://github.com/shamsur-rahim/RedmineDataset.

IV. DESCRIPTION OF DATASET

After crawling and processing the issues from Redmine repository, our dataset contains following attributes:

i. *Issue Id*. It indicates entry number of issues on Redmine repository.
ii. *Tracker*. This attribute contains the type of issues like bug, defect, feature or patch.
iii. *Subject*. Contains the short description about the issue that is being reported.
iv. *Status*. It shows the issues' current state. An issue may be newly opened (new), closed and resolved. An issue can be needed feedback.
v. *Priority*. When an issue has been created on Redmine repository, it can be stated with different level of priority like normal, high and low.
vi. *Category*. This attribute depicts the different categories of issues such as documents, translations, email notification, administration, security etc.
vii. *Author*. It's necessary to identify the specific user who creates the issue.
viii. *Assignee*. To whom the issue is assigned.
ix. *Resolution*. It depicts the issue whether it is a duplicate of another issue or is it reproducible or not.
x. *Progress*. This attribute indicates the current progress on the specific issue.
xi. *Target Version*. It depicts the software version for which the issue has been placed.
xii. *Affected Version*. This indicates the software version that going to be affected unless the correspondent issue has been resolved.
xiii. *Creation Date*. It contains the DateTime of the issue has been created.
xiv. *First Updated Date*. When the correspondent issue has been updated for the first time.
xv. *Last Update Date*. When the correspondent issue has been updated for the last time.
xvi. *Due Date*. The targeted date to complete the issue.
xvii. *Closed Date*. When the correspondent issue has been closed.
xviii. *Count of Reopening*. It depicts the number of reopening of a correspondent issue for resolution.

In regards to the format, the Redmine dataset has been packaged in an XML file (redminedataset.xml). The schema of the XML file has been shown in Figure 3.

Meanwhile, Figure 4 represents the fragments of the XML file content where we can observe the different attributes for an issue. For instance, the *tracker* attribute denotes the type of the issue (defect or feature or patch), issueId represents the unique identifier of the issue and so on.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified"
elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="root">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" name="row">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="tracker" type="xs:string" />
              <xs:element name="issueId" type="xs:unsignedShort" />
              <xs:element name="subject" type="xs:string" />
              <xs:element name="status" type="xs:string" />
              <xs:element name="priority" type="xs:string" />
              <xs:element name="catagory" type="xs:string" />
              <xs:element name="author" type="xs:string" />
              <xs:element name="created" type="xs:string" />
              <xs:element name="lastUpdated" type="xs:string" />
              <xs:element name="startDate" type="xs:string" />
              <xs:element name="dueDate" type="xs:string"  />
              <xs:element name="assignee" type="xs:string" />
              <xs:element name="progress" type="xs:string" />
              <xs:element name="targetVersion" type="xs:string" />
              <xs:element name="affectedVersion" type="xs:unsignedByte"
/>
              <xs:element name="resolation" type="xs:string" />
              <xs:element name="firstUpdate" type="xs:string" />
              <xs:element name="closeDate" type="xs:string" />
              <xs:element name="reOpen" type="xs:unsignedByte" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Figure 3: XML Schema of the Redmine Dataset.

```
.....
<row>
  <tracker>Defect </tracker>
  <issueId>649</issueId>
  <subject>Menu translations broken</subject>
  <status>Closed</status>
  <priority>High</priority>
  <catagory>Translations</catagory>
  <author>Michael Pirogov</author>
  <created>2/13/2008</created>
  <lastUpdated>2/22/2008</lastUpdated>
  <startDate>2/13/2008</startDate>
  <dueDate></dueDate>
  <assignee>Jean-Philippe Lang</assignee>
  <progress>100%</progress>
  <targetVersion>-</targetVersion>
  <affectedVersion></affectedVersion>
  <resolation></resolation>
  <firstUpdate>2/13/2008</firstUpdate>
  <closeDate>2/15/2008</closeDate>
  <reOpen>1</reOpen>
</row>
```

Figure 4: A fragment extracted from redminedataset.xml

V. OVERVIEW OF DATA

In this section, we will provide insights on our dataset. The Redmine dataset contains exactly 13820 issues over the time span of almost 10 years, from December, 2006 to November, 2016. The dataset contains 3 types of issues and it uses 4 types of the tag to represent the priority of each issue. Table 1 denotes the summary of issue types and their priorities.

