Enabling Repeatable Graph-based Experimentation and Education

Design Document

Team Number: 54
Client: Suresh Kothari
Team Members and Roles:
Austin Gregory: Documentarian
Blake Mulnix: Lead Developer
Kyle Ferguson: Developer
Matthew Schaffer: Developer
Peter Marasco: Scrum Master
Team Email:
sdmay20-54@iastate.edu
Team Website:
sdmay20-54.sdweb.ece.iastate.edu

Revised: Dec 8th 2019

Executive Summary

Development Standards & Practices Used

- Google's Java Style Guide
- IEEE Standards for Software Engineering
- Agile SCRUM Development Process

Summary of Requirements

- Fix visualization issues
    - Styling
    - Layout
    - Interactivity
- Fix integration with Jupyter
    - Functions on all browsers
    - Lends itself to interactivity
- Create a generalized graph schema
    - Allows serialization of Atlas graphs
    - Allows creation of graphs via API

Applicable Courses from Iowa State University Curriculum

- CS 227 - Intro to Object Oriented Program
- CS 228 - Data Structures
- CS 311 - Algorithms
- CS 319 - Web design
- CS 363 - Databases

New Skills/Knowledge acquired that was not taught in courses

- Working on an open source project
- Working with third party visualization libraries

Table of Contents

List of figures/tables/symbols/definitions

1 Introduction

1.1 Acknowledgement

Work on this project was done with assistance from Payas Awadhutkar, who has worked closely with the software before and will serve as a technical guide to the software and codebase. We are working closely with him to familiarize ourselves with the project and the problems we need to solve.

1.2 Problem and Project Statement

General Problem Statement:

The Atlas plugin for Eclipse is capable of generating and visualizing control flow graphs of codebases, which allows a developer to visually debug a program. This is very useful, however, the visualization functionality is proprietary and can only be done locally with Atlas, which makes it difficult for developers to share these graphs amongst themselves. Our client's tool, CHPG, takes an exported graph from Atlas and attempts to replicate Atlas' visualization. Right now CHPG is struggling to render these graphs in a coherent way. The problem this project seeks to solve is to improve CHPG's visualization capabilities so that they are comparable to Atlas', making CHPG a more usable tool for teams in the debugging process. Once this issue is fixed we will work to make CHPG more generalized so it can be used across other domains where graphical representation is useful.

General Solution Approach:

Our approach to solving this problem is to break it down incrementally. We are collecting a list of changes that need to be made as we compare Atlas and CHPG graphs. We are tackling the obvious issues first, and working our way down the list. Assuming there are no issues with the library currently in use we should be able to fine tune the visualization over time to reach a viable state. If we reach this state and still have time we will continue to work on incremental improvements to CHPG's other functions, with the goal of leaving CHPG as a much more viable tool.

1.3 Operational Environment

The end project here is software so we do not need to take physical operational environment into account.

1.4 Requirements

Currently the most important functional requirements for our project center around visualization and integration with Jupyter. Once the issues surrounding these components are resolved, we can move forward and work on other aspects of CHPG. A high level overview of the function requirements is given below:

1. Fix visualization issues
    a. Styling
    b. Layout
    c. Interactivity
2. Fix integration with Jupyter
    a. Functions on all browsers
    b. Lends itself to interactivity
3. Create a generalized graph schema
    a. Allows serialization of Atlas graphs
    b. Allows creation of graphs via API

Our non-functional requirements center around the following categories:

1. Performance: Our application should perform processing intensive actions in a reasonable amount of time
    a. Visualization
    b. Analysis
    c. Input and Output
2. Generalization: The database schema we develop should be generalized to be used across any domain
3. Expandable: The application we develop should lend itself to further expansion by open source developers

1.5 Intended Users and Uses

The intended users of this product are researchers, students, and professionals. We will make a robust and generalized graphing tool that can be used to understand, analyze, and teach systems and processes that can be represented as graphs. Our application will enable them to share graphs and findings easily.

1.6 Assumptions and Limitations

Assumptions:

- Functionality outside of the visualization aspect of the tool is outside our scope until visualization is working as desired.
- The exported Atlas graph that CHPG takes as input will be a valid graph, so we don't need to validate.

