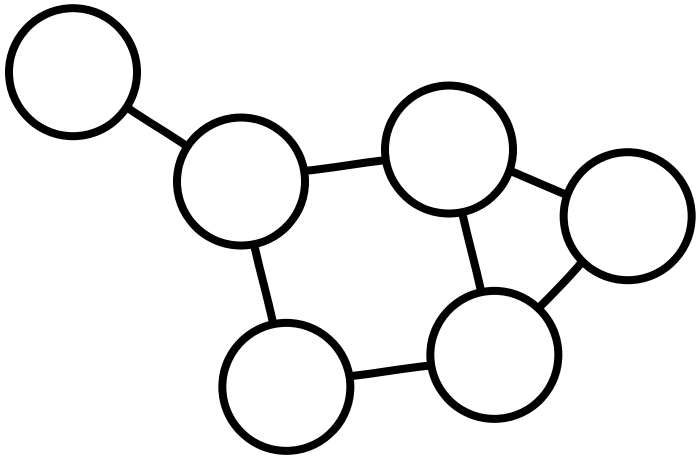
Capstone Client Requirements

RapidGraph

An open-source, web-based visual graph analyzer designed for rapid graph creation and analyzation, and rapid, modular development



By team randm (Michael Anderson & Rob McGuire-Dale)

Sponsored by Christine Wallace

CS 461 / 462 / 463: CS Capstone Experience with Dr. Bailey
8 November 2009

Team Details

Name: randm

Number: 21

Members:

Michael Anderson
andermic@engr.orst.edu
503 780 8715

Rob McGuire-Dale
mcguirer@engr.orst.edu
541 250 0762

Sponsor

Christine Wallace
wallacch@eecs.oregonstate.edu

Problem Introduction

Graphs are an important mathematical concept and are integral to a wide variety of computer science applications. Mathematics and computer science students clearly need easily-accessible and user-friendly tools to learn about, generate, and analyze different types of graphs in a number of different ways to assist in their graph theory education. Current graph analyzers tend to be non-portable, a hassle to install, closed-source, and difficult to use.

Project Description

The mathematical graph analyzers that exist today generally require one to install and/or use unintuitive, complicated mathematics software. We aim to create a web-based graph analyzer that is easily accessible to anyone with a modern web-browser. It should be easy to intuitively and rapidly create graphs with a visually-appealing user interface, and efficiently analyze them with an modular, plug-in-based, extendable back-end.

Upon completion of this project, we will have a functioning graph analyzer running as a web app on a publically-accessible web server, and a project page that will provide fully-comment-documented source code and documentation including how to install install the application on one's own web server, and how to extend its functionality through a reference guide, examples, and a tutorial.

Requirements

Functional Requirements:

- Enable users to create a graph quickly, easily, and visual environment.
- Allow users to zoom, scale, rotate, and otherwise manipulate graph images.
- Enable users to customize the look and feel of graph elements, to turn obscure mathematical models into practical images.
- Give users the ability to save and load a graph file from his or her local disk or the webserver hosting the project site.
- Have support for standard, weighted, directed, and mixed graphs.
- Implement algorithms that determine the type of a graph (simple, planar, bipartite, complete, clique, n-cubes, different flavors of trees, etc.)
- Implement algorithms that find special paths and cycles in a graph (Hamiltonian paths, Euler circuits, Dijkstra's algorithm, etc.)
- As a proof-of-concept, build add-ons (plug-ins) that use our system to model specific, graph-based, real-world applications, such as the "knight's tour" problem, peer-to-peer networking, the "travelling salesman" problem, the "instant insanity" puzzle, map-coloring, and drawing of chemical molecules.

Technical Requirements:

- Web-based to allow easy, cross-platform access with no installation necessary.
- Open-source to maximize educational value.
- Written primarily in JavaScript to maximize portability.
- Code should be well documented, modularized/object-oriented, with a clean architecture for readability and expandability.

Versions

Milestones:

- Project site implemented: Main trac, wiki, and get hosting set up at the OSL. This will be our main documentation/project management hub for the project.
- Framework in place: Main Django application implemented. This will be the framework that the entire application will be built upon.
- Core application implemented: Main application implemented as a Django app. This will be the main front-end of our application, and be a hub for every other piece of functionality we add on.
- Plug-in system created: System for aggregating Django sub apps as plug-ins implemented. Each plugin extends the functionality of the UI to analyze the graph.
- Base analyzer plug-ins created: Base analyzer functionality implemented via separate plugins.
- Application deployed: Application running and accessible on a server.

