KBase Automation using Selenium

Design Document

SD-May21-26
Dr. Myra Cohen: Product Owner,
Jake Veatch: Developer,
Daniel Way: Developer,
Daulton Leach: Scrum Master,
Caleb Meyer: Developer,
Hunter Hall: Developer,
Sergey Gernega: Developer
sdmay21-26@iastate.edu
Team Website

Revised: 11-13-2020

Executive Summary

Development Standards & Practices Used

The project will follow the Agile development methodology. It will have an iterative development process where requirements and ideas emerge from collaboration among team members. The engineering standards that apply to this project will include: consistent syntax for filing naming and code styles, the utilization of automation where possible to reduce human error, the use of industry-standard languages and frameworks, Test Driven Development(TDD) that ensures work collaboration, and the use of S.O.L.I.D architecture principles. To insure the project is moving forward we will incorporate weekly advisor meetings, weekly team meetings, the use of slack, and the use of trello for our communication standards.

Summary of Requirements

- Configure Selenium to interface with/run KBase apps
- User set parameters to initiate automation using Selenium
- Intelligently sample the configuration space from set parameters
- Automate the testing of a certain sample configuration
- Use free and open source tools to reduce cost
- Keep the project itself as open source

Applicable Courses from Iowa State University Curriculum

Com S 227, Com S 228, Com S 309, Com S 327, Cpre 288, Com S 311, SE 319, SE 329, SE 339, ENGL 314, SP CM 212

New Skills/Knowledge acquired that was not taught in courses

- Configuration and use of Selenium testing software
- Use of KBase software
- Automating interactions with KBase software
- Automated JUnit tests through Github
- Maven Checkstyle tests to monitor code syntax

Table of Contents

List of figures/tables/symbols/definitions

1 Introduction

1.1 Acknowledgement

The team would like to take this opportunity to thank Dr. Myra Cohen for her significant contributions towards the project and our team's success. Dr. Cohen consistently aided the team's efforts and provided guidance on how to progress the project. Dr. Cohen is an expert in the field and without her vision and knowledge of the task at hand the team would not be where they are today. We want to take this time to thank Dr. Cohen for her efforts and contributions to make our team and project successful.

1.2 Problem and Project Statement

1.2.1 Problem Statement

The Department of Energy's Knowledgebase (KBase) is a platform used by biologists to predict and simulate biological functions. KBase is entirely configurable by the user which results in an exponential number of configurations. Users can control the structure of the simulations they wish to predict/simulate. While this is very advantageous to the user, this causes many failures in the program. Our project aims to build a testing framework leveraging selenium to automatically test KBase configurations. In order to identify program failures, we will need to test all possible outputs and record the results from those tests. Our project aims to: "build a GUI testing framework using an automated testing framework such as Selenium, to (1) infer the configuration model; (2) intelligently sample the configuration space; and (3) automatically test a sample of configurations." (Dr. Myra Cohen). Additionally, the KBase interface was built without automation in mind. In order to find the best biological outcome a user may need to run thousands of tests. Our project aims to alleviate this and give the KBase users an automated tool that can check hundreds of inputs at once.

1.2.2 General Solution approach

The team prototypes a few options for approaching this problem. We landed on using a Selenium web driver to take requests from a GUI containing all configurable options for the KBase tests. The web driver opens up a browser window of the users preferences then automatically fills in the variables on the KBase website for the Flux Balance Analysis (Flux Balance Analysis are a simulated test that replicates and produces data on the growth of an organism in specific conditions). The user is given a simple GUI with all the configurable variables for a FBA to work with. This interface passes variables directly to the web driver.KBase has an exponential combination of potential variable inputs, giving us a fair amount of testing to run. Due to this the user can run our application once and fill parameters from a variety of methods for a variety of different tests. Methods include, reading from a file, randomized, or input by hand. Our program captures the output logs and other relevant information from failing tests and reports that to theKBase developers. The reason for doing this will show us the errors within the KBase application and allow the KBase developers to improve their application via our testing. This is the purpose of our project. The number of combinations to test is exponential, so KBase needs a way to test their software automatically and report the errors to their developers. The project is important to all KBase users. The users we have spoken to have reported errors in the software. With our automated testing and output log collection we can define what exactly goes wrong to replicate errors to the KBase team to allow them to further enhance their platform for all the biologists who rely on it.