Limitations:

- We do not have access to the Atlas visualization source code because it is proprietary.

- We are currently limited by the capabilities of the visualization library that we are using. If this becomes a problem we may need to change libraries.

1.7 Expected End Product and Deliverables

The final product will be an updated version of the visualization algorithm for CHPG and any additional improvements that we are able to make to the software within the time we have to work on this project. Visualization shall mimic Atlas as best as possible, meaning that the generated graphs should be easily readable and understood. These changes will be delivered in increments every 5-6 weeks. Further deliverables will be decided based on the progress of the project and areas that are identified as needing improvement as time goes on.

2. Specifications and Analysis

2.1 Proposed Design

Our approach to completing requirements related to the visualization aspect of the project has been to leverage the capabilities of the visualization library that CHPG currently uses, CytoscapeJS. Alternatively, we could have found other visualization libraries, but we feel that CytoscapeJS is flexible and robust enough to suit our needs.

To fix the issues of integration with Jupyter, we are going with an entirely new approach to the problem. We are going to use a local server to serve up any data that we need to send to Jupyter for visualization. This will allow us to escape issues of cross compatibility that are present in the current implementation.

Expanding upon the graph serialization functionality of the tool is something we will analyze further once the previous to requirements have been resolved. As of now, we know that we want to generalize the graph serialization enough to be able to use graphs exported from Atlas and graphs created via API.

2.2 Design Analysis

As of this point, we have been focusing on the visual aspect of the design. We have

1. Fixed node styling
2. Implemented a tool to interact with the graph (hide/change the color of a node).
3. Improved the overall graph layout to a degree
4. Improved Jupyter integration

Our improvements to the node styling have been a success. They look much cleaner and look much more similar to their Atlas counterparts. Additionally, the interaction tool adds value and can be expanded on as we see the need.

We spent quite a bit of time looking into possible configurations for the CytoscapeJS library, namely within the Dagre and Klay layouts, and have settled on the options that give us the closest alternative to Atlas' proprietary visualization. While still not ideal, we have had the most success with the Dagre layout so far. Our approach has been to use what is available within CytoscapeJS, but going forward we may need to shift that approach to customize the library/layout for our needs.

In addition we also improved integration with Jupyter notebook by using web sockets to deliver files containing the visualization to the browser. This was a big improvement from when they were

stored in a temporary directory and Jupyter failed to load them in. Now they consistently display in the browser. One issue at the moment is that the socket connection fails when run on a Mac, so that will need to be addressed. We anticipate running a small local server to handle serving up data to the Jupyter notebook dynamically. This should solve all cross compatibility issues.

The strengths of our solution so far are that it generates much cleaner, more intuitive graphs than before, provides a form of user interaction, and integrates more easily with Jupyter. The weaknesses of this solution are that it needs to be fixed to work with Mac OS, and that the visualization can still be improved to more closely mimic Atlas than it does currently.

2.3 Development Process

We are using an Agile design process due to having deliverables every 5 to 6 weeks. Our work will be focused on identifying problems with the current visualization tool and then completing these tasks within the 5-6 weeks. Agile is a good fit for this cyclical design process and deliverable time frame.

2.4 Design Plan

One of our primary classes is CHPG.io, which is responsible for the input and output of graphs. This class essentially sends or receives graph information to/from an HTML with formatting for the nodes/edges of a graph.

Another integral class to our software is CHPG.GraphView, which is responsible for graph visualization. This class generates the properties of the graph based on the chosen format so that it can be visualized. CHPG.GraphView is where the vast majority of our alterations have occurred. There are several helper classes that we have not altered, given to us by the client.