Design

Our system will have 3 layers:

- A front-end web interface implemented in JavaScript/jQuery, HTML, and CSS.
- A Django web framework that glues the frontend and backend together. We will first build a core Django application containing the functionality that will be common to our entire web interface, and then extend it with plug-ins that offer more specific functionalities (such as the proofs of concept).
- A backend implemented mostly in Python, but partially in C/C++ to speed up execution of more time-expensive graph algorithms.

Specific Tasks & Timetable

Please refer to the attached Gantt chart, which breaks down each milestone into tasks. Please also note that each task time includes a time buffer just in case certain tasks take longer than expected.

Risk Assessment

One possibility is that a merge of the frontend and the backend might be difficult since each of them are will be coded by two different people working in parallel. We will watch out for this problem by zipping up these two parts piece-by-piece, and integrationally test them as we both move forward. In other words, as new functionality is added to either the front or backend, we test to ensure that they may be used together.

Another risk is the fact that tasks may take longer than expected. We have anticipated this, and built in time buffers into each task.

Testing

We will use readily available Python and C/C++ unit test frameworks such as the unittest library for Python and UnitTest++ for C/C++ for the back end. Each functional unit will be unit tested, and integration tested.

Testing the functionality of the front end will be slightly more difficult as automated unit tests will not work for user interfaces. Techniques such as the PARE usability testing methods will need to be employed.

Member Roles

The workload naturally divides itself into three parts: the web front end, the backend where graph algorithm computation takes place, and the proof(s) of concept. Rob will be responsible for the front end, and Mike will code the backend. Mike and Rob will work on the proof(s) of concept together.

Integration Plan

Since the frontend will be an interface for the computations done in the backend, the backend will be implemented as an API full of methods that the frontend can call. The proofs of concept will either be split into frontend and backend parts that Rob and Mike may both work on in parallel, or Rob can code one proof of concept while Mike codes another.

Dataflow Sequence Diagram

Please see attached dataflow diagram.

User Interface Requirements

The main user interface will need to allow the user to perform the following tasks:

- Add/delete vertecies to/from the graph
- Add/delete edges to/from connect the verticies
- Add/delete weights to/from the edges
- Add/delete names/labels to/from the edges/vertecies
- Zoom/rotate the graph
- Create a new graph
- Save/load a graph from the user's machine/profile
- Re-position vertecies
- Color the edges/vertecies
- Select and run installed plugins