Table 1
Overview of the Dataset in Terms of Issue Type & Priority

Issue Type \ Priority	Urgent	High	Normal	Low	Total
Defect	221	595	5692	313	6821
Feature	34	159	4588	241	5022
Patch	12	26	1886	53	1977
Total	267	780	12166	607	13820

Table 1 indicates that among all types of issues, detect type issues hold the major share followed by feature and patch. On the other hand, the number of issues with priority *Normal* is the highest with value 12166.

Figure 5 illustrates the count of issues by their status where there are 9717 issues hold *Closed* staus and 3766 issues hold *New* status. The closed issues can be a good source for

predicting the required time or possibility of reopening for the new issues and predicting the priority of upcoming issues.

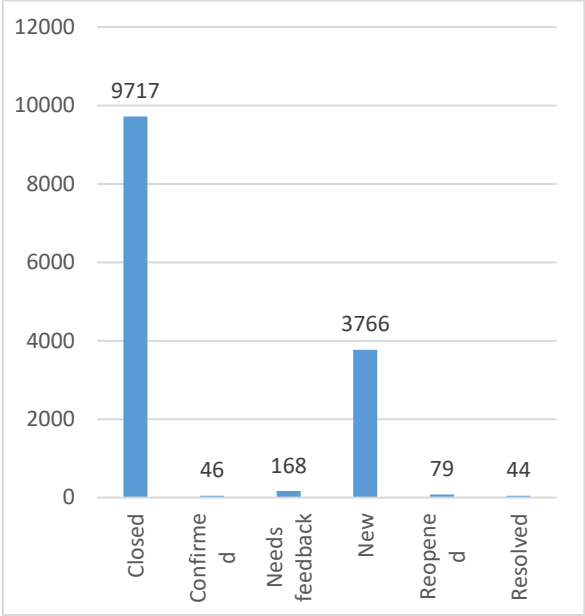

Figure 5: Count of Issues by Status

VI. Data Analysis and Proof of Concept

A statistical analysis has been performed on the prepared dataset. We have filtered the issues having the status of new. We have found that 27.25% (3766 issues out of 13820) issues are still unresolved. Next we calculated the age of each unresolved issue using the following equation:

$$\forall \mathbf{i} \in \mathbb{N}: A_i = C_i - x; \quad (1)$$

where $\mathbf{A_i}$ is the age of issue $\boldsymbol{i}$, $\mathbf{C_i}$ is the creation date of the issue $\boldsymbol{i}$ and $\boldsymbol{x}$ is the present date.

Figure 6 shows that 1247 issues are unresolved, aged about 5 to 10 years old. Meanwhile, Figure 7 shows the priority status of unresolved issues, aged of 6 to 10 years. Among the unresolved issues, about 1150 issues are with normal and high priority.

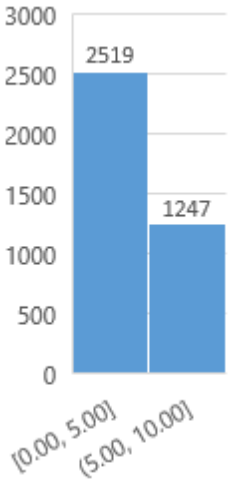

Figure 6: Count of Issues that are unresolved aged of 5 to 10 years

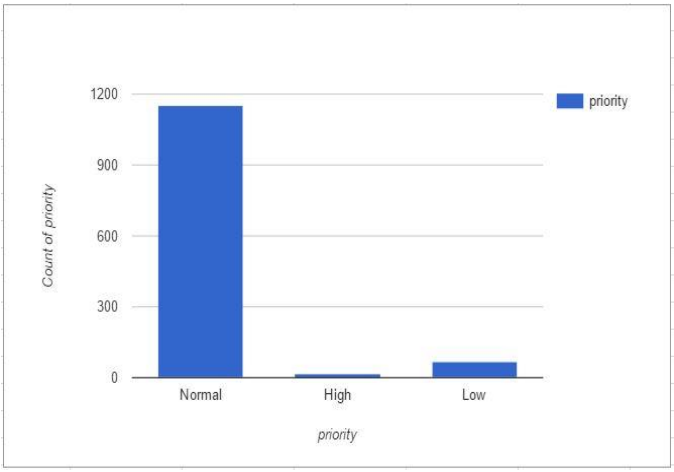

Figure 7: Count of unresolved issues aged 6 to 10 years with priority.

