Feasibility Study

CS 5150, Spring '18
February 13th, 2018

Development Team

Saqif Badruddin <ssb229@cornell.edu>
Joseph Chuang <jcc436@cornell.edu>
Weiyou Dai <wd248@cornell.edu>
Jianhua Fan <jf773@cornell.edu>
Daniel Jordan Hirsch <djh329@cornell.edu>
Athena Danlu Huang <dh527@cornell.edu>
Tyler Yeung Ishikawa <tyi3@cornell.edu>
Jacob Ethan Rauch <jer322@cornell.edu>

Student Contact

Daniel Hirsch <djh329@cornell.edu>

Client

Kyle Harms <kyle.harms@cornell.edu>

I. Statement of Task

Our understanding of this project is to create a new feature set for the Atom text editor that enables users to create projects with individualized packages and formatting. This improvement would enable professors in charge of large classes, such as the client, Kyle Harms (lecturer of CS 2300), who is teaching a class with the Atom text editor, to enable or disable specific packages and styling preferences on a per-project basis for the class. These project-specific settings would allow the professor to control the learning environments of his students from the compilers to the font size.

The core of the project consists of customizing Atom base code to read configuration files placed in project directories. This will enable a user to put a config file in the original project source code and enable the project to have the desired features immediately. Other planned extensions to this project include a git package that allow students to push their work directly to their instructor from Atom, as well as configuration “profiles” that can be toggled between.

II. Benefits of Project

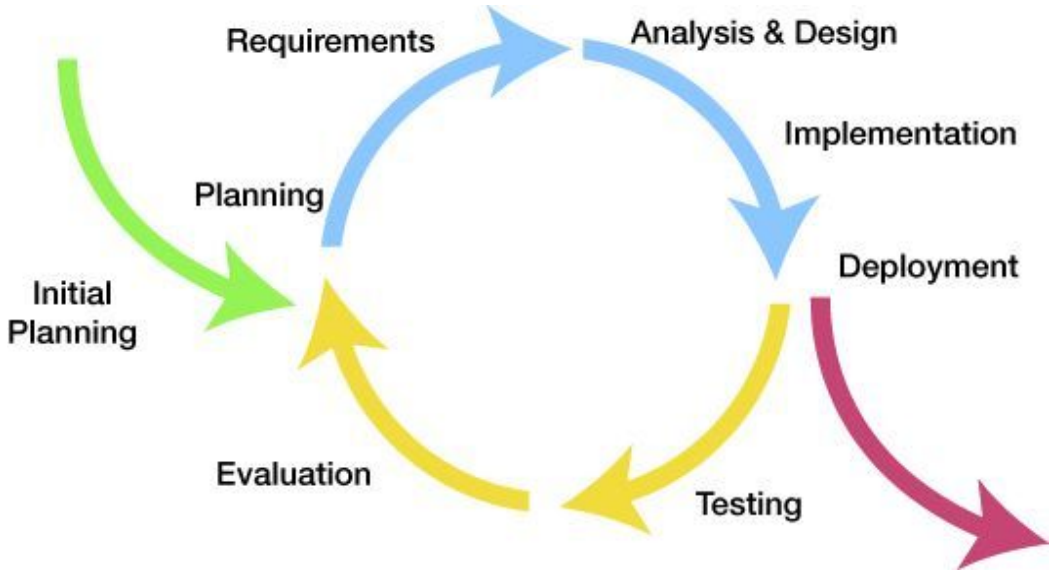
In addition to aiding the professors and students of CS 2300, this project has many benefits for all users of the Atom text editor:

1. One of the main goals of this project is to allow professors to micromanage every aspect of their student’s development environment. This is beneficial for the academic community because it allows professors to help students learn without worrying that they are using packages and features that would inhibit the goals of an assignment via another compiler, linter, or other additive feature.
    - A specific example of this could be teaching beginning programmers how to code the right way. The professor could start out with configuring a project to use “block programming” in order to teach programming fundamentals and concepts. As the course progresses, the professor disables more and more features until eventually the students are able to code the entire assignment themselves.
2. This project also can largely benefit the Atom community as a whole. With project-specific packages and settings, programmers will be able to create their ideal environment from the beginning. Being able to set different font sizes and other settings on a per-project basis should allow developers to code with much greater ease. Moreover, projects with different languages and stacks will automatically be set up the right way, so developers can get right to work instead of fooling around with their IDE for an extended period of time.

III. Process

Given the short-term nature of this project, the team will adopt a lightweight, iterative style of development, where the team will work on building core features first, then slowly add

incremental features as the development progress. This will enable the client to receive all of the major feature requirements, as well as additional bonus features if time permits. An iterative design process will allow us to first achieve the core features the team promised, then the secondary features in the next iteration, and finally the long shot goals in the final iteration.