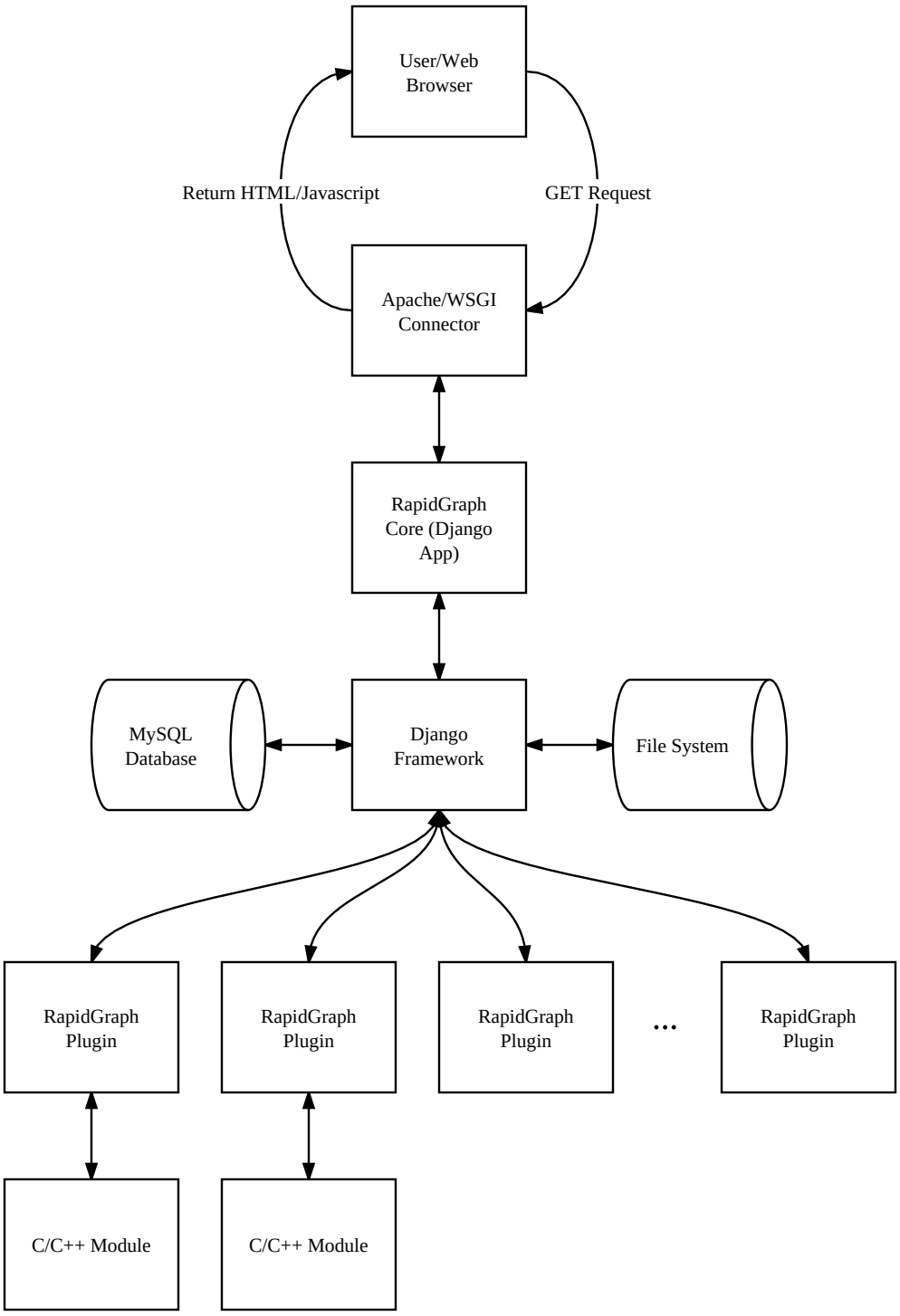
Signatures

Michael Anderson _____

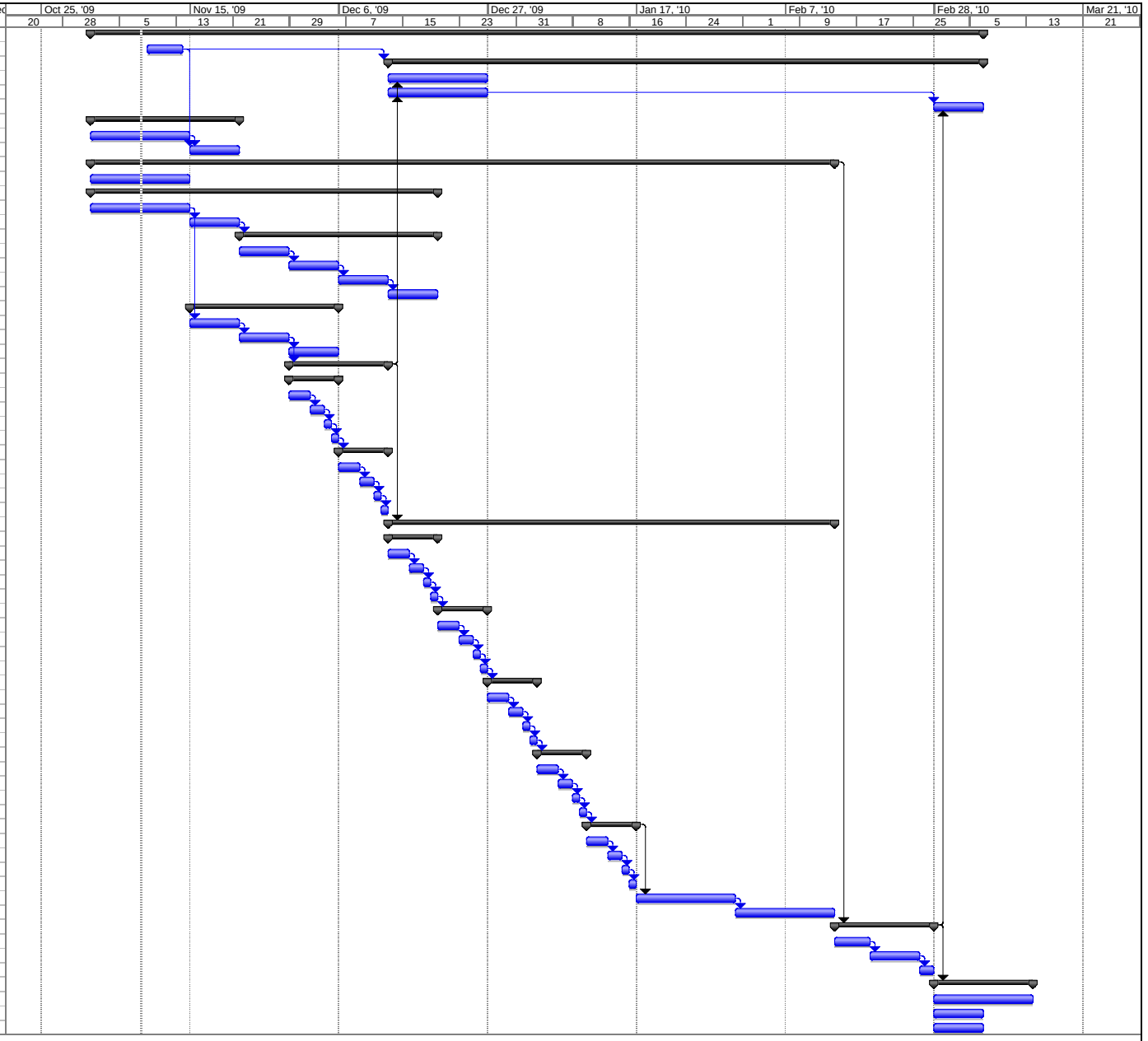
Rob McGuire-Dale _____

Christine Wallace _____

RapidGraph Dataflow Diagram



ID	Task Name	Duration	Start	Finish	Predec
1	**Project Administration**	**126 days**	**Sun 11/1/09**	**Sat 3/6/10**	
2	Procure Trac instance	5 days	Mon 11/9/09	Fri 11/13/09	
3	**Documentation**	**84 days**	**Sun 12/13/09**	**Sat 3/6/10**	**2**
4	Write user guide	14 days	Sun 12/13/09	Sat 12/26/09	24
5	Write plugin tutorial	14 days	Sun 12/13/09	Sat 12/26/09	24
6	Write installation instructions	7 days	Sun 2/28/10	Sat 3/6/10	5,63
7	**Project Management**	**21 days**	**Sun 11/1/09**	**Sat 11/21/09**	
8	Gather requirements	14 days	Sun 11/1/09	Sat 11/14/09	
9	Create initial tickets	7 days	Sun 11/15/09	Sat 11/21/09	8,2
10	**Application**	**105 days**	**Sun 11/1/09**	**Sat 2/13/10**	
11	Overall design	14 days	Sun 11/1/09	Sat 11/14/09	
12	**Framework**	**49 days**	**Sun 11/1/09**	**Sat 12/19/09**	
13	Design framework	14 days	Sun 11/1/09	Sat 11/14/09	
14	Create main Django app	7 days	Sun 11/15/09	Sat 11/21/09	13
15	**Core application**	**28 days**	**Sun 11/22/09**	**Sat 12/19/09**	**14**
16	Research Javascript UI options	7 days	Sun 11/22/09	Sat 11/28/09	
17	Design core UI	7 days	Sun 11/29/09	Sat 12/5/09	16
18	Create base UI functionality	7 days	Sun 12/6/09	Sat 12/12/09	17
19	Preliminary usability testing	7 days	Sun 12/13/09	Sat 12/19/09	18
20	**Plugin System**	**21 days**	**Sun 11/15/09**	**Sat 12/5/09**	
