Enabling Repeatable Graph-Based Experimentation and Education

Senior Design Spring 2020

Team 54:

Austin Gregory, Blake Mulnix,
Kyle Ferguson, Matthew Schaffer,
and Peter Marasco

Client: Knowledge Centric Software Lab

Advisor: Dr. Suresh Kothari

The Problem: Overview

- Professors want to enable repeatable experimentation to derive knowledge from large graph-based datasets
- Ensoft’s Atlas provides graphing functionality; however it is not ideal:
  - Atlas requires a license to use it and while licenses can be obtained by students for no cost, it can take some time to acquire these licenses
  - Using Atlas requires that you install Eclipse and the Atlas Plugin, a process that can be a bit difficult for those not experienced with it
- Our client wanted an open source implementation of graph visualization that can be used within a Jupyter Notebook

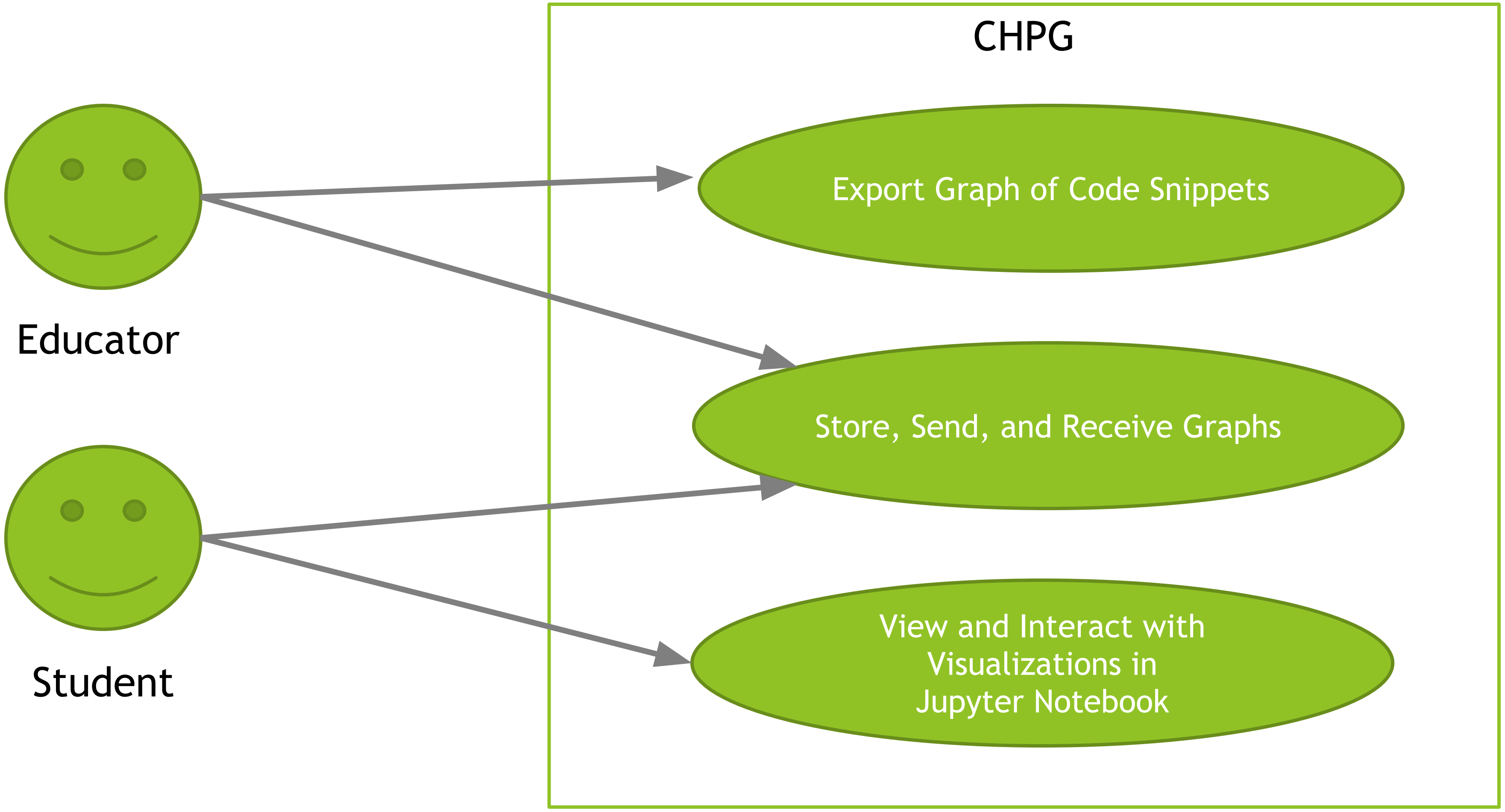
The Problem: Existing Prototype

- The Knowledge Centric Software lab (KCSL) had developed an open source prototype that enabled some of desired graphing functionalities
- The project was capable of producing visualizations of software graphs within a Jupyter Notebook
- However, the prototype was very buggy and did not provide the features that our client desired

The Solution:
Improve the Prototype

- Our team has improved upon the prototype to provide a product that is capable of visualization and provides the features our client desired
- Namely, we have achieved the following
  - Eliminated bugs within the existing prototype
  - Refactored and documented the existing code to make the project more maintainable and expandable for teams in the future
  - Added the desired visualization features that the prototype was initially lacking

Project Plan: Conceptual Sketch


CHPG
Educator
Student
Export Graph of Code Snippets
Store, Send, and Receive Graphs
View and Interact with Visualizations in Jupyter Notebook

Project Plan:
Functional Requirements

- Ability to visualize Control Flow Graphs
  - Easy to read and visually pleasing styling
  - Layout that closely resembles Atlas’s graph layout
  - Interactivity
    - Nodes can be moved
    - Node colors can be changed
- Integration with Jupyter
  - Functions on all browsers
  - Functions on all operating systems

Project Plan:
Non-functional Requirements

- Performance: Perform processing intensive actions quickly
  - Interactivity
  - Input and Output
- Simplicity of Installation: The application should be easy to install and set up
- Ease of Use: Easy for those new to software development
- Expandable: App should lend itself to further expansion by open source developers