Source: https://curtisbacon.wordpress.com/2014/10/12/design-iteration/

IV. Business Considerations

As Cornell students, the team owns the copyright to the software that the team creates for this project. However given that the majority of this project will be pushed to GitHub for the Atom text editor project, all rights will be distributed to the public community and the group will not require any further licensing upon distribution.

V. Scope

Immediate Features – Those that the team can guarantee

- Project/Folder-Specific Settings: Enable users to specify *settings* that only apply when editing files in a specific directory by adding them to a pre-defined .cson file in the directory. These settings might include font size, tab length, whether to show whitespace indicators, etc.
- Project/Folder-Specific Packages: Enable users to specify *extensions* that should be disabled when editing files in a specific directory by adding the package names to a pre-defined .cson file in the directory.

Additional Features – Those that the team is not guaranteeing as part of the project, but the team believes may be feasible before handover.

- Git Package for easy code submission
- Landing page with explanations of how to use the package

Long-shot Features – Those that the team believe would be good enhancements, but very well may not end up in the final product due to time constraints.

- Profiles: Ability to manually switch between “profiles” with different defined settings (e.g. Office Profile might have font size of 14pt, but Classroom Profile might have font size of 24pt)

Dependencies

- npm: Used for developing Atom packages
- apm: Used for managing Atom packages

Security

- No major security considerations must be taken into account, since this is a self-contained program built for the Atom text editor

VI. Technical Feasibility

A. Immediate Features

In order to complete this system, the team will need the following functionalities:

1. The ability to read specific config files from a directory in the current project window

This can be accomplished using the Atom text editor’s built-in global `atom` object. That is, in the file `init.coffee`, the team can add a function that looks in `atom.project.getDirectories()` and parses through the directories to search for a specific file extension, `.cson`. The team can do this by manually parsing the file path as a string. The team can then read these files using Atom’s built-in asynchronous file read method and change the settings accordingly.

2. The ability to parse a string as a config file, assuming these files are valid CSON

This can be accomplished using the CSON parse function available through the npm installable library CSON. This will make npm a dependency of this project.

3. The ability to write to the Atom configuration from the scripts

Using the `atom.config.set` method that is global in the Atom coding environment, the team can actually take advantage of the existing APIs to easily configure setting and save files, updating the Atom windows appropriately.

4. Listen to file changes to rerun script

Using the `atom.workspace` observers, the team can listen for file changes and rerun the script to find the most appropriate config file to read from and update the environment settings. For

example, the team can use the `atom.workspace.onDidStopChangingActivePaneItem` event subscription method (https://atom.io/docs/api/v1.23.3/Dock#instance-onDidStopChangingActivePaneItem) to assign a callback method that runs when the user changes tabs to a different file. The event handler `atom.window.onfocus` can also be used to run commands when the user changes windows.

5. Enable and disable packages in Atom

Just as the team can change style above, the team can do this with packages by editing

```
"*":
  core:
    disabledPackages: [
      "my-package"
    ]
```

Using the `atom.config.set`, we can disable any packages we do not want activated in the current context.
There are also methods called `enablePackage()/disablePackage()` that the team can use for this. For further details, see:
https://atom.io/docs/api/v1.23.3/PackageManager#instance-disablePackage

6. Install this code on a user's computer.

The team will be creating an Atom integration that will contain all of the necessary code to enable this functionality. A bash script or other installation method will be provided.

These accomplish the main functionality of this system. As for elements that are considered extra to this project:

B. Additional Features
Since the scope of this project is so heavily impacted by time constraints, the team have certain technical accomplishments the team would ideally aim to achieve. These may or may not be feasible, but if time allows for it, the team may change this project trajectory to accommodate it. The components are as follows:

1. Git Submission

This would involve an additional panel to the Atom editor that would allow users to submit their work directly to GitHub. This could be done by creating a package that has a submit button for users. This submit button will checkout and commit to a group branch that students are enrolled in (this group feature already exists, and is used by the Cornell University Department of

Computer Science currently) and pushes their code for them. This is done using a bash script that will be shipped as part of the package.

2. Landing Page

Many popular application extensions are accompanied by a landing page with information, screenshots, and documentation of usage. For this project, a site like this could be a valuable resource for others to discover and learn more about this package's usage and features.