1.2.3 Project Output

At the end of the project we hope to have a running standalone application that can be used and built off of to test and run FBA analyses on KBase automatically. Since we only have a year we are keeping the project fully open source to allow the next team to pick up where we leave off to continue improving and propelling the automated testing process further and in turn creating a better KBase application.

1.3 Operational Environment

The end product is expected to be run and available constantly to any KBase users. The environment the software application runs in will be the user's own system. Our application will be downloaded by each KBase user on their own machines. Our application is purely software therefore the environment is limited to the user's system. The only hazard that our project could encounter is no internet connection for the user. Our application needs an internet connection to operate. This environmental hazard is out primarily out of our control as developers.

1.4 Requirements

Functional

- Configure Selenium to interface with KBase apps
- User sets parameters to imitate automation using Selenium
- App will sample the configuration space from set parameters
- Automate the testing of certain sample configurations
- Will collect collect output data from KBase
- Users will be able to randomize inputs
- All inputs from the KBase GUI will be available

Non-Functional

- Will not noticeable disrupt KBase traffic flow
- App Gui will have similar layout as KBase GUI
- App will have a dark mode

Environmental

- Will require a valid and active KBase account
- Will require a internet connection
- Must be run on device which supports Selenium

Economic

- Will be open source and use open source resources

1.5 Intended Users and Uses

The intended users of this application will be those who are already using the KBase GUI. The purpose of this application is to allow the automation of the KBase GUI, therefore it is expected that the users of this app will be family with the KBase system and how to use it. No experience beyond what is already required for KBase operation will be needed in order to operate the app.

The standard use of this app will be that once the user opens it they will enter in their selenium credentials. The app will then open a web driver and log into selenium on their behalf. Once logged in the user will be able to select a narrative they wish to automate. Once all inputs have been configured for the narrative the app will then run the KBase narrative and collect output for each set of configurations of the user input.

1.6 Assumptions and Limitations

Assumptions

- A user will have a working KBase account.
- A user will be familiar with KBase.
- A user will have a working internet connection.

Limitations

- Will not store a user's KBase credentials, in order to avoid security risks.
- The product will be open source and not cost users to use.
- The app will not store collected data in a database.

1.7 Expected End Product and Deliverables

There will be one deliverable. A stand-alone application that uses Selenium to interface with the KBase GUI. This app will have an interface GUI that will be similar in presentation as a KBase narrative GUI, but with the additional options of presetting multiple configurations to run including randomizing variables as well as auto configuring combinations. The app will run these configurations through the KBase system on the user's behalf by opening a web browser of the users choosing and configuring the KBase narrative based on the configurations input by the user. Any resultant data that KBase Produces will be collected by the system and stored as a file.

2 Project Plan

2.1 Task Decomposition

KBase Interface Analysis

Analysis of the KBase web application to better-understand the user's workflow and how the web pages are built. This will inform how we might interact with and automate the interface. Particular deliverables from this task may include specification of workflows for interaction with KBase and identification of particular DOM selectors or risks we could encounter as we automate.

Automation Technology Analysis

Analysis of automation technologies like Selenium to understand which might best-fit our use-case. This will include considerations like the performance of the tool (can we parallel jobs?), support of the tool (does it accommodate modern browsers and devices?), and ease-of-use (will interaction with the framework require programming knowledge?). Deliverables should include identification of a best-fit automation technology and steps necessary to implement an MVP using the technology.

User Interface Input Implementation

This project will involve the implementation of a basic Java GUI to accept parameter specification from our users with validation and potentially advanced range specification. This may be a simple Java Swing interface with inputs/form-controls curated for the particular KBase application we're targeting, which is a Flux Balance Analysis (FBA).

