Predicting Software Evolution with Machine Learning to Enhance Upfront Software Design

José L. Alvarado[1] | Sayed Mohsin Reza[2] | Omar Badreddin, PhD.[2]
University of Puerto Rico, Río Piedras[1]
University of Texas, El Paso[2]

Abstract

This research seeks to employ the use of data analysis to minimize the amount of maintenance required on the long run, making it more efficient. Using different tools to extract software quality data, the bottom line is to gather as much data from previous versions of the software being evaluated to be able to outline improvement areas. By optimizing such areas, the maintenance dependency can be reduced, therefore, improving the overall longevity of software.

Introduction & Background

A common issue within the tech industry is that people have a wrong perception on what is the most expensive and time-consuming stage of the software development process. Therefore, maintenance is one of the most essential parts of software engineering. How can we use computer science skills to make software more efficient and maintenance less expensive? A possible answer is this research's route of data analysis based on historic events of the software.

Methodology

- After **software has been selected**, acquire as many prior versions as possible.

- Versions should have six months in between

- Use Intelli J IDEA with a plugin named CodeMR to **extract software quality data**.

- **Gather the metric variables** that we wish to analyze and **perform the analysis using tools**, such as: python, excel, Apple's numbers, etc.

- **Generate different graphs and figures** that aid us in detecting tendencies, specific problematic classes or abnormal software behavior.

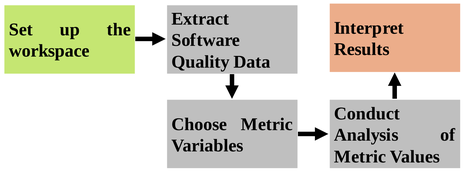

Fig 1. Research Methodology Flow Chart

This information allows us to assess the major areas of improvement within the software so that developers can focus on working those areas to avoid future extensive maintenance.

Results

As we analyze different versions of both repositories, we can notice interesting observations such as the increase in complexity for both repositories from January to March. This might be due to the Corona Virus pandemic which affected the ability of professionals to conduct their regular maintenance procedures

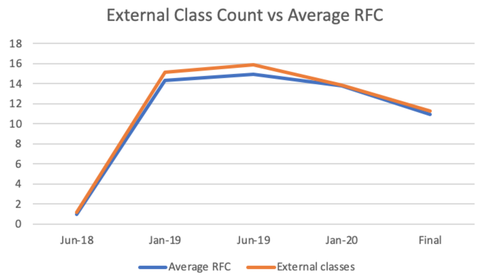

Fig 2. Sky Walking External Class Count vs Avg RFC

- RFC is influenced heavily by the number of external classes where their graphs have a parallel behavior.

- As the **external class count increase**, **the RFC increases as well** and when one decreases, so does the other.

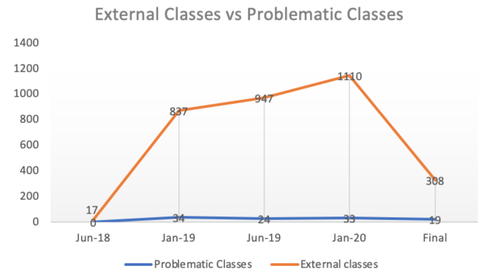

Fig 3. Sky Walking External Class Count vs Problematic Class count

- In figure 3 we can see how there is a **dramatic increase on external classes and an increase on the number of problematic classes.**

- There is **a decrease for later versions** which looks like some degree of **maintenance was able to resume after a few months** of the original corona virus outbreak.

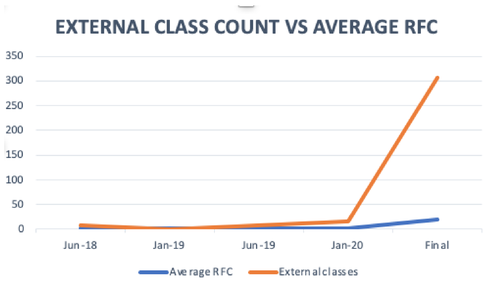

Fig 4. Signal Android External Class Count vs Avg RFC

- In Figure 4 we can corroborate the **RFC keeps growing in relation to the increase in external classes**.

- Although not as drastic as for problematic classes in Figure 5, we can see how there is a **change in slope for RFC at the same time external classes increase.**

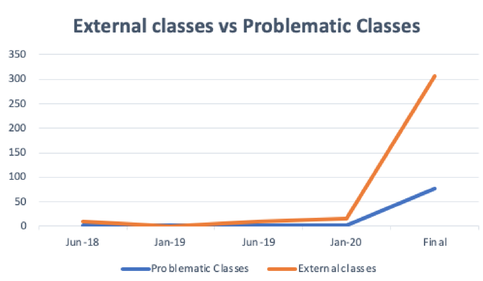

Fig 5. Signal Android External Class Count vs Avg RFC

- Problematic classes increases directly proportional to the increase on the external classes count.

- There **is no drop with the current versions** which presents considerable **issues with software maintenance** with Signal Android.

A common tendency that was observed over time was the number of external classes used in each project increased significantly. With this, both programs saw an increase in their respective average RFC. This is basically the capacity for a class to be called upon or make different calls. This entails a very noticeable increase on the software's complexity.

Conclusion

Producing less maintenance dependent software can help companies minimize the amount of time that is invested towards maintenance. The decrease in maintenance will positively impact companies financially since it reduces the cost of conducting maintenance on a regular basis. More so, reducing the need for maintenance makes the software more trustworthy for the developer and the client. Nevertheless, the most important concept for this research, as seen previously, is teamwork. Without teamwork/collaboration it's very difficult to generate such accurate and useful information to impulse improvement.

Key Takeaways

-Maintenance is the most time consuming and expensive part of software engineering.

-We can see an **increase in the LOC** of both projects by the start of **spring 2020** which could be tied to the **Corona Virus Outbreak**.

-Increase regarding **external classes** also brought an **increase in problematic classes**.

-After the initial COVID-19 outbreak, one of the programs was **able to improve**.

Contact Information:

José L. Alvarado
University of Puerto Rico, Río Piedras Campus
Undergraduate Computer Science Student
jose.alvarado14@upr.edu
(787) 210-2842