3. Profiles

This feature consists of allowing a user to create configurations that are local to their own system and can be manually toggled on and off. The difficulty would be finding a user-friendly and intuitive way to integrate it into the editor's workflow. This is a feature that would require considerable testing, and hence falls at the lowest priority for any of these features given the time constraints. Profiles are a feature that other development projects have tried (our client specifically brought up the example of Netscape), and would require a new approach to be worth pursuing.

VII. Deliverables

Management Deliverables:

1. Requirements Analysis – A document and a presentation to go over the formal requirements of the project, both functional and non-functional. This deliverable establishes the system's functionality, constraints, and goals through consultation with the client and ensures the team is working on a system that closely matches the wishes of the client. This deliverable gives the clients chance to modify and correct items that were mis-communicated or missed out before allowing the team to proceed further in the design.
2. Design Document – A document and a presentation to go over the design of the system. This is the development team's opportunity to go over how the project is to be implemented to the client. The work on this deliverable will be led by more technical and experienced members in the group, based on the understanding of the requirements established in the previous deliverable. The design document should contain both the system architecture and user interface.
3. Source Code – A document, a presentation, and the source code of the final completed project. This final deliverable wraps up and concludes the project. In this deliverable, the team delivers the final implementation based on the requirements specified and the design developed in previous stages.
4. Test Document – A document with the plan and results of testing final completed project. The system must be tested thoroughly with unit tests and a final acceptance test before deployment to the Atom community.

Technical Deliverables:

1. An *Atom integration* that supports project/folder-specific configuration and package management – An addition for the Atom editor that any Atom user can install and use it to manage per-project/per-directory configuration (e.g. font size, color scheme), as well as project-specific environment settings (e.g. specific extensions that should be disabled or enabled). This may be either an Atom package, a command line script that configures Atom to have our desired features, or changes to the core Atom program.
2. A *graphical user interface* for Atom users to easily toggle configurations and packages – An administrative interface where a user, after installing the aforementioned package, can easily view, manage, and modify different configurations and environment settings.
3. A *landing page* showcasing the package – A web page to describe and promote the package the team developed. Detailed usage instructions and functionality of this package will be presented.

VIII. Outline Plan (Principal activities and Milestones)

To begin with, the team identifies this project as an iterative design process, progressing from first achieving the core features the team promised, then the secondary features in the next iteration, and finally long-shot goals. Hence, the team designed the milestones using a numbering conventions similar to that of software versioning. Milestone 1 serves the delivery of Assignment 1. All Milestones numbered 2.x are to be completed before Assignment 2's presentation; similarly for 3.x and 4.x. In this way, the team has the major milestones to give us a large picture, as well as detailed milestones to ensure progress.

Milestone 1 (February 14, 2018) – Complete Assignment 1: Feasibility Studies.

Milestone 2.1 (February 21, 2018) – Requirements Analysis (draft)

An initial draft of the requirements analysis should be done after a formal requirements gathering meeting with the client. Preferably, the team should also have talked with the client's contact at Github/Atom, and learned about their expectations, discussing these with the client.

Milestone 2.2 (February 28, 2018) – Requirements Analysis (final)

A final version of the Requirements Analysis should be completed. It should detail the client's requirements for the software, categorizing the requirements into primary, desired, and optional features. It should have been reviewed and granted consent by the client by this date. At this point, the requirements should also be discussed with the Atom team as well as the ability to pull our code into the Atom core.

Milestone 2.3 (March 5, 2018) – Software Architecture + Implementation for per-project configuration

Since the team is building on top of a sophisticated, open-source project, software architecture and compatibility should be designed and confirmed at an early stage. As Milestone 2.3, a design document of the software architecture should be done and presented to the Client to gather feedback.

Implementation of the minimum features will also be done: the team should be able to support per-project configuration (as opposed to the editor-wide, global configuration by default)

Milestone 2.4 (March 12, 2018) – Preliminary Git Integration Design

A preliminary design for the UI of the git package with specifications on how to manage the scripts and checkout features.

Presentation and deliverable for Assignment 2 should also be prepared.

Milestone 3.1 (March 19, 2018) – Completion of Design + Implementation for per-project package management

Design should be completed and finalized. Implementation for the next-stage feature should be done: user should be able to turn on/off packages in a per-project fashion. A reference would be the package.json in npm. A demo will be presented to the client.

Milestone 3.2 (March 26, 2018) – Implementation for per-folder configuration management

Taking what was done in Milestone 2.3 one step further, the team should support updates to the Editor configuration that listens and reacts to tab changes. Configuration should be swapped not only for different projects, but for files in different subdirectories under the same master project. User should be able to load different custom configurations and manage them. A demo will be presented to the client.