KBase FBA Programming Automation

A Selenium routine must be developed to take the parameters specified through our Java Swing GUI and automation programming the KBase application (Flux Balance Analysis) with those parameters. Programming will involve a variety of challenges including accommodating timing of certain elements (dialogs with fade-outs, for instance) and interaction with a variety of primitive and rich inputs (checkbox vs. a multiple-selection from a set of media).

KBase FBA Execution and Management Automation

Once programmed, we must automate execution and monitoring of a KBase Flux Balance Analysis application. This includes verifying the parameter programming is complete, interacting with the "Run" button, and monitoring the status updates to ensure processing is proceeding successfully and indicating to the collection step once processing completes.

KBase FBA Collection and Review Automation

After processing completes successfully, we will need to collect results from the KBase Flux Balance Analysis by identifying and scraping particular elements from the webpage. These may include the objective value of the simulation and output logs from the job. These values should then be written to the output data structure of the job to be processed or visualized later.

Job Queue and Runner Implementation

The user may enter parameters with many permutations, so to automate execution we will develop a job queue to hold those parameter sets. This queue will be asynchronously linked to the GUI and runners, and will handle delegating jobs to runners and reporting the outcome of a job's processing to the GUI. The runners are responsible for executing the Selenium automation for a given job.

User Interface Results Display

Once a job has completed, we will need to report this back to the user. This may include a basic tabular view, as well as a status view for jobs which have been queued but may not have completed processing yet. This view may indicate aggregate values as well, with a later task.

User Interface Export Functionality

Once a set of parameters has completed execution, the user should have the opportunity to export that data for further evaluation in a tool like Excel. A set of jobs results should be exportable to the CSV file format from some GUI interaction, like an export dialog.

Multi-Format Parameter Specification

Parameters may be specified for the application in a variety of formats depending on the parameter's type. Explicit values like "true" or "1" maybe be specified, a set of explicit values may be specified like "1, 4, 10", a range may be specified like "10-100", or randomness may be applied either uniformly or with preference for extremes. THe permutations of these ranges/inputs will need to be processed into a discrete set of jobs and parameters which can then be queued and processed.

Result Aggregation and Identification

Once a set of jobs and parameters have completed processing, their results can be aggregated and outliers may be identifier for the user. These could be visualized through the UI or exported to some other format, like a CSV. We might attempt to programmatically identify outliers in the data and indicate them to the user.

2.2 Risks And Risk Management/Mitigation

TASK	RISK	RISK MITIGATION
KBase Interface Analysis	**0**	
Analysis of KBase interface behavior	**0**	
Automation Technology Analysis	**0.5 - There is a risk that the automation technology that we choose to use becomes deprecated and no longer supported and/or unstable.**	**We mitigate this by choosing a well supported and mature automation framework such as Selenium. We also utilize Maven to manage our dependencies to prevent auto-updating and creating potential breaking changes from updates.**
User Interface Input Implementation	**1 - There are many UI frameworks to choose from. Some may have limitations on the quality of the UI.**	**We chose to use Java Swing, a well-defined and documented library to build our UI and have designed our application in a way that we could create a new UI to collect requirements if necessary.**
KBase FBA Programming Automation	**1 - All software is inherently flawed.**	**We will ruthlessly unit and regression test the application to ensure the highest standard of code quality. We also have automated tests that run continuously during code contribution to ensure passing tests.**

KBase FBA Collection and Review Automation	**2 - Given that the project is automating the interaction of a GUI, there are high risks that the GUI (which we do not maintain) will change over time. This can break the automation if elements are not where they should be.**	**We have designed our interactions in a way that a broken element location can be changed with the swap of a single string, describing the location of that element. Reducing the amount of developer work needed to resolve the issue.**

2.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

- Take job input from a user in multiple forms such a custom GUI form, or text files such as CSV. These input forms should include various input types, such as boolean, float, or free-form strings. Be able to take two different formats of job input.
- Execute jobs from a user on KBase with multithreaded runners. This allows users to run multiple jobs at the same time. Be able to run at least five runners at one time.
- Collect output from the jobs and report them back to the user in a format of their choosing, such as a text/CSV file. Be able to report output to a user in two different formats.
- Iterate on the project by increasing the amount of KBase applications, inputs, and outputs that can be created by the project.
- Show runner feedback to the user through the GUI or CLI to display progress of jobs. Be able to show runner feedback in the console and through the GUI.

