**CS 6356.001: Software Maintenance, Evolution & Re-Engineering**

**Assignment 7: Mining Change Request Logs**

**Team 17**

**Team Members:**

* Shivani Talatam
* Zamaan Bawa
* Pritul Dave

**Research Question 1:**

Looking at Change Request 1 for JEdit, why did the Impact Analysis take roughly half the steps in comparison to the steps taken to determine the Concept Location?

In order to find the answer to this question, we will create a table with the number of steps taken to find the Concept Location and compare it with the number of steps taken to determine Impact Analysis. With this data, we can calculate the averages and find other statistics. It is important for us to determine the answer to this question so we can learn if spending more time on finding the concept location can lead to a lower time spent for Impact Analysis and so on. If more time is spent in learning the problem and examining the different files, we will be able to analyze and solve the problem efficiently.

First, let’s look into the requirements of this change request. The students were tasked to update the status bar to include “the word offset of the caret from the beginning of the file and the number of words in the file. As a result of this change, the status bar should report: “2,6 (76/138886)(12/23731)”.” When reading through the change logs, I noticed how many students noted that they had to examine many files in order to find the location. This makes sense, but it still does not explain how students were able to analyze the code quickly and determine what is needed to solve the problem.

Here is the table with our data:

| Team Number | No. of Steps for Concept Location | No. of Steps for Impact Analysis | Time Spent for Concept Location (minutes) | Time Spent for Impact Analysis  (minutes) |
| --- | --- | --- | --- | --- |
| 1 | 9 | 4 | 150 | 120 |
| 2 | 21 | 7 | 65 | 35 |
| 3 | 10 | 2 | 20 | 15 |
| 4 | 7 | 4 | 55 | 15 |
| 5 | 7 | 2 | 75 | 15 |
| 6 | 10 | 2 | 40 | 20 |
| 7 | 9 | 3 | 150 | 30 |
| 8 | 8 | 4 | 30 | 30 |
| 9 | 10 | 4 | 80 | 45 |
| 10 | 7 | 5 | 35 | 30 |
| 11 | 18 | 3 | 41 | 15 |
| 12 | 18 | 3 | 30 | 10 |
| 13 | 9 | 5 | 30 | 20 |
| 14 | 14 | 3 | 60 | 30 |
| 15 | 11 | 5 | 35 | 25 |
| 16 | 7 | 9 | 15 | 40 |
| 17 | 11 | 2 | 45 | 10 |

With these results, we can now calculate the averages for each column:

For the Concept Location steps, we have 10.94.

For the Impact Analysis steps, we have 4.12.

For the Concept Location time taken, we have 56.23 minutes.

For the Impact Analysis time taken, we have 29.7 minutes.

Looking at the averages, we can see our theory is supported. The steps to find the Concept Location were roughly double the steps to complete the Impact Analysis. We can even see it took twice as long in terms of time to do the same thing. This is an interesting finding since many teams had the same outcome in this scenario. Only Team 16 took more steps and time to do the Impact Analysis. This is something we will look into as we start the qualitative analysis. The time spent is a fascinating parameter here since many teams spent either half or even less time on the Impact Analysis. We did have teams such as Team 1 spend much more time compared to the other teams and Team 3 which spent a combined 35 minutes doing both steps. This is another factor we will look into.

Now that we have results which provide a quantifiable perspective, we can now focus on answering the question why it took twice as long to find the Concept Location in comparison to the Impact Analysis. First let’s compare the results from Team 1 and Team 3. Team 1 took 9 steps for the Concept Location versus 10 steps for Team 3, but Team 1 spent 150 minutes in comparison to Team 3’s 20 minutes. The question here is how did Team 3 find the Concept Location so fast? Looking into each of their change requests, we can see that Team 1 spent time trying to learn about the application by using JEdit. On top of that, they examined 15 different classes and also ran a regular expression tool to find the location. The specific classes they visited included: StatusBar.java, JEditBuffer.java and StatusBarOptionPane.java. Team 3, on the other hand, was able to quickly determine the class that needed to be modified. In this case, they found StatusBar.java to be the target. They debugged the class to be certain and marked the class as the Concept Location. We can see a similarity between Team 1 and Team 7. Team 7 spent 150 minutes on finding the Concept Location. They also spent time searching through other classes and analyzed each one which added time. As we look through other teams' change logs, it is certain that the increase in time is related to how detailed their analysis was of each class visited in this process.