Milestone 3.3 (April 9, 2018) – Implementation for per-folder package management

At this point the team should support turning on/off packages in a per-directory fashion, which is a major improvement from Milestone 3.1. A demo will be presented to the client. At this point we should open the pull request to GitHub as well

Milestone 4.1 (April 23, 2018) – Testing, Debugging.

The project needs to be well-tested and debugged at this milestone. After passing the acceptance test, the team would preferably do user testing on INFO 2300 projects with help from the client. Comments would be collected and if any issues arise, they should be resolved at this point.

Milestone 4.2 (April 30, 2018) – Git Feature Integration

As a desired additional feature, the team will implement the git package for easy processing of students' work.

Milestone 4.3 (May 5, 2018) – Presentation 4

Entire package should be ready for delivery. Documentation and user instructions, preferably as a landing page featuring the package, should be completed. At this point, the code should be ready to be merged into github. Only barriers should be awaiting approval from Atom.

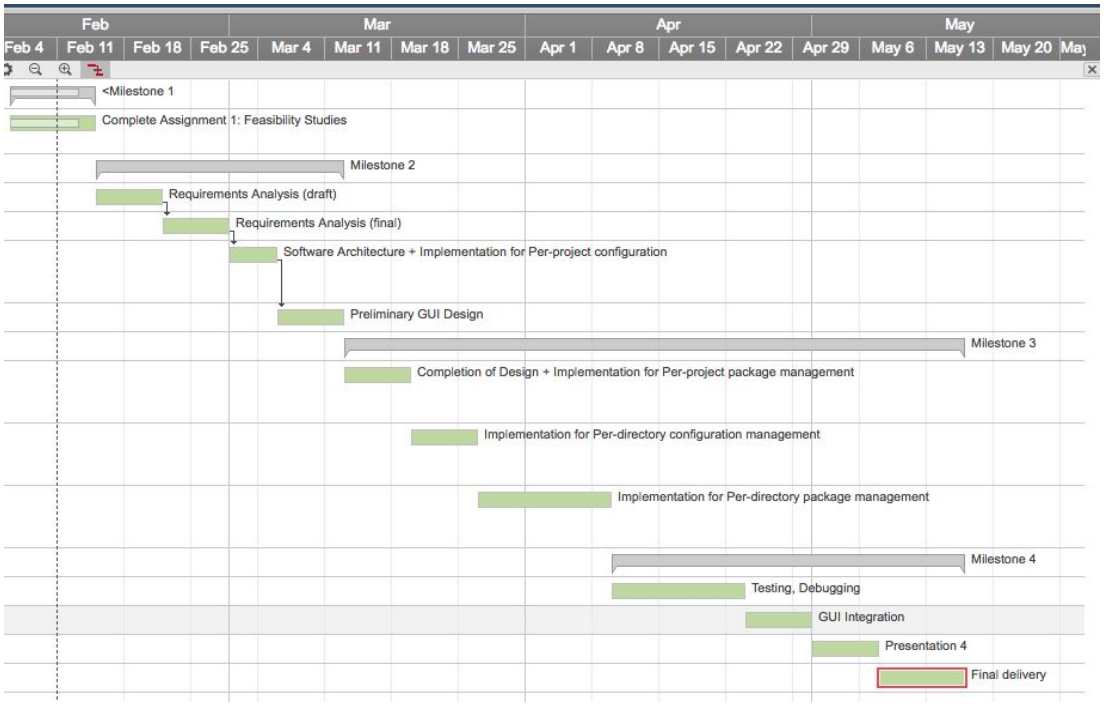
Milestone 5 (May 16, 2018) – Final delivery

All deliverables are prepared and a presentation is presented to the Client. The project source code should be handed over to the Client for the final milestone.

IX. Task Gantt Chart

https://app.smartsheet.com/b/publish?EQBCT=60824412785c43d7b05cd826c82f088f

Task Name	Start Date	End Date	Duration	Predecessors	% Complete	Status
<Milestone 1	**02/06/18**	**02/14/18**	**7d**		**80%**	**In Progress**
Complete Assignment 1: Feasibility Studies	02/06/18	02/14/18	7d		80%	In Progress
Milestone 2	**02/15/18**	**03/12/18**	**18d**			**Not Started**
Requirements Analysis (draft)	02/15/18	02/21/18	5d			In Progress
Requirements Analysis (final)	02/22/18	02/28/18	5d	4		Not Started
Software Architecture + Implementation for Per-project configuration	03/01/18	03/05/18	3d	5		Not Started
Preliminary GUI Design	03/06/18	03/12/18	5d	6		Not Started
Milestone 3	**03/13/18**	**05/16/18**	**47d**			Not Started
Completion of Design + Implementation for Per-project package management	03/13/18	03/19/18	5d			Not Started
Implementation for Per-directory configuration management	03/20/18	03/26/18	5d			Not Started
Implementation for Per-directory package management	03/27/18	04/09/18	10d			
Milestone 4	**04/10/18**	**05/16/18**	**27d**			Not Started
Testing, Debugging	04/10/18	04/23/18	10d			Not Started
GUI Integration	04/24/18	04/30/18	5d			Not Started
Presentation 4	05/01/18	05/07/18	5d			
Final delivery	05/08/18	05/16/18	7d			