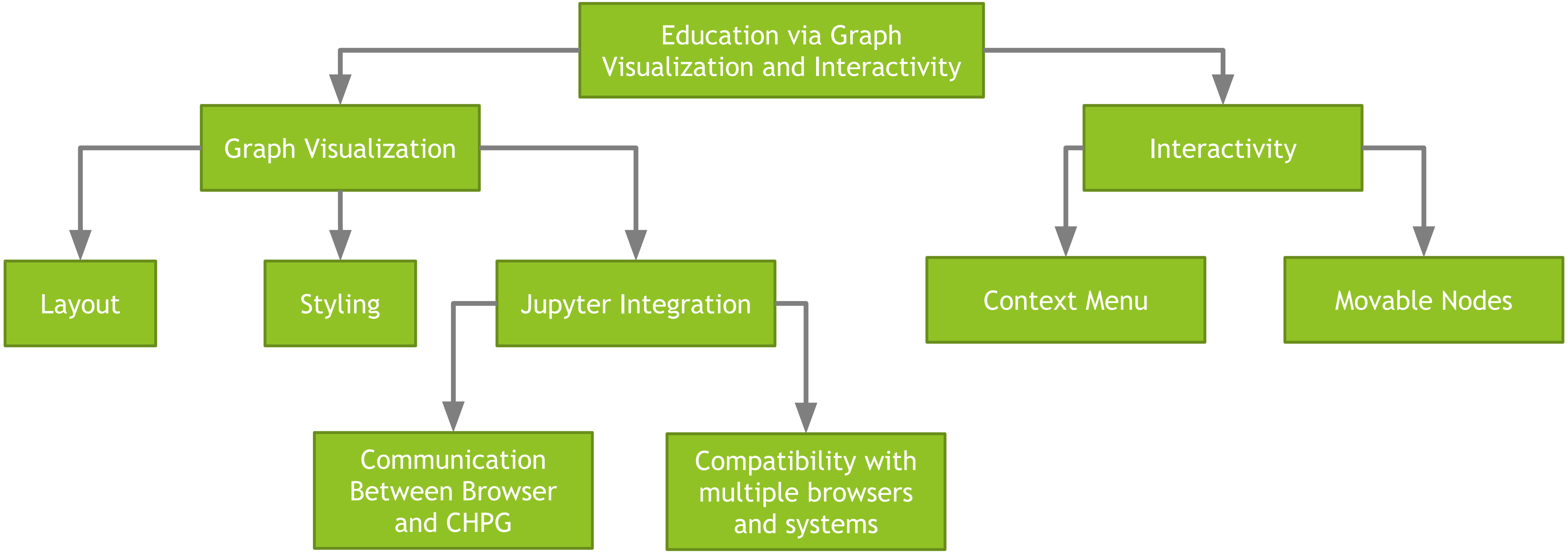
Project Plan:
Technical Constraints

- Atlas graph data: Required to work with graph data in Atlas's format
- Jupyter platform: Must design solution within the framework of this system
- Java and JavaScript: Limited to using libraries written for these languages
- Portability: CHPG graph files need to in a serialized and portable format
- Open Source: Inability to use proprietary software due to lack of license

Project Plan: Risk

- The main risk we faced in the project was our inexperience in the field of graph visualization.
- This is a large field within Computer Science and there has been a huge amount of research done on this topic
- We managed this risk using the following strategies:
  - Communication with Advisors/Client