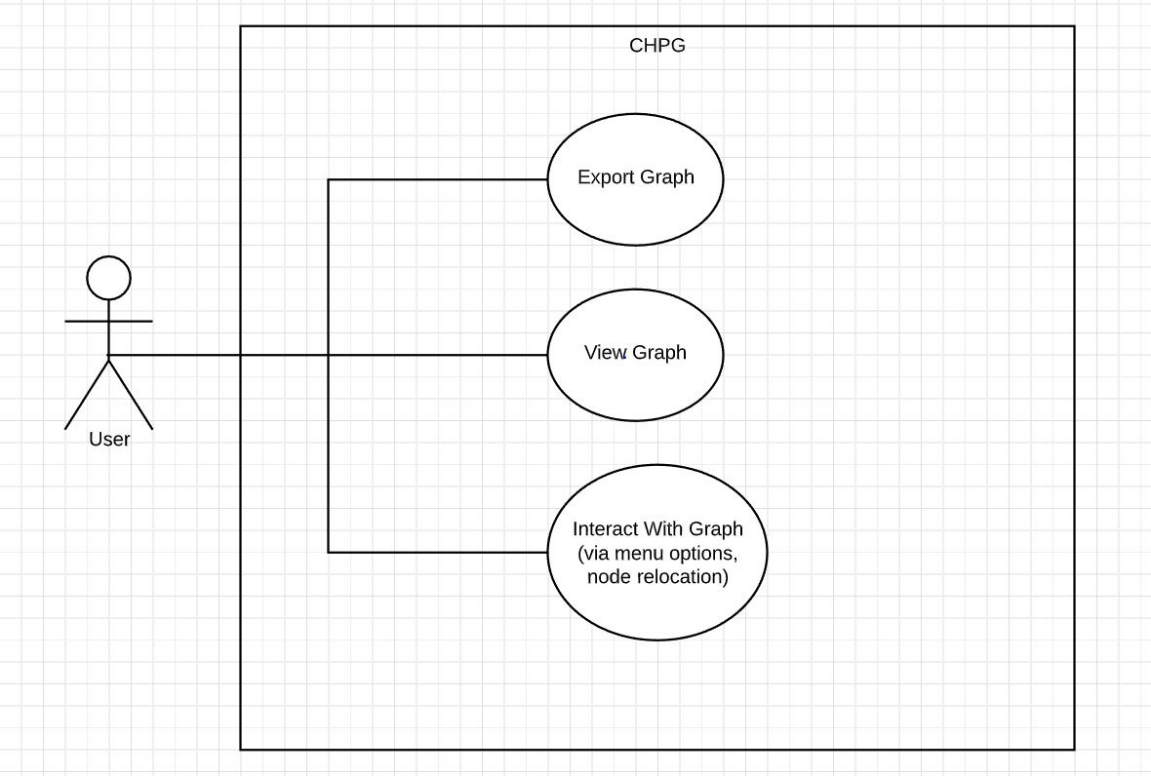

FIGURE 1

3. Statement of Work

3.1 PREVIOUS WORK AND LITERATURE

The most relevant project that we are aware of is Atlas Smart View. Atlas Smart View is an eclipse plug-in which displays a control-flow graph which represents the control-flow of code being executed. Our project is meant to be a replication of Atlas Smart View. The reason this project is useful, is because Atlas Smart View is proprietary, meaning we don't have access to the source code without a license. We want to create an open-source version of this project, so that it can be used and built upon by researchers and educators (and whoever else wishes to use it!)

Beyond the open-source advantage of our project, we also will have a portability advantage, since our graph visualization will work for any code, so long as the exported graph format remains the same.

3.2 TECHNOLOGY CONSIDERATIONS

The current iteration of our project puts us slightly behind the curve of Atlas Smart View. There are a few bugs that we must fix in-order to put our project on the same technical level, and even then, we may have an efficiency and/or memory disadvantage, but that is still up in the air.

One of the key technological tools that we are utilizing is CytoscapeJS, a javascript library for graph visualization. Some of the strengths of this tool is the ease of implementing graph visualization by providing visualizations for nodes and edges, as well as graph-population algorithms. The weakness of this technology is the slight loss in free customization of the graph visualization. We are limited by what CytoscapeJS has to offer, but fortunately what it does offer is roughly exactly what we need. Other graph visualization tools that we have researched yield similar strengths and weaknesses, and CytoscapeJS seemed to be the best option.

Another technological consideration we are currently looking into is how we import/export graphs. Our current implementation will likely struggle with massive graphs (thousands of nodes), as it will be both time and memory inefficient in how the graphs are saved and sent. We are looking into a complete overhaul of how we import/export graphs by switching to a JSON format through a library Jaxon for better efficiency. The strength of this planned change is scalability and memory efficiency, where the weakness is simply the time that will need to be invested in performing this redesign of graph importing/exporting.