2.4 Project Timeline/Schedule

Our project development will follow an agile format, and the application's implementation Fwill be performed in roughly three phases. The first phase will be "Analysis" and occur during the first-half of the Fall semester. From August through September, our team will perform analysis on the KBase system as well as technologies we may use to automate aspects of the system.

Following analysis, we will implement the basic proof-of-concept functionality for automation KBase through a testing framework, such as Selenium. This basic implementation will complete by the end of November, which will begin an 8 week hiatus before the Spring semester begins.

The Spring semester will include more advanced feature implementation that builds atop the basic functionalities implemented in the Fall semester. These more advanced functionalities include job queuing, multiple runners, multi-format parameter specification, and result aggregation and identification. This more advanced implementation continues through the Spring semester.

Implementation of tasks within these three phases will generally align with iterations in our Agile development process, where we produce a deliverable for our client/advisor and determine an appropriate deliverable for the next iteration's completion. This process allows the client/advisor to redirect our efforts in a continual feedback process, which should enable us to produce a solution which aligns well with the client's use-case. The Gantt chart above details exact dates for task/iteration implementation.

2.5 Project Tracking Procedures

We will use git as our version control by storing our code changes. Git will also be used to lay out issues and teammates can self assign issues according to their strengths. Trello will be used to efficiently map out which tasks need to be completed, are being worked on, and are completed to remove ambiguity. Slack will be used to communicate easily between team members any issues, code snippets, or general help that is needed. We will meet weekly as a team to iron out any issues

throughout the week and give updates on project progress. We also will meet weekly with our client to gain requirements, demonstrate progress, and ask any questions that would be beneficial to the growth of the project.

2.6 Personnel Effort Requirements

This table is a breakdown of the effort requirements for each team member for a standard two week sprint

Task	Number of Hours	Explanation
Research	1-2	Topics may require research before designing a solution
Software/project design	1-2	Solutions should be well thought out before implementation to promote optimal solutions
Software development	5-10	Creating software solutions
Software Testing	1-3	Testing software for bugs and to make sure it performs required task/s
Reviewing team pull requests	1-2	Reviewing team members pull requests to ensure their code meets team standards
Team meetings	1-2	Meetings with clients to demo product and obtain future project requirements and team meetings to assign project requirements and plan sprints

2.7 Other Resource Requirements

There are various outside resources that must be used in the completion of our project. Resources are used to either enable functionality in the project, or aid in the development process of the project.

We have open-sourced our project on GitHub with an MIT license to open the development to anyone upon completion of the project. This also serves as a remote repo and collaboration tool for the team in the development of the project. Developers on the team utilize IntelliJ or Eclipse IDEs

(depending on user preference) to work on the project. The team also utilizes GitHub Actions to automatically build the project and run automated unit tests and check-styles on code formatting. The rest of the resource requirements are open-sourced software, available through Maven to utilize in adding functionality to the application. Dependencies such as Java Swing, and Maven-Checkstyle.

2.8 Financial Requirements

Total financial resources required to complete the project will equal to $0.00 . The reason the project development will be free is KBase is an open source project, the technologies and licenses required to develop are free and or have been provided through being a student of Iowa State. Selenium, IDE's we choose, frameworks, version control, and other technologies are free to use and require no financial resources.