21	Design plugin system	7 days	Sun 11/15/09	Sat 11/21/09	13
22	Create plugin system	7 days	Sun 11/22/09	Sat 11/28/09	21
23	Unit test plugin system	7 days	Sun 11/29/09	Sat 12/5/09	22
24	**Base Plugins**	**14 days**	**Sun 11/29/09**	**Sat 12/12/09**	**22**
25	**Graph type**	**7 days**	**Sun 11/29/09**	**Sat 12/5/09**	
26	Design	3 days	Sun 11/29/09	Tue 12/1/09	
27	Code	2 days	Wed 12/2/09	Thu 12/3/09	26
28	Unit test	1 day	Fri 12/4/09	Fri 12/4/09	27
29	Integration test	1 day	Sat 12/5/09	Sat 12/5/09	28
30	**Paths and cycles**	**7 days**	**Sun 12/6/09**	**Sat 12/12/09**	**29**
31	Design	3 days	Sun 12/6/09	Tue 12/8/09	
32	Code	2 days	Wed 12/9/09	Thu 12/10/09	31
33	Unit test	1 day	Fri 12/11/09	Fri 12/11/09	32
34	Integration test	1 day	Sat 12/12/09	Sat 12/12/09	33
35	**Additional plugins**	**63 days**	**Sun 12/13/09**	**Sat 2/13/10**	**24**
36	**Map coloring problem**	**7 days**	**Sun 12/13/09**	**Sat 12/19/09**	
37	Design	3 days	Sun 12/13/09	Tue 12/15/09	
38	Code	2 days	Wed 12/16/09	Thu 12/17/09	37
39	Unit test	1 day	Fri 12/18/09	Fri 12/18/09	38
40	Integration test	1 day	Sat 12/19/09	Sat 12/19/09	39
41	**Knight's Tour problem**	**7 days**	**Sun 12/20/09**	**Sat 12/26/09**	**40**
42	Design	3 days	Sun 12/20/09	Tue 12/22/09	
43	Code	2 days	Wed 12/23/09	Thu 12/24/09	42
44	Unit test	1 day	Fri 12/25/09	Fri 12/25/09	43
45	Integration test	1 day	Sat 12/26/09	Sat 12/26/09	44
46	**Traveling Salesman problem**	**7 days**	**Sun 12/27/09**	**Sat 1/2/10**	**45**