X. Visibility

The team has decided that team members will communicate through Slack and schedule meetings through Whentomeet on a need-by-need basis. The team has determined tentative “good” and “bad” times. The team have allocated Tuesdays at 4:10PM as a generally safe time for short-notice meetings. Since the team is relatively large (8 members), it is likely that there will be times when not everyone can be present. However, the team should all be able to meet any Monday at 7:30PM.

The team will communicate with the client on a need-by-need basis as well. The client has reasoned this the best plan since there is little understanding of how much involvement he will have in the overall development. Meetings will be set up through email and will take place in his office in person. The team will report the project progress during these meetings and through email.

XI. Risk Analysis

Progress will be monitored and tracked through regular team meetings, as well as potential problems throughout any stage of development. Any issue potentially affecting any core deliverables should be immediately brought up in a shared group channel, and said issue should be the main focus of the team effort until resolved.

Client Risk

According to the client, the problem to be solved has been a pain point for a long period of time. It is not likely that the client will cancel the project or change the basic requirement. All the development will be in-house by the team based on communication with the client. Though it is unlikely, it is still possible that the team may underestimate the problem or misinterpret a requirement. However, appropriate meetings should safeguard against this. It is possible that some of the features have been explored and built by someone outside the team and the work can be reusable. In such cases, the work will be cited.

Usage Risk (Pull Request)

Our client has expressed that he himself will not be able to upkeep this feature. Thus, he has expressed interest in this feature being merged into Atom Core. While we have planned discussion with our client’s contact’s in GitHub as well as various milestones in the merge process, the pull request may not be accepted by Atom, or at least done so in a timely matter. If Atom chooses to not accept our pull request or becomes backlogged with other requests, our code could never be merged in and thus not upkept. With no maintainer, the code would not be usable. As mentioned above, appropropriate measures are being taken to be in contact with the Atom developers throughout this process as a safeguard.

Technical Risk

A possible risk is that the Atom-provided packages for modifying global settings within a package do not allow for the full scope of features the team requires (eg. selective package activation). In such a case, a fallback option may be to use bash scripts that operate on the filesystem to directly modify the global configuration file on disk.

Timeline Risk

A potential “risk” may exist in terms of the timeframe of the project. While attempting to analyze the scope of per-workspace settings, some core features were deemed quite trivial to implement, whereas some others were deemed possibly too much in terms of scope (for example, correctly nested workspace configurations). There is a slight risk of ending the semester while still working on a tougher deliverable. In said case, a viable fallback is to communicate with the client and determine whether this is an acceptable risk. If not, the team can focus more on extending the other optional goals, such as the user interface.

Personnel Risk

The large start-up time required for this project presents a risk to delivery. No one on the development team has worked with Atom packages before, nor do all members have Javascript proficiency. To mitigate this risk, the team has evaluated the technical proficiency of each individual team members and allowed abundant amount time for start-up and technical acquisition. Furthermore, the large size of the development team combined with other current team member endeavours may result in some projects being complete and some behind schedule. For this reason, the team has implemented the detailed schedule above to keep members on track, with internal progress report due to ensure development is on schedule. If one project does fall behind, the team will discuss and allocate resources accordingly and immediately. A fallback option will be assigning proficient developers for development tasks while others can still take care of other tasks such as writing documentation, preparation of presentation, etc. Another fallback option is reduction of extra features and more focus on the core features. Before each milestone, team members will have assigned tasks and mid-term discussion of possible impediment during the development.

XII. Conclusion

Based on the analysis presented in this feasibility study, the team has collectively agreed that the project “Extending the Atom Editor for Per-project/Per-directory Configuration Support” is feasible. The benefits are significant enough to justify the development effort required, and the team members also possess the adequate skills to implement the project following an executable timeline.

Given the time constraint of one semester, the team believes that all *immediate* features requested by the client can be satisfactorily fulfilled by the preliminary deadline for the project, May 5, along with some or all *additional* features proposed by the client. The recommendation of the feasibility report is to proceed with the aforementioned project.