Trend Analysis of jiNx of development metrics for jiNx

Capstone Project
SSW 695A Capstone Spring 2021
Dr. Vesonder

Anthem Rukiya J. Wingate

Stevens Institute of Technology
School of Systems and Enterprises

"I pledge on my honor that I have not given or received any unauthorized assistance on this assignment/examination. I further pledge that I have not copied any material from a book, article, the Internet or any other source except where I have expressly cited the source."

Abstract

It is critical to determine protype viability throughout the software development lifecycle of jiNx. Development will validate and verify the performance of each iterative prototype and track metrics corresponding to the time to implement each solution and the time to compile the completed solution for a quantifiable determination of prototype viability. Time to complete and time to compile will be framed as metrics using the GQM process. Data collection and statistical validation of the results will be conducted using tools outlined in the overview of this paper. The measurements observed for each implementation are discussed and reflections on the implications of these observations are composed to support broad scale conclusions determining the viability of the prototype. Discussion of the outcomes lends attention to the limitations encountered by the team while completing this study and finally draws conclusions which briefly reiterate findings and offer insight to potential future utilization and impact of this development pathway.

Keywords: Python, Flask, Machine Learning, GQM

Table of Contents

1. Overview
    a. Introduction
    b. Relevance of Study
    c. Methodology
    d. Data Collection

e. Dataset Variables

f. Dataset Resource

g. Tools

2. Metric Validation

   a. Goals

   b. Questions

   c. Metrics

   d. Mechanism

3. Results and Discussion

   a. Data

   b. Limitations

   c. Reflection

   d. Conclusion

4. References

5. Appendices

   a. Source Code

   b. List of Tables

   c. List of Figures

Amid a pandemic-induced declining global economy, it is notable that software companies are experiencing profit growth and facing increased demand. The implication of this unexpected behavior in the market is that there is an answering provision of high performing product releases to respond to ever-increasing demand. In this project conducted by Anthem Rukiya J. Wingate, the performance metrics of prototypes of jiNx are studied in order to make a quantitative determination of longevity and efficiency.

Relevance of Study

This study has constructed a means by which to model quantitative comparative analyses between prototypes of jiNx. This project seeks to validate comparisons in performance of these prototypes by implementing the application of a quantifiable testing methodology.

Methodology[1]

There are two metrics that have been identified to complete this study. The time to implement the prototype, i.e. "solution" and the time to compile the solution will be recorded in observation and examined in order to draw conclusions illuminating the efficiency of iterative versions of jiNx built and committed to the Github platform. The Goals, Questions, Methods Process (GQM) has been implemented to identify these metrics.

Data Collection. Project data has been collected via metrics compiled for each prototype at time of build. These metrics include a code review of the solution lending particular focus to cyclomatic complexity, shared arguments and operators, and code coverage. Shared arguments and operators in overlapping solution architectures will serve as dataset variables intended to establish correlation between each prototype in order that a valid comparison may be performed using chosen metrics.

Dataset Variables.

The handlers that are shared in common between each prototype are listed in the following tables. These beans are comparable due to commonality of use in solution and relative complexity of solution. Cyclomatic complexity is determined using the following formula:

$$CYC = E - N + 2P$$

P = Methods
E = Libraries
N = Classes

The prototype built in the DiTTo_SpeechToText repository was found to have a cyclomatic complexity of 137. The prototype built in the DiTTo_YoutubePredictor repository was found to have a cyclomatic complexity of 127. The prototype built in the jiNx repository was also found to have a cyclomatic complexity of 127. Notably the cyclomatic complexity is fairly close between these three prototypes. The slight difference between the initial approach engaged in DiTTo_Speech To Text and the final approach engaged in jiNx is likely indicative of the difference in either the number or level of detail in the requirements implemented during these two differing modalities. Supporting evidence of these conclusions were drawn in examination of the Kanban workflows on the Github project board outlined during corresponding weeks leading up to the final commit for each prototype. Note that insights from Githubs analytical software are limited to that which could be obtained after the shift in project direction from DiTTo to jiNx.