3 Design

3.1 Previous Work And Literature

The interesting thing about our project is the fact that it is using a common testing utility to embark on something that hasn't been attempted before on our specific platform. In the end we are trying to create a platform that allows KBase developers to test an exponentially large combination of complex inputs automatically. This in itself has never been attempted before. Since the KBase web application is highly specialized in its own GUI there currently is not a "competing" product/application available. Something we consider special and challenging about our project is that we have no previous experiences from other developers/companies to give us a rough idea of how to attack this problem. We are essentially inventing the wheel. As for relevant background, a working knowledge of KBase is necessary as well as credentials to operate the KBase application. In order to learn more about KBase it is necessary to read what KBase is and get familiar with running Flux Balance Analysis (With an authenticated account). To Familiarize with KBase we referenced KBases website itself [1]. Additionally, our implementation of this automatic testing procedure relies heavily on use of Selenium. This being the case, we all needed to have strong working knowledge of how to operate Selenium. This can be obtained from Selenium's web page itself [1]. Aside from these two utilities we are developing and prototyping solutions to this project. To gain more insight on KBase as a whole and what the KBase team is looking for, we have met with a member of the KBase team. This has allowed us to gather more information to create a better solution. Since this has never been attempted before, part of the project is determining if this is a feasible solution to the project. Having a working knowledge of these utilities is the only prerequisite for this project. We are not able to follow other projects due to the nature of this particular project.

1. See our reference table at the end of the document for more information (Section 6.2)

3.2 Design Thinking

Our project has a few aspects that are shaping our design. The KBase developers need a stand alone application to test their advanced algorithms. The Flux Balance Analysis has a wide range of potential inputs. The inputs range from from simple booleans to floats to lists. Each boolean input raises the potential combination of inputs by an exponent of 2 meanwhile, the floats increase the potential combinations by an exponent of the range of tolerated inputs. From this short example we can see how testing all possible combinations is infeasible. This is the problem we are tasked with solving. We gained insight on this problem during the empathize step and we have defined this issue in the define step. This problem statement shapes our applications design. We need a standalone product that can efficiently test an exponential amount of inputs, this will automatically run these tests based on user specifications and return a log of the outputs. This is the core problem we are trying to solve with our product. For our application it was pretty clear cut that Selenium was the way to go. As a group, we thought about and decided on Selenium as the main driver of our project. Other design decisions were considering the development module to use for the project. The only one that made sense to the team to use was Agile development.

3.3 Proposed Design

So far, our team has performed detailed analysis around the KBase Web Interface, and specifically around the "Narrative" interface which is a derivative of Jupyter notebooks. From the analysis and consultation with our advisor, we've determined that Selenium is the most-appropriate tool for our project's purposes.

As we move into implementation of our KBase Testing Automation project, we've decided-on a modular architecture where the GUI we develop, the Selenium configuration automation, Selenium data collection automation, and export functionalities are all kept separate. This is desirable for several reasons which satisfy both functional and non-functional requirements for the project. First, it is optimal for a distributed team with work being performed in-parallel. With distinct modules and clearly-defined interfaces between them, we can develop two interdependent modules without concern for integration problems. Secondly, this architecture makes unit and integration testing very simple. As mentioned, the interfaces to these modules are well-defined and can be tested with ease. Lastly, but most importantly, this architecture leaves us an opportunity to extend the system for other inputs, parameter configurations, and export formats. This architecture also abides by common software engineering standards by being clearly laid-out and familiar, so future extension and support can be performed by unfamiliar developers.

More materially, this design involves the following: a GUI module where a Java Swing UI lives for interaction with the user to produce jobs, a Job Manager module which handles queueing and assigning jobs to runners, Runners which handle executing jobs against KBase, and export modules which handle exporting the final data format from those jobs to some external format, like a CSV.

3.4 Technology Considerations

A main goal of this project is to keep it open source and let the next generation of inspired developers pick up where we left off and take the project further. Due to this the project is designed to work on multiple IDEs (Eclipse, Intellij). Other technologies we take advantage of is Selenium as it's a core component of our project. Selenium has many strengths being a free, language independent automation framework. Selenium has plenty of documentation and has been very easy for the team to pick up and integrate. Selenium does have its drawbacks and trade offs. Selenium has a high initial cost as well as strong dependencies on third party applications to drive the

software. Selenium Can also make testing a bit of a hassle due to the timing and automation of the web driving services. All this in consideration we still feel Selenium is the best choice for our project. To solve some of these issues we need to produce high quality and clean code for our project. This will help efficiency especially when running exponentially growing amounts of tests.