    - By frequently communicating our status to our advisors/client, we could get feedback on our approach before proceeding incorrectly
  - Agile Development Process
    - By employing an agile approach to software development, we could strive for incremental improvements for all team members so that our knowledge of the material can expand gradually and uniformly

System Design: Functional Decomposition



Education via Graph Visualization and Interactivity

Graph Visualization

Interactivity

Layout

Styling

Jupyter Integration

Context Menu

Movable Nodes

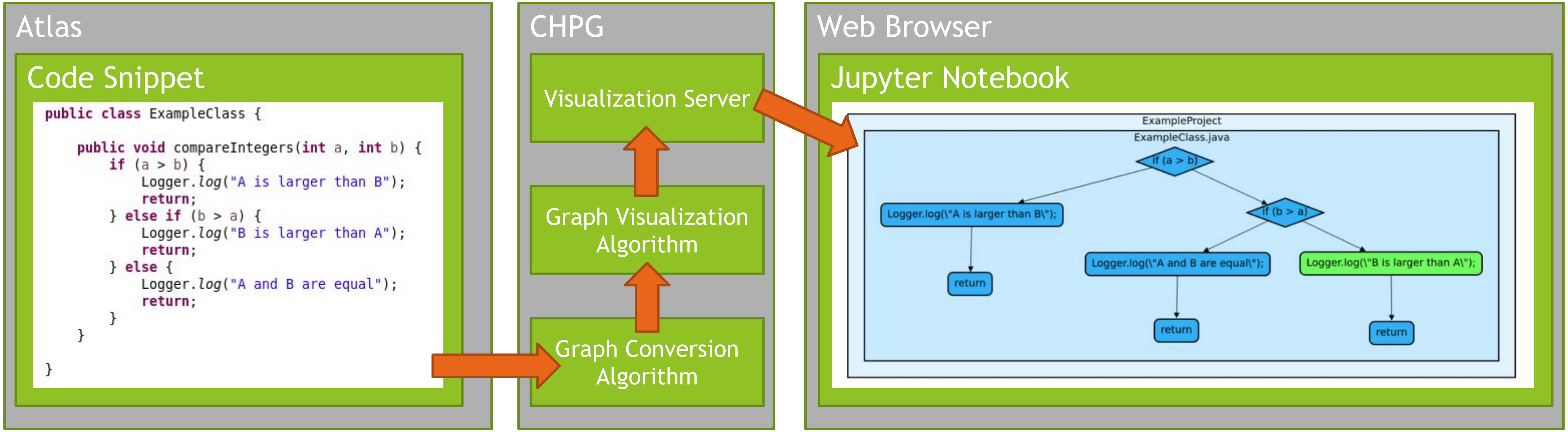
Communication Between Browser and CHPG

Compatibility with multiple browsers and systems

System Design: Technologies Used

- Software
  - Eclipse IDE
  - Ensoft’s Atlas Shell
  - Jupyter Notebook
  - IJava Jupyter Kernel
  - CytoscapeJS Library
- Programming Languages
  - Java
  - JavaScript