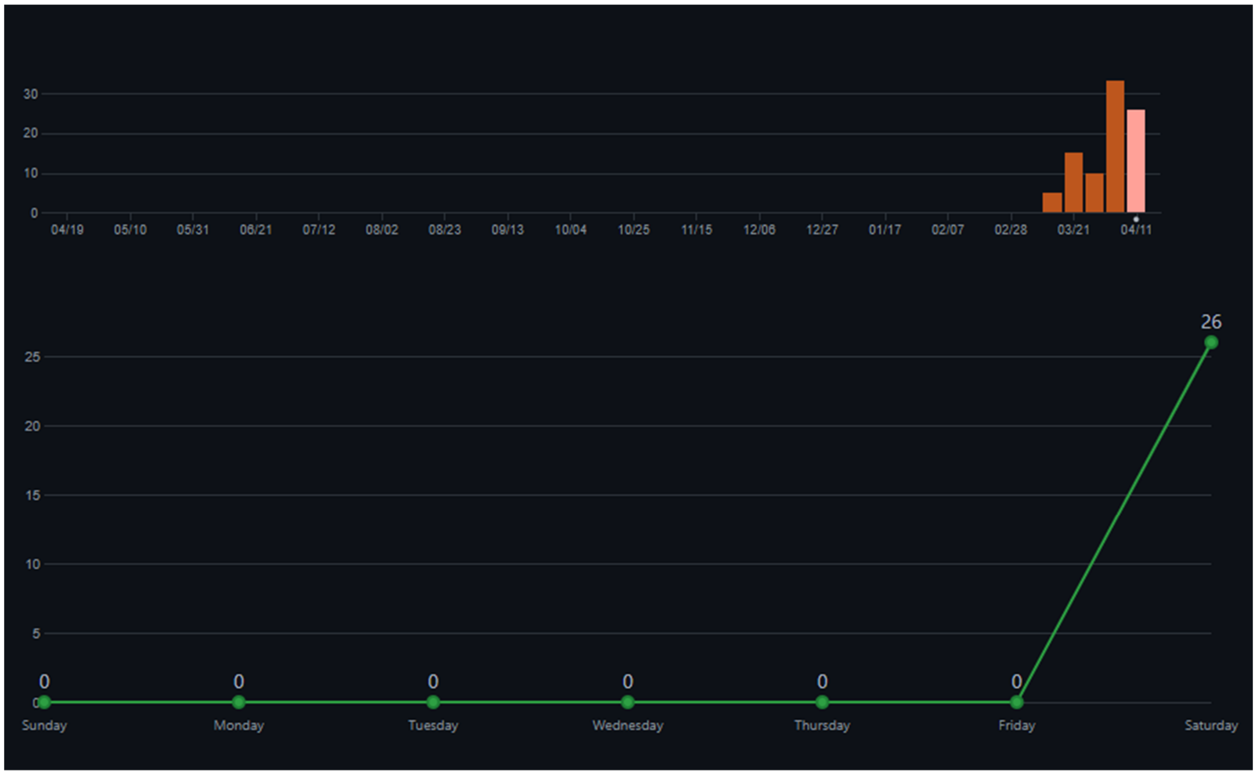


Figure 1

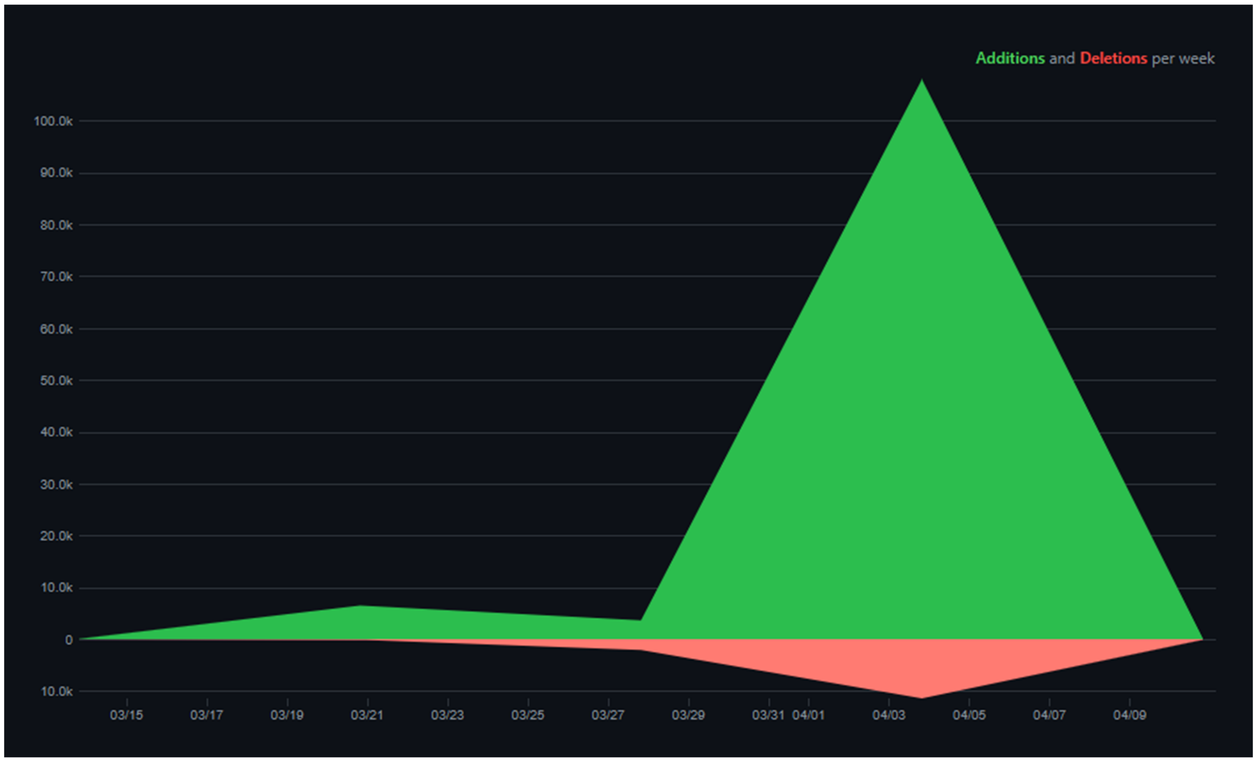


Figure 2

Dataset Variables.

Prototype	DiTTo: SpeechToText	DiTTo: YoutubePredictor	jiNx
Modules	17	9	9
Libraries	60	42	42
Classes	19	13	13
Methods	48	49	49
LOC	1283	939	882
Cylcomatic Complexity	137	127	127

Table 1.1

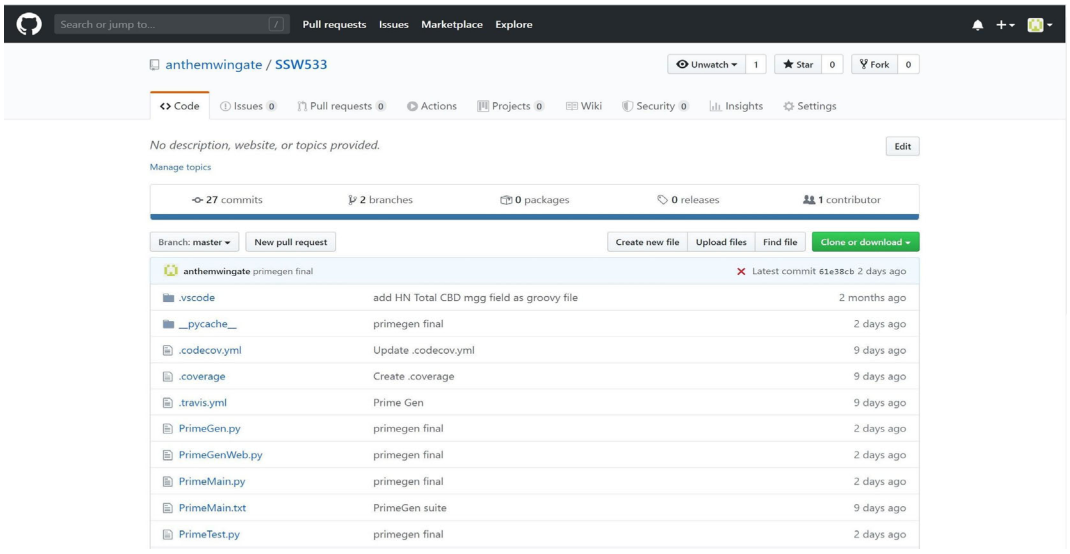
Table 1.3

Dataset Resource. The datasets collected in this study are sourced from the implementation of a solution for a PNG. The code for the PNG was architected using Intellij managed using Github. During the development process, builds of the PNG were continuously integrated using Travis CI and analyzed for Coverage using CodeCov.IO.

Tools.

GitHub

A product of Microsoft Organization, web-based private and public repositories for all levels of collaborative software development.



anthemwingate / SSW533

No description, website, or topics provided.