Now that we have a general idea of why the Concept Location took longer, let’s see why Impact Analysis was quicker. Going back to Team 3, they spent 15 minutes on Impact Analysis and only needed 2 steps to complete it. They were able to determine the “updateCaretStatus()” method as the method responsible. On top of this, they were able to cancel out other classes such as TextArea.java or StyleEditor.java as classes they would have to potentially modify. Out of all the teams, Team 16 was the only one which took more time with Impact Analysis than Concept Location. In 7 steps and 15 minutes, they were able to use the application, check the TextArea.java and StatusBar.java classes and determine they needed to look into the StatusBar.java class. In their steps taken to complete the Impact Analysis, we can see that they not only analyzed StatusBar.java, but also analyzed other classes such as JEditEmbeddedTextArea.java, TextArea.java, TextAreaDialog.java, Buffer.java and more. They analyzed a total of 9 different classes in the 40 minutes spent in this section. This is certainly an outlier as the other teams primarily focused on the StatusBar.java class. It seems Team 16 may have switched the steps and done Impact Analysis for the Concept Location and vice versa.

The results above show that the increased steps and time for the Concept Location lead to a lower amount for Impact Analysis. Even if it was a minimal amount, there was a correlation between the two. Team 16’s results are the outlier in this study, but looking into their analysis and breakdown in their change request, the results may be switched. Overall, the mining process led to results that answered the question asked and provided results which support the theory.

**Research Question 2:**

**Question:**

What is the impact of the number of method changes during software actualization on the time required for the verification phase, and how can this impact be mitigated?

*The analysis and results are carried out by randomly sampling the logs combined from JEdit and MangoDB and it does not have any dependency to specific change.*

**Motivation:**

The motivation for studying the impact of the number of method changes during software actualization on the time required for the verification phase is that software updates are essential to keep software systems secure, functional, and up-to-date. However, the process of updating software can be time-consuming, costly, and error-prone, especially when there are a large number of changes to the software methods.

If we can understand the impact of the number of method changes during software actualization on the verification phase, we can develop strategies to mitigate this impact and make the software update process more efficient and effective. For example, we may find that reducing the number of method changes during software actualization, or prioritizing certain changes over others, can help to minimize the time required for the verification phase. Alternatively, we may find that certain testing or verification techniques are more effective at identifying issues related to method changes, allowing us to catch and resolve these issues more quickly and efficiently.

**Methodology:**

The methodology adopted for answering the research question involves analyzing the random sampled 125 data points collected from software change logs. Specifically, we use a quantitative approach to determine the relationship between the number of method changes during software actualization and the time required for the verification phase.

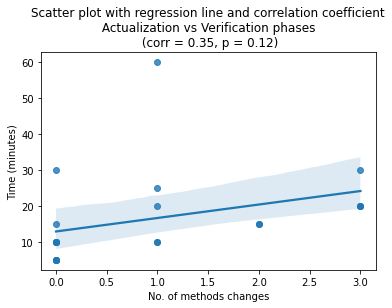
To perform the analysis, we first load the data into a python pandas dataframe and convert the relevant columns to numeric data types. We then filter the data by phase and extract the number of method changes and the time required for the verification phase.

To visualize the relationship between these variables, we use a scatter plot with a regression line. The regression line is fitted using ordinary least squares (OLS) regression, which allows us to examine the trend and strength of the relationship. We also calculate the Pearson correlation coefficient and associated p-value to quantify the strength and significance of the linear relationship between the variables.

Additionally, we inspect the scatter plot for any patterns or outliers that may indicate a nonlinear or complex relationship between the variables. These insights may inform the development of more sophisticated models that better capture the relationship between the variables.

Overall, the analysis aims to provide insights into the impact of the number of method changes during software actualization on the verification phase and suggest possible ways to mitigate this impact. The regression line fitting and Pearson correlation analysis provide a quantitative basis for understanding the relationship between the variables and informing the development of more effective strategies for software updates.

**Results:**



Our analysis reveals that there is a positive correlation between the number of method changes during software actualization and the time required for the verification phase. The correlation coefficient of 0.35 indicates that as the number of method changes increases, there is a slight increase in the time required for the verification phase.

However, the p-value of 0.12 suggests that while the number of method changes may have some impact on the verification phase, there are likely other factors at play as well. For example, the complexity of the changes, the quality of the code, and the expertise of the testers may also influence the time required for verification.

To mitigate the potential impact of the number of method changes on the verification phase, our results suggest that we may need to adopt a more nuanced approach. Rather than simply reducing the number of changes, we may need to prioritize certain changes over others based on their criticality or potential impact on the software system. Additionally, we may need to use more advanced testing and verification techniques to catch and resolve issues related to method changes more quickly and efficiently.

Overall, our results provide some useful insights into the relationship between the number of method changes and the verification phase.

**Explicit Answer:**

Based on our analysis, we have identified a direct correlation between the frequency of method changes during software updates and the duration of the verification phase. Specifically, our findings suggest that a greater number of method changes are associated with a longer verification phase, highlighting the impact of method changes on software development timelines.

**Conclusion:**

The study aims to investigate the impact of the number of method changes during software actualization on the time required for the verification phase and provide suggestions for mitigating this impact. The analysis of 125 data points shows that there is a positive correlation between the number of method changes and the time required for the verification phase. The study shows a positive correlation between the number of method changes and the time required for the verification phase.