The analysis clearly depicts that due to the existing prioritization techniques, many of the issues are not getting required resources for its completion. This supports the hypothesis regarding issue starvation. As a consequence, we can state that, the issues are also suffering from starvation in software development.

VII. Future Research Direction and Conclusion

Issue response time plays a vital role in software development. We have proved our hypothesis that issues are getting starved due to the lack of advance prioritization techniques. We believe that the finding from this research will result in new prioritization techniques which will overcome Issue Starvation.

We have used the Redmine dataset to analyze issue response time and find out that some issues are facing starvation. We believe that, the usage of Redmine dataset will not be limited to this far. Our dataset can be adopted to:

i. Develop new issue prioritization techniques to improve productivity.
ii. Develop predictive models to analyze the possibility of an issue to be reopened.
iii. Test different hypotheses regarding software developments and maintenance.
iv. Develop models for predicting the priority of issues.

Data gathered from Issue Tracking System (ITS) is essential to perform further research on software engineering. In Redmine ITS repository, we have found several important data fields that can be a vital measure to analyze and understand the pattern of a solution for issues.

References

[1] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Principles*. John Wiley & Sons, 2006.

[2] M. Ortu, G. Destefanis, B. Adams, A. Murgia, M. Marchesi, and R. Tonelli, "The JIRA repository dataset: Understanding social aspects of software development," in *Proceedings of the 11th International Conference on Predictive Models and Data Analytics in Software Engineering*, 2015, pp. 1-4.

[3] A. Lamkanfi, J. Pérez, and S. Demeyer, "The eclipse and mozilla defect tracking dataset: a genuine dataset for mining bug information," in *Proceedings of the 10th Working Conference on Mining Software Repositories*, 2013, pp. 203-206.

[4] D. Spinellis, "A repository with 44 years of unix evolution," in *Proceedings of the 12th Working Conference on Mining Software Repositories*, 2015, pp. 462-465.

[5] J. Spolsky, "Painless Bug Tracking," in *Rock Star Developer News*, 2000. Available at https://www.joelonsoftware.com/2000/11/08/painless-bug-tracking/

[6] J. Marini, *Document Object Model*. McGraw-Hill, Inc., 2002.

[7] R. Verborgh, and M. De Wilde, *Using OpenRefine*. Packt Publishing Ltd., 2013.

[8] B. S. Ainapure, *Software Testing and Quality Assurance*. Technical Publications, 2009.

[9] P. Berander, and A. Andrews, "Requirements prioritization," in *Engineering and Managing Software Requirements*, A. Aurum, and C. Wohlin, Eds. Springer, 2005, 69-94.

[10] M. Rahim, M. Hasan, A. E. Chowdhury, and S. Das, "Software engineering practices and challenges in Bangladesh: A preliminary survey," *Journal Telecommunication, Electronic and Computer*, submitted for publication.

[11] B. Regnell, M. Höst, J. N. och Dag, P. Beremark, and T. Hjelm, "An industrial case study on distributed prioritisation in market-driven requirements engineering for packaged software," *Requirements Engineering*, vol. 6, no. 1, pp. 51-62, 2001.

[12] D. Leffingwell, and D. Widrig. *Managing Software Requirements: A Unified Approach*. Addison-Wesley Profesional, 2000.

[13] J. Karlsson, and K. Ryan, "A cost-value approach for prioritizing requirements," *IEEE Software*, vol. 14, no. 5, pp. 67-74, 1997.

[14] K. Beck, *Extreme Programming Explained: Embrace Change*. Addison-Wesley Profesional, 2000.