System Design



Atlas

Code Snippet

```java
public class ExampleClass {

    public void compareIntegers(int a, int b) {
        if (a > b) {
            Logger.log("A is larger than B");
            return;
        } else if (b > a) {
            Logger.log("B is larger than A");
            return;
        } else {
            Logger.log("A and B are equal");
            return;
        }
    }

}
```

CHPG

Visualization Server

Graph Visualization Algorithm

Graph Conversion Algorithm

Web Browser

Jupyter Notebook

ExampleProject
ExampleClass.java
if (a > b)
Logger.log(\"A is larger than B\");
return
if (b > a)
Logger.log(\"A and B are equal\");
return
Logger.log(\"B is larger than A\");
return

System Design: Details

- Creation of Atlas Control Flow Graph
  - The Atlas Eclipse plugin is used to create a CFG of snippet of code
  - This CFG is then stored to a variable in memory
- Conversion and Exportation CFG
  - The CHPG Conversion Algorithm is used to convert the Atlas CFG into a CHPG graph
  - The CHPG Export Algorithm is used to serialize the CFG and export it to a file
- Visualization and Interaction with Graph
  - Jupyter Notebook then uses the IJava kernel to execute the CHPG commands to load the CHPG graph file into memory
  - The CHPG Visualization Algorithm then creates and displays the graph visualization in Jupyter using the CytoscapeJS library

System Design:
Test Plan

- Due to the structure of this project, unit testing was not an easy task
- Our advisors agreed that manual testing would suffice
- We manually tested that the output of Atlas's graph visualization matched with our visualization for the functions within the Xinu operating system
- We manually tested that our project worked across the following:
  - Operating Systems:
    - Windows 10
    - Mac OS
    - Linux
  - Browsers:
    - Chrome
    - Firefox
    - Safari

Demonstration

Development Process: Decision to use Agile

- Our group developed our solution using an Agile development lifecycle
- Our client suggested that we use this type of process and we agreed that it suited the type of project we were doing
- We elected to use flexible length sprints of approximately 4 weeks in duration
- In addition we held weekly discussions with our client over near-term goals, priorities, and progress

Development Process: Overview of Our Process

- The first sprint was used to:
  - Discuss the problem with our client and establish
    - Use Cases
    - Requirements
    - Priorities
  - Familiarize ourselves with the existing prototype
  - Research potential third-party libraries we could use
  - Set up our development environments
- All following sprints involved
  - Discussing progress with client
  - Adjusting priorities
  - Assigning tasks to team members
  - Executing tasks and implementing features

Development Process: Benefits of Using Agile

- We avoided the issue of spending all of the fall semester planning and leaving ourselves only the spring semester to implement our solution
- We were able to focus on bite-sized chunks of functionality within each sprint
- We were able to implement these chunks of functionality and then demonstrate our progress to our client and get rapid feedback
- We followed the philosophy of “failing fast”, allowing us to rapidly determine whether a potential technical solution was a good solution

Conclusion:
Member Contributions

- Kyle Ferguson - Developer
  - Fixed node styling errors
  - Worked on graph node type interpretation solutions
  - Made containers for node groups' respective functions and project
- Austin Gregory - Developer/Documentarian
  - Fixed graph layout (node and edge styling)
  - Created weekly status reports
- Peter Marasco - Developer/Scrum Master
  - Implemented the graph visualization context menu
  - Gathered information from graph data to be used in containers
  - Maintained the Agile Scrum board of tasks and issues

- Blake Mulnix - Lead Developer
  - Refactored graph visualization
  - Implemented solution to Jupyter integration issue
  - Managed master branch
- Matthew Schaffer - Developer
  - Fixed graph layout
    (node and edge styling)

Conclusion:
Current Project Status

- Visualization
  - We have solved the issues related to graph visualization
  - We refactored the visualization code to more expandable and robust
  - We have added containers which displays the class and function the graph visualizes
- Graph Interactivity
  - Added context menu to graph to alter aspects of the graph visualization
  - Added moving nodes/edges in graph visualization
- Jupyter Integration
  - We have solved Jupyter integration on most systems
  - We have solved errors displaying CHPG files through Jupyter

Engineering Standards and Practices Used

- Google’s Java Style Guide
  - We wrote and documented our code using these standards for Java development
- Agile SCRUM
  - We used this development lifecycle to execute our project in an iterative manner and get rapid feedback

Questions

Thank You!