3.3 Task Decomposition

Our tasks include a list of features to implement and bugs to fix.

1. Fix visualization issues
    a. Styling
        i. Make graph nodes should resize properly
        ii. Make graph nodes change shape for various different node types
    b. Layout:
        i. Make the layout of nodes mimic Atlas's layout
        ii. Make the edge style and location mimic Atlas's edge styling
    c. Interactivity
        i. Implement graph navigation present in Atlas
        ii. Implement graph analysis tools present in Atlas
2. Fix integration with Jupyter
    a. Ensure functionality is the same on all browsers and operating systems
    b. Ensure Jupyter integration lends itself to interactivity
3. Create a generalized graph schema
    a. Ensure schema allows serialization of Atlas graphs
    b. Essure schema allows creation of graphs via API

3.4 Possible Risks And Risk Management

Knowledge of the graphing tools is the only major risk associated with our project. To overcome this risk, we will use an Agile development process to provide incremental improvements and keep communication channels with our advisors and client open.

We will attempt the avoid risk altogether by making sure all members have a good understanding of how each part of the software works. Delegating tasks to members with a strong understanding of the area they are developing will allow us to avoid risk.

3.5 Project Proposed Milestones and Evaluation Criteria

TODO OVERVIEW

Milestone	Evaluation Criteria
Allow CHPG to run independent from Jupyter for testing purposes	CHPG can successfully create identical visuals with and without Jupyter Notebook
Fix visualization errors provided by client	Client approval of CHPG visuals created from XINU Code
Add visualization interactive features similar to Atlas's	Client approval of CHPG visualization features
Formally update database schema to allow for new features	Documentation of schema updates

Table 1

3.6 Project Tracking Procedures

We will track our progress using Trello, Status Reports, and Weekly Group Meetings. We have started with a list of bug and features, and will track our progress by moving each item to the appropriate list as we work on them. Then, we will record progress through Status Reports, and discuss our progress and future changes through Weekly Group Meetings. Furthermore we will schedule meetings with our client when possible to discuss further improvements to our application.

3.7 Expected Results and Validation

We expect to eliminate all major bugs from the current iteration of our project, as well as add features beyond that of Atlas Smart View. Testing will be conducted by running the graph visualization tool on multiple graph based datasets. The visuals created by the CHPG visualization tool will then be compared to visuals created by Atlas for verification. The operating system Xinu will be used as our test subject. We will compare Atlas's visualization and our visualization for each graph that can be created from Xinu.