3.5 Design Analysis

Our proposed design from 3.3 works well because it accommodates the functional and nonfunctional requirements of this project. It enables us to develop effectively, even in a distributed team, but also grants us the efficiency of an architecture intended for automation of the KBase system with complex parameter inputs. So far, the greatest problem for us has been the variability in KBase's interface construction. Elements may move, their identifiers may change, and an unexpected feature change to KBase could render our system broken. We will continue to investigate creative methods of identifying controls, labels, and features of the KBase application so our automation is more robust against these sort of unexpected changes.

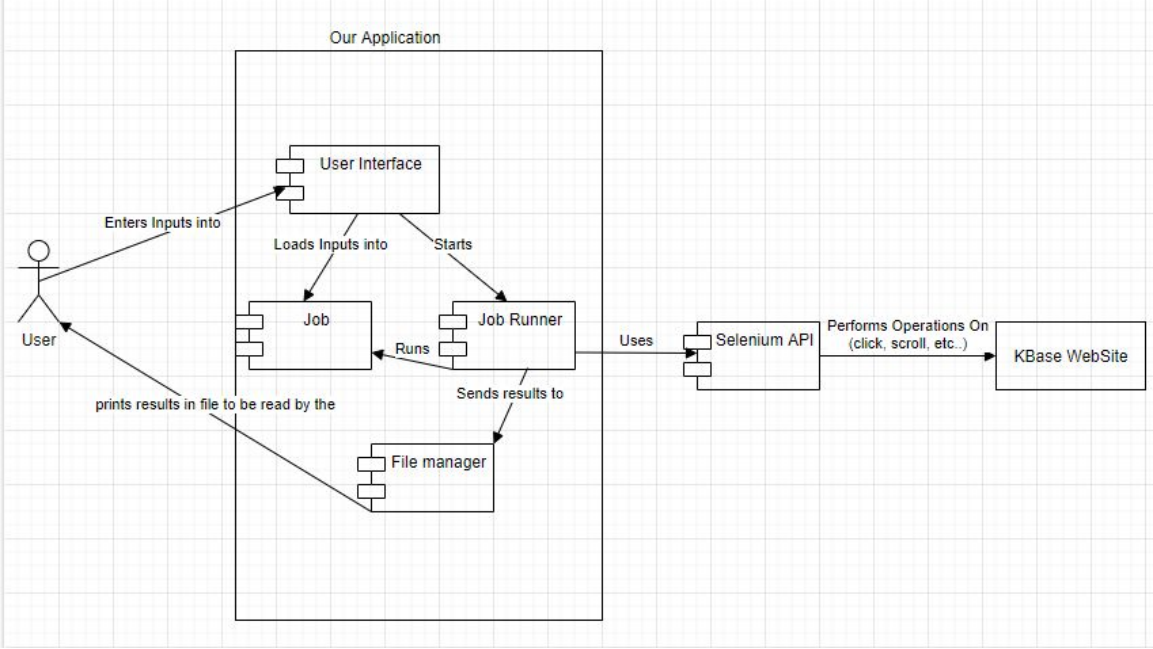
3.6 Development Process

The design process we are following is an Agile design process. The team thinks very highly of the Agile development process. With Bi-weekly sprints we can keep the client in the loop constantly and allow them to input feedback and ideas quickly and efficiently. We are creating a product for the KBase developers and users alike. Using Agile development we felt this was the best way to keep these users in mind and in the loop. Agile development also gives the benefit of allowing us to change/adjust requirements per the user. If the user wants X and we have Y using Agile methodology we can easily shift to X. Using Agile we keep our client up to date with the project's status and gives us an accurate idea of when certain features will be implemented. Due to the reasons stated above the team believes that an Agile approach is the best one.

3.7 Design Plan

Following the use case laid out in 1.5. The design plan is to have a single user interface that a user will interact with. The Interface will take in user input and place it into a job. The interface will then start a job runner to run the job. A job runner runs a job by looking at the parameters the job holds and uses the selenium API to perform the appropriate operations on the KBase website. Once the results are available on KBase the runner collects the results and sends them to a file manager. The file manager organizes the results and writes them to a file for the user to read later.