47	Design	3 days	Sun 12/27/09	Tue 12/29/09	
48	Code	2 days	Wed 12/30/09	Thu 12/31/09	47
49	Unit test	1 day	Fri 1/1/10	Fri 1/1/10	48
50	Integration test	1 day	Sat 1/2/10	Sat 1/2/10	49
51	**Instant Insanity puzzle**	**7 days**	**Sun 1/3/10**	**Sat 1/9/10**	**50**
52	Design	3 days	Sun 1/3/10	Tue 1/5/10	
53	Code	2 days	Wed 1/6/10	Thu 1/7/10	52
54	Unit test	1 day	Fri 1/8/10	Fri 1/8/10	53
55	Integration test	1 day	Sat 1/9/10	Sat 1/9/10	54
56	**Chemical Molecules**	**7 days**	**Sun 1/10/10**	**Sat 1/16/10**	**55**
57	Design	3 days	Sun 1/10/10	Tue 1/12/10	
58	Code	2 days	Wed 1/13/10	Thu 1/14/10	57
59	Unit test	1 day	Fri 1/15/10	Fri 1/15/10	58
60	Integration test	1 day	Sat 1/16/10	Sat 1/16/10	59
61	Final integration testing	14 days	Sun 1/17/10	Sat 1/30/10	56
62	Final usability testing	14 days	Sun 1/31/10	Sat 2/13/10	61
63	**Application Deployment**	**14 days**	**Sun 2/14/10**	**Sat 2/27/10**	**10**
64	Procure a server slice at the OSL	5 days	Sun 2/14/10	Thu 2/18/10	
65	Set up server	7 days	Fri 2/19/10	Thu 2/25/10	64
66	Get application running on server slice	2 days	Fri 2/26/10	Sat 2/27/10	65
67	**Presentation**	**14 days**	**Sun 2/28/10**	**Sat 3/13/10**	**63**
68	Create project book	14 days	Sun 2/28/10	Sat 3/13/10	
69	Create posterboard	7 days	Sun 2/28/10	Sat 3/6/10	
70	Pocure a large TV	7 days	Sun 2/28/10	Sat 3/6/10	



Project: ganttChart
Date: Sun 11/8/09

Task Progress Summary External Tasks Deadline
Split Milestone Project Summary External Milestone