27 commits · 2 branches · 0 packages · 0 releases · 1 contributor

anthemwingate primegen final · Latest commit 61e38cb 2 days ago

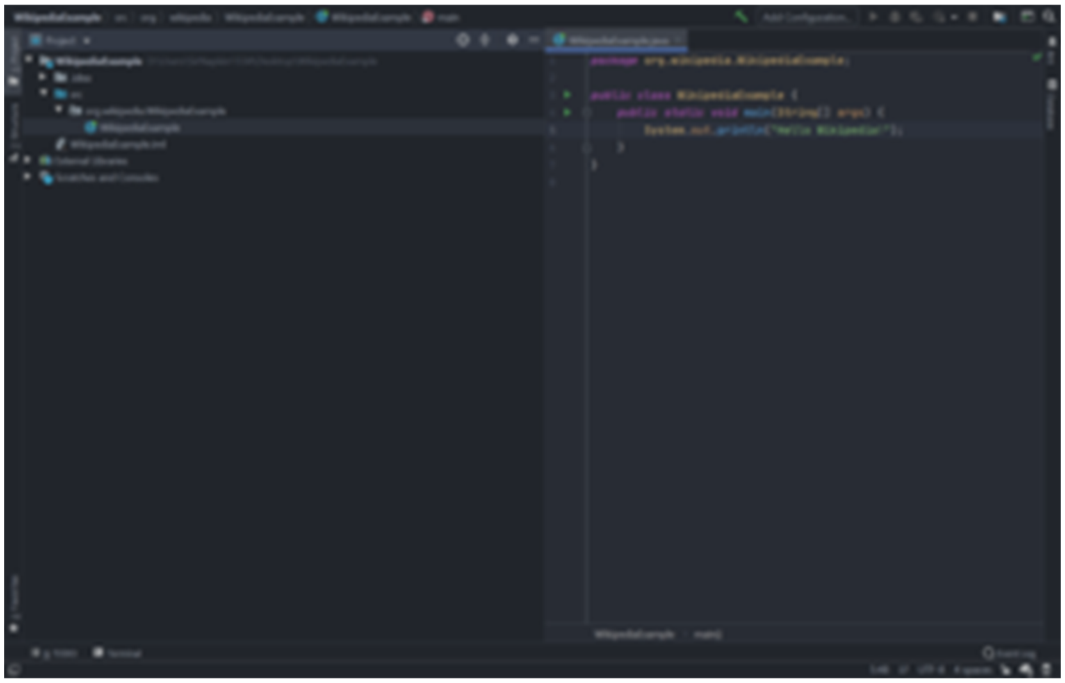
.vscode	add HN Total CBD mgg field as groovy file	2 months ago
__pycache__	primegen final	2 days ago
.codecov.yml	Update .codecov.yml	9 days ago
.coverage	Create .coverage	9 days ago
.travis.yml	Prime Gen	9 days ago
PrimeGen.py	primegen final	2 days ago
PrimeGenWeb.py	primegen final	2 days ago
PrimeMain.py	primegen final	2 days ago
PrimeMain.txt	PrimeGen suite	9 days ago
PrimeTest.py	primegen final	2 days ago

Table 1.2

Figure 3

IntelliJ

Is an integrated development environment developed by JetBrains.



```
package org.wikipedia.WikipediaExample;

public class WikipediaExample {
    public static void main(String[] args) {
        System.out.println("Hello Wikipedia!");
    }
}
```

Figure 4

Metric Validation[2] The purpose of this study is to observe and verify through measurement the viability of each prototype based on performance. "Every company that builds products to sell is concerned with time to market" (Jones 2007). "Process or internal innovations, involve developing new or improved methods of development that can shorten development times, reduce costs, or improve quality" (Jones 2007). The metrics chosen in this GQM process were identified in accordance with a key business driver - development time.

Time to Implement Solution. The proposed metric is a measurement of the duration of time expended from first to final commit for each iterative prototyping cycle. Each week day within the number of days represents an accrual of four hours development time including time allocated for research.

Goals. The goal of this metric is to provide precise measurement for the time required for a developer to architect for a deepening pool of functional requirements. Data acquired in these observations may be analyzed in order to compare relative performance of the prototype as requirements become more detailed and the project progresses toward product release.

Questions. How much time would be required for the average programmer to architect a solution based on relative degree of domain knowledge at the beginning of the iterative cycle? How does knowledge of the domain impact the time expended architecting a solution?

Metrics. The time to architect was recorded and broken up into two categories. These categories are listed based on effort utilized for research and effort utilized for coding. These categories are intended to shape a more accurate picture of the distribution of effort across the total time to architect the solution by providing key insight into how much time was spent researching how to get the solution and building it. It is the expectation of this study that the distribution of effort will vary entirely dependent on the relative degree of domain knowledge acquired at the time of the first commit of the iterative prototyping cycle for each respective prototype.

Mechanisms.