4. Project Timeline, Estimated Resources, and Challenges

4.1 Project Timeline

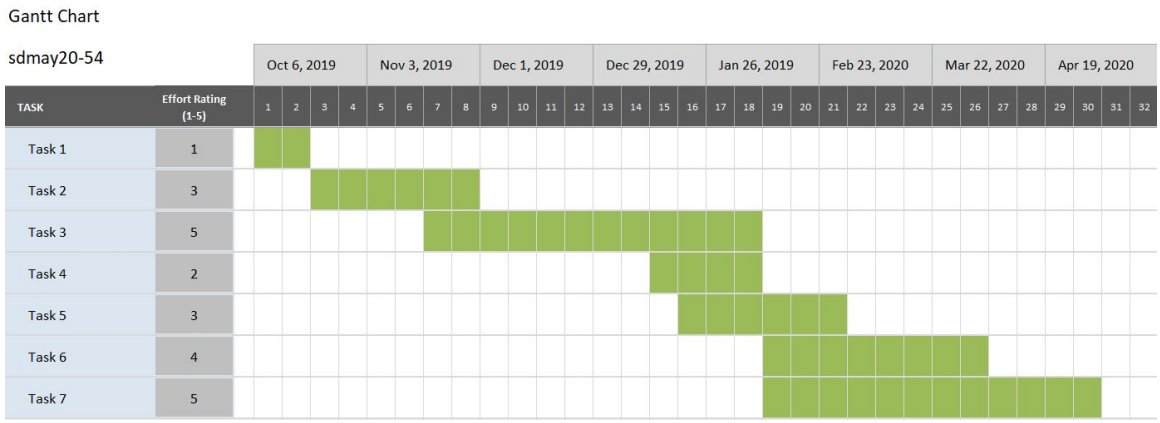

Figure 2

This timeline allows for extra time for high effort task, while overlapping them with low effort task to mitigate discouragement while working on a difficult task. Tasks that overlap also come from different areas in the codebase, in order to prevent developers from stepping on each others' toes and reducing merge conflicts. Two weeks are left at the end of the semester to allow for the project to be wrapped up, and allow for proper documentation and assessment of our work.

In the later stages of the project we allow for one of the most difficult tasks to be completed in a large development window. The other task with a maximum difficult rating is scheduled to be worked on earlier in the year so we won't have to overlap these two tasks. This will allow us to spread out work throughout these two semesters.

4.2 Feasibility Assessment

This project should be easily manageable within the timeframe given the scope of the project, and the scope will expand to fill our timeframe. Our biggest foreseen challenge with this project is the possibility that the visualization library we are using will not be suitable for our needs and we will have to find and learn to use another library. However, thus far CytoscapeJS seems to be working well and should give us the functionality we need to accomplish our goals.

4.3 Personnel Effort Requirements

Task effort estimations are based on a scale from 1 to 5, with 1 being simple and 5 being complex.

Task	Description	Effort Estimation
Make graph nodes should resize properly	Graph nodes should resize based on text and text should not extend out of the node's boundaries	1
Make graph nodes change shape for various different node types	Graph nodes should change shape for various types of nodes (such as conditionals in control flow graphs)	2
Make the layout of nodes mimic Atlas's layout	The layout of graphs should closely mimic the layout of graphs created in Atlas	4
Make the edge style and location mimic Atlas's edge styling	The edge styles of graphs should mimic the edge style of graphs created in Atlas	4
Implement graph navigation present in Atlas	The graph should allow for the same navigation features present in Atlas graphs	5
Implement graph analysis tools present in Atlas	The graph should allow for the same graph analysis features present in Atlas graphs	5
Ensure functionality is the same on all browsers and operating systems	The graph visualization should function properly on all browsers and operating systems	3
Ensure schema allows serialization of Atlas graphs	The database schema should allow for the serialization of graphs created by Atlas	3

Ensure schema allows for creation of graphs via API	The database schema should allow for the serialization of graphs created via an API	4

Table 2

4.4 Other Resource Requirements

No material requirements necessary aside from our personal computers.

4.5 Financial Requirements

No financial resources required.

5. Testing and Implementation

5.1 Interface Specifications

The main interface of our system will be the interface of the graph schema implementation. We will cover each point of this interface by using unit tests.

5.2 Hardware and software

We will be using the JUnitlibrary to test our application. JUnit enables the creation of automated unit tests in Java.

5.3 Functional Testing

We will conduct high-level, manual integration testing to ensure our applications functions properly, integrates with Jupyter, and runs on all systems.

We will write unit tests for each functional unit of our implementation of the graph database schema. Our testing plan depends highly on our implementation of graph database and thus we cannot lay out a detailed set of tests until that we begin work on that implementation.

We will the unit tests we create as a suite for regression testing. This will ensure all incremental changes to the application do not break previously added features.

5.4 Non-Functional Testing

We will use manual verification to ensure that our application performs visualization, analysis, and serialization quickly.

5.5 Process

We will use a combination of manual verification of the software's functionality and automated testing of each functional unit of the software.

5.6 Results

We will consult with our client and advisors on the results of our testing to ensure that they are satisfactory. We will use their analysis to improve our test cases and process if they are not satisfactory.

6. Closing Material

6.1 Conclusion

We have concurrently downloaded the software needed for the project and did some exploring of the application. This allowed us to come up with some features that we want to implement for the project. One of the big things is making the control flow graph look nicer. We hope to implement that by making the graph more intuitive, not having the lines cross each other, and having the blocks change shape based off the statement inside it.

6.2 References

This is not applicable right now.

6.3 Appendices

This is not applicable right now.