This set up satisfies the requirements that "User sets parameters to imitate automation using Selenium", "Configure Selenium to interface with KBase"," Automate the testing of certain sample configurations', "Will collect output data from KBase". By separating the job from the job runners, we can let the user queue up several jobs, while limiting the number of jobs that can run at the same time. This is to meet the requirement "Will not noticeably disrupt KBase traffic flow".



Use Case Diagram

User Interface	Interface for user to input parameters and create jobs
Job	Stores parameters
Job Runner	Takes the parameters stored in a job and performs operation on the KBase site using selenium. Collects results from KBase and sends them to File Manager
File Manager	Organizes Results and writes them to a file

Module Descriptions

4 Testing

4.1 Unit Testing

The KBase testing framework we're developing has several distinct modules. These were kept separate intentionally as part of the architectural design the group worked on early in the project. In addition to granting us the ability to extend the system later, with new inputs, runners, or outputs, it has also granted us the ability to test individual units of our application. For instance, once a job has had its parameters set, it must be handed off to a runner for execution. Our Selenium runner will move through several phases as it configures the KBase interface, executes the job, and collects results from the Flux Balance Analysis's output. These individual steps may be

tested in isolation through unit tests to validate that the correct actions are being performed in KBase through the Selenium API, and that the output collected from the application's DOM is being correctly instantiated into an output data structure for later processing.

4.2 Interface Testing

We do not do any interface testing for our project. Our interface changes periodically and is minimalistic so our team has decided that interface testing would not be as beneficial as other types of testing. In the future, we may do some broad interface testing that is automated by Selenium.

4.3 Acceptance Testing

Our accepting testing is mostly manual performance tests that are performed by our team while working on the project. This means that each of the team members are responsible for manually testing the code that they plan to implement to ensure that it works in accordance with project requirements. In addition, the people who review the pull request of any given team member should make sure that the code works either by asking the team member who created the pull request or by manual testing that the code works themselves. Lastly we demo the project every two weeks to our client so that they can assess the project and determine if it meets their requirements.

4.4 Results

We have automated unit testing and syntax checking in our repository. So far, it has helped us to catch test and syntax failures before contributing them to our repository. We can also run unit tests locally during development to ensure that we are not making breaking changes as we develop.

We have a difficult time testing Selenium itself given that is an interaction layer with a UI. We have mocked out Selenium for our unit testing. We are going to continue adding tests as we develop the code to cover our logic that we are adding to the project.

5 Implementation

5.1 Overview

The team is utilizing an agile development strategy and development has been in process the entire semester. From initial meetings between the team and our advisor it was settled that an agile design process would be the best way to attack this problem. The project required us to test and prototype some potential solutions and utilizing an agile methodology we could do this efficiently. Implementation was broken down into multiple smaller teams that worked on components individually that would later be combined to further progress towards our working prototype. The teams broke down to work on implementing: GUI that passes parameters to the Selenium web driver, Web driver programming, output collection, and application set up. The goal for the end of the fall semester was to have a working prototype of all these components combined together. The

plan for next semester is to polish these features and continue implementing additional features that the KBase team can leverage.

5.2 Fall Progress

As we stated in the previous section the goal for this semester was to have a working prototype that our team, advisor, and KBase users could use. To do this we needed to accomplish a number of tasks that the group divided using Git Issues. The team chose tasks that interested them and we plugged away working in Sprints of two weeks. Every two weeks the goal was to show a new feature(s) to keep our client in the loop. The main tasks we needed to accomplish were: Generic setup, GUI for the user to feed information to the selenium driver, Program the selenium driver to use those values, Run the FBA with those values, gather output, send output to a file the user can view for informational purposes. Due to our focus on developing in sprints rather than using other development methodology we have accomplished all these tasks. At the current state the product is usable by the client and could be utilized to assist in testing the KBase software. We have a fully functioning GUI allowing the user to enter the main parameters (excluding a few complicated, optional ones) which are then passed off to the selenium web driver in the form of a job object. This job gives all the details the user requested to the driver. The driver then executes the job and the results are sent as output in a JSON file. We plan to expand this application to include the complex input parameters, ability to run multiple jobs, extract results of many jobs to a CSV file and many more additional features to fully flesh out the application. These will be the primary focus in the Spring semester. Our objective from here is to progress the project as far as possible and have a fully functioning application that future teams can pick up and take even further for the benefit of KBase.