Repository	Development Time (days)
DiTTO_SpeechToText	
DiTTo_Youtube Predictor 2	
jiNx 3	

Table 2

Time to Compile Solution. The proposed metric is a measurement of the duration of time required for the compiler to run the program. The compiler converts the solution into machine code.

Goals. The goal of this metric is to provide a precise measurement of the time required to compile a successful build for the prototype. Data acquired in these observations may be analyzed in order to compare relative performance of each prototype.

Questions. How does increasing knowledge of the domain impact development time and compile time for each successive prototype?

Metrics. The average time of 5 test executions for which the time to compile was recorded.

[T1, T2, T3, T4, T5: n = 5]

Each test was recorded as variable T. Incrementing the subscript for each consecutive test from one to five, yields the variables representing each recorded observation for the time to compile. The average time to compile was then calculated by taking the sum of the five test executions, *T1:T5* and dividing by the total number of tests, n.

$(T1+T2+T3+T4+T5)/5=T_{avg}$

Mechanisms.

Prototype	DiTTo: SpeechToText	DiTTo: YoutubePredictor	jiNx
Compile Time	Average Time (sec)	Average Time (sec)	Average Time (sec)

Table 3

Results and Discussions[3]

Time to Implement.

Repository	Time programming (days)
DiTTo_SpeechToText	12
DiTTo_YoutubePredictor	17
jiNx	11

Table 4

Time to Compile.

Prototype	DiTTo: SpeechToText	DiTTo: YoutubePredictor	jiNx
Compile Time	179 s	132 s	92 s

Table 5

Discussion. Time to implement for jiNx
Overall, jiNx was more expedient in comparison to the preceding implementations as might be expected. With each iterative cycle of development, time applied to research decreases as domain knowledge is expanded. Time to compile for jiNx was also significantly improved. This too can be attributed to the benefit of an iterative prototyping development approach. Based on these numbers, time to implement paired with compile time can be astute metrics when gauging performance. The study may adequately suffice as a means of tracking development activities as well as to design more efficient development pathways for project managers seeking better overall performance.

Limitations. *What is lost with quantitative analysis of prototype performance?*

Nuance is lost with this approach to analysis of performance. There are a multitude of factors which might further mitigate the performance of each iterative prototype that are not addressed here. As an example, requirements were more complex for DiTTo_SpeechToText than in jiNx. In addition, the DiTTo_YoutubePredictor did not implement the multivariate regression algorithm. Finally, jiNx is a fully deployed API whereas the other two incarnations of the project were locally run applications.

Reflections. *What is gained with quantitative analysis of prototype performance?*

These metrics provide insight into the development pathway for those contributing to the code base. A developer seeking to better understand their capabilities might use these metrics to determine what their characteristic levels of production are based on time utilized and time to compile. Identifying successful builds to compare relative cyclomatic complexity in comparison to compile time adds a level of dimension to understanding successful patterns when architecting solutions predicated by a relatively high degree of understanding within the domain.

Conclusion. Quantitative analysis of these performance related metrics using a GQM approach to the implementation of functional requirements cannot be directly correlated to trends of build success. These metrics require only the additional variable of domain understanding to gain sufficient insight into a perceived trend in behavior. As to definitive conclusions that might yield predictions in future performance of prototypes constructed in subsequent phases, it can be stated with confidence that as domain knowledge increases, so too will increase overall efficiency demonstrated by the metrics of development time and compile time.

References[4]

Jones, C. (2007). *Software Engineering Best Practices (Measurements, Metrics And Industry Leadership article)*, New York: McGraw Hill

Department of the Airforce Software Technology Support Center (2003). *Guidelines for Successful Acquisition and Management of Software-Intensive Systems: Weapon Systems Command and Control Systems Management Information Systems (Condensed GSAM Handbook Ch. 8),* Hill AFB: USAF.

Appendices[5]

Source Code.
https://anthemwingate.github.io/jiNx/

List of Figures.

Figure 1	Commits, pg. 3
Figure 2	Code Frequency, pg. 3
Figure 3	GitHub Screenshot, pg. 4
Figure 4	IntelliJ, pg. 5

List of Tables.

Table 1.1, 1.2, 1.3	Dataset Variable, pg. 4
Table 2	Example of Time to Implement data capture mechanism, pg. 8
Table 3	Example of Time to Compile data capture mechanism, pg.
Table 4	Time to Implement data capture mechanism, pg.
Table 5.1, 5.2, 5.3	Time to Compile data capture mechanism for each Github repository, pg.