**Ride Share Scheduling, Software Architecture, Design and Production**

Richard Kershner

Date: 10/08/2023

**Ride Share Scheduling, Software Architecture, Design and Production**

**Introduction**

Computer and software engineering deals with looking at developing a system, going through the whole cycle, from the start of visualizing something new to the end which which goes beyond putting out a product and might even go to the retirement or end of the software or systems used. Central to engineering of any kind is due care, the concept of making sure to leave nothing out, look at it from every direction possible. This really unfolds through all the stages, from the initial proposal to maintenance, future upgrades and retirement of a system or software package and its use. It is impossible to cover every contention so to explore this, a single project will be covered in its different phases, in this case exploring algorithms for ride share scheduling. This will be covered in the following four areas: The topic proposal, system design, testing and a final evaluation.

**Topic and Proposal (Previous Assignments)**

Longmont, CO city council is pursuing a new “microtransit” system of running 4 to 6 vans or small buses, *Cadey, D. (2023, April)* utilizing a third party to run the program. This service already exists and is being implemented under another company in the Denver Metro area with over 60 small buses in service across over 40 areas. There are challenges in the current scheduling system needing to be addressed which will be covered under the challenge, the goal, and scope. In addition, a description of the information system and a timeline will be supplied.

**The Challenge**

The current algorithm used for scheduling rides is outdated and, in many cases, creates a driver route that is significantly less then optimal. Numerous passengers have complained about driving past their destination once, twice and even three times before their drop off comes up on the manifest, the instructions the driver follows. They have also complained on booking a short ride which turns into a forty five minute plus tour of the city. In addition, the routes are less then optimal, zigzagging which can be seen in Appendix G, Figure 8, as well as in a final output in Appendix G, Figure 9. in some cases backwards, wasting resources used and creating longer drive times. The current implemented module was designed over a decade ago and is outdated.

**The Goal**

The goal is to explore options for optimizing the scheduling module for better efficiency in the actual drive route. This utilizes a prototype type model with clear statistics and graphs to show the different benefits of different algorithms. Once an optimized algorithms are explored, the findings will be submitted to the city council as well as a number of vendors seeking the possible “new” contract.

**Scope Limitations**

The current project does not include specific passenger considerations like how many seats are available on the vehicle, how many wheelchair spots are available, or added time needed for people with disabilities for boarding/departing. Additionally it does not address drive issues like traffic hour, driver bathroom stops, etc. These additional factors should be explored and the final system designed being made to accommodate future upgrades. These aspects would greatly affect a scheduling program as a whole but should not overly effect testing the actual algorithms.

**Description of Information System Used (Major Components)**

The project build is a scheduling prototype that will be implemented in Python. The central concept is to be able to test different algorithms and, although the actual code will be optimized for testing and not for deployment, having it uniform and modular is very important. For example, the request file and mapping information is created before hand in the random request part. This should accommodate swapping a file with actual ride history of an existing system. A diagram showing how different modules in the project connect with each other can be found in Appendix A, Figure 3. As shown in the diagram, these are the following major areas: Generating random ride requests, routes generated, different algorithms to generate different routes, and the final statistics and graphs to explain the benefits of the different route options. The development and testing of these modules will be covered throughout the rest of the paper. Central to the overall development there are two major areas that need to be expanded on prior to development. One is source of mapping and the second is the calculations for the algorithms.

Mapping source is it’s own challenge and *Google Maps(t.d.)* services are free for basic developer and require a license for commercial use. Because the project relies on having numerous routes to test, In creating the mapper, a sample set of information was downloaded and altered prior to this project. Running the mapper program each time, because of the amount of data downloaded, raised a flag from the services. To accommodate not having to pay for a license, this information is stored in it’s own file and a mapper object was already created to load and call route information. The mapper was designed to be easily replaced by a full mapper software if testing on actual data and an actual system.

The second part that needs special mention is the algorithm module. The whole project centers on the concept that scheduling algorithms get complex. In exploring the initial algorithms, it is noted by *Coester, C. & Koutsoupias, E. (2018, Nov.)* on scheduling taxis and *Masoud, M. & Belkasim, S. (2018, Oct.)* on scheduling cement trucks. It is noted that the complexity of approaching different algorithms from an academic mathematical challenge is extremely complex and does not necessarily work when applied to real scheduling issues. Instead, this project proposes exploring different algorithms from a testing platform on semi-real data simulating it’s use in accepting one requested route at a time. Pending on the findings, this can be expanded to use actual data and a licensed mapping service. See Appendix D for initial class diagrams of algorithms.

**Timeline**

To form an estimate of the timeline, a separate document was created with the initial design objects set. Each object (mostly classes) was given a score guessing on the lines of code. Complex code was doubled. In evaluating personal past programming, this translated as 10 lines of code per hour. Roughly 300 lines of code translates as 30 hours over 6 weeks or 5 hours a week in programming. In redoing the timeline, algorithms would have been tripled or quadrupled for logic testing. The rest was fairly accurate. An additional 10% to 30% could be added to documentation review with the lower number accommodating a project that already had a set documentation and the later being a new project, like this one, needing extra time to explore requirements.

The initial timeline, below, does not include the analysis and most of the design phase. For actual allocation of resources these would also need to be considered. A breakdown of the build setup timeline can be found in Appendix B. In the timeline, there are two testing processes. One is by code, making sure everything comes up correctly. The second is a hand walk through where each step is double checked to make sure the logic is working correctly as well as the code is functional. This does not replace testing while the code is produced to make sure it is functional but is a second and third test. Documentation has two parts, inline with the code, making sure, once the code is complete, it is readable. Additionally, documentation focuses is main parts with a special write up on the algorithms and their functionality. A final presentation is undetermined but would include a report on the findings, possibly a tutorial video.

**Conclusion Proposal, System Components and Timeline**

This brings the project introduction, the system setup and a planned timeline together. Having clear analysis of different aspects of the proposed project will help lock it in to improve the chance for success. It should be noted that there is a continual loop fine tuning all the pieces, including tapping into the actual software design, below, which helped to plan the full timeline.

**Software Design**

Software design on project ride-share testing scheduling algorithms, covers an overview of final design. The initial components are meant as a guidelines and not the final plan. As the project evolved, the Waterfall Agile hybrid module of design was a great starter for getting a flash perspective of what might be needed. The project was definitely more Agile and classes where adjusted a few times to accommodate needed changes. For the project the following will be covered: Software components, testing, final output and documentation.

**Software Components**

Software components fall into four major areas. Requests, routes, algorithms and statistics. Diagrams for the components can be found in Appendixes C and D. Each one of these handling different aspects and overlapping in different ways.

***Requests***

Requests need to cover the basics of an individual filing for a ride. The request a pick-up and drop-off locations denoted by addresses as well as a time. These are then encapsulated into groups, where each group is a set number of requests. For example, 5 requests, if none are rejected as not being able to fit, would denote a possible 10 stops, 1 for pick-up and 1 for drop-off for each request. The time is a limiting factor as a request should not be allowed to push the ride back or forward too much in time from the requested time. Finally there is an epoch run where many groups are processed to generate averages for different statistics as noted below in statistics.

***Routes***

Each route is generated with a select algorithm. The route then receives requests one at at time and attempts to fit them in the stops based on the algorithm. The route, then, is denoted by a time and a place for each stop. This is what a driver might see on their manifest. Stop at address 1 at a certain time. Then address 2 at a certain time, and so on. Each stop should include an ID referencing the request in the group, used for statistics and for mapping denoting when dropped-off versus picked-up. Over all the routes are setup in an epoch container of multiple