Our study shows that changes made to software development methods can impact the time required for the verification process. Specifically, we found a positive correlation between the number of method changes and verification time, indicating that method changes can increase the complexity of the code and thus require more time for verification.

Our research suggests that careful planning and management of software development projects are crucial to minimize delays caused by method changes. By prioritizing thorough testing and minimizing unnecessary changes, developers and project managers can avoid unexpected delays and ensure successful project outcomes.

Our findings highlight the importance of a well-planned testing and verification process in software development. By investing in high-quality testing and verification, developers can ensure that method changes are thoroughly tested and verified, reducing the likelihood of errors and delays in the development process.

In conclusion, our study underscores the importance of thoughtful planning and management of software development projects to ensure that changes to development methods are made with appropriate testing and verification. By prioritizing quality and rigor in the development process, developers can minimize delays, avoid rework, and deliver high-quality software products.

**Research Question 3:**

**Question**

According to the mango log files, the average time taken to perform Change Request 2 in the software development process is significantly less than the average time taken to perform Change Requests 1 and 3. Why?

We were tasked with implementing three change requests for the Mango software and recording the completion time for each change in log files. Change Request 1 required us to modify the software to display values with no more than two decimal places of precision. Change Request 2 involved disabling a specific feature by default, and Change Request 3 was focused on resolving an error that would occur when attempting to change the administrator's password on the 'Users' page of the software.

To analyze the three change requests, we recorded the total completion time for each change and calculated their respective averages. The following table provides details of the total time taken for each log file and its corresponding change request.

| **S.No** | **Change Request 1** | **Change Request 2** | **Change Request 3** |
| --- | --- | --- | --- |
| 1 | 190 | 160 | 148 |
| 2 | 365 | 42 | 170 |
| 3 | 130 | 90 | 320 |
| 4 | 200 | 52 | 480 |
| 5 | 260 | 70 | - |
| 6 | 170 | 95 | - |
| 7 | 155 | 450 | - |
| 8 | 346 | 130 | - |
| 9 | 130 | 120 | - |
| 10 | 114 | 75 | - |
| 11 | 190 | 68 | - |
| 12 | 135 | 90 | - |
| 13 | 147 | 45 | - |
| 14 | 140 | 42 | - |
| 15 | - | 75 | - |
| 16 | - | 37 | - |
| 17 | - | 47 | - |
| **Total=>** | 2672 | 1688 | 1118 |
| **Average=>** | 190.85 | 35.91 | 279.5 |

From the table provided above we can infer that the average time taken to complete Change Request 2 was the lowest among the three change requests, with an average time of 35.91 minutes. In contrast, Change Request 1 and Change Request 3 had average completion times of 190.85 minutes and 279.5 minutes, respectively. These figures suggest that there is a significant difference in the time taken to complete Change Request 2 as compared to the other two requests. The fastest completion time for CR#2 was 37 minutes, while the slowest completion time was 450 minutes.

As we now get the inference that CR#2 was performed quickly by all the teams, we should understand why. So let’s do the analysis step by step.

After examining all the log files, we found that the 'concept location' was the most time-consuming part of the process. The average times are as follows:

Average time for concept location in CR#1 = 94.85

Average time for concept location in CR#2 = 43.11

Average time for concept location in CR#3 = 116.25

The 'concept location' process took significantly less time on average for Change Request 2 because, on average, only two classes were inspected. However, for Change Requests 1 and 3, the number of classes inspected was higher, leading to a longer 'concept location' process.

The 'concept location' process for this Change Request 2, commonly involved searching for a specific keyword, indicating a straightforward process for locating the necessary code.

During the impact analysis of Change Request 2, the teams examined an average of only three classes and four methods to determine the potential impact of the change. This suggests that the change request was relatively contained and did not require significant modifications to the codebase.

In the 'Actualization' step, only one class was modified, with the majority of teams choosing to make changes to the 'User.java' class located in the '/Mango/src/com/serotonin/mango/vo/' directory. This step involved implementing the changes outlined in the change request and updating the codebase accordingly. The fact that only one class needed to be modified suggests that the changes required by the request were relatively straightforward.

Finally, in the 'Verification' step, the teams validated the changes made during the 'Actualization' step. This step was completed quickly and did not require much inspection or modification of classes or methods. This suggests that the changes made during the 'Actualization' step were effective and did not introduce new issues into the codebase.

So we can conclude based on the analysis of the log files and the steps taken during the change request process, it appears that Change Request 2 was the easiest and quickest to implement. This change request involved modifying a specific feature to be disabled by default, and the impact analysis revealed that the changes required were relatively contained, with only a few classes and methods needing to be examined.

**Team Contributions:**

* Shivani Talatam
  + Worked on third research question
  + Effort: 33%
* Zamaan Bawa
  + Worked on first research question
  + Effort: 33.3%
* Pritul Dave
  + Worked on second research question
  + Effort: 33.3%