6 Closing Material

6.1 Conclusion

So far, we have implemented a prototype of our application as a standalone Java executable. We have a codebase set up to interact, authenticate, program inputs, run jobs, and collect output from KBase. We can run jobs through the entire perspective of a user.

Moving forward, we are working on making the application parallelizable, faster, easier, and more configurable for the user. We want to add new parameters to program into a KBase job. We also have goals to allow users to run jobs in a variable number of concurrent processes.

We are developing in an agile format. This may not be the optimal solution, but that will be discovered as we continue to develop and iterate on the project. Other solutions that we did review did not have the portability of a standalone Java application, and we wanted users to be able to run this on any device they may have.

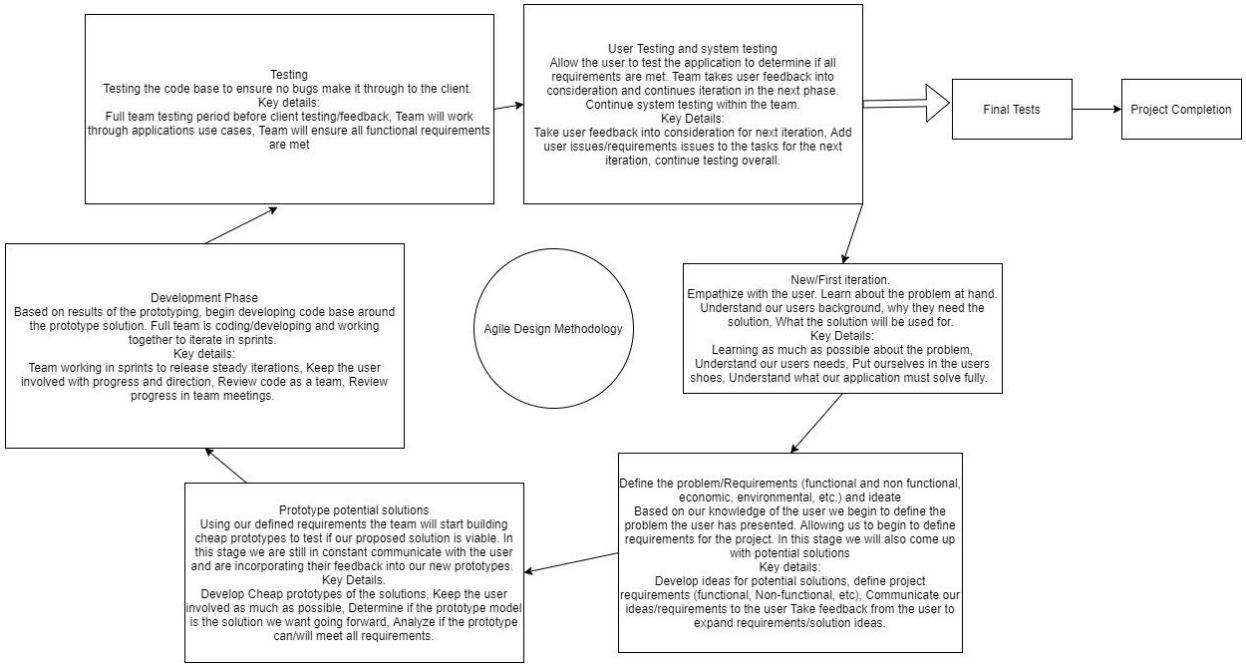
6.2 References

"Welcome to KBase Predictive Biology." *KBase*, 6 Aug. 2020, www.kbase.us/.

"Selenium Automates Browsers. That's It!" *SeleniumHQ Browser Automation*, www.selenium.dev/.

6.3 Appendices

Break down of our Agile development process each two week sprint:



Typical two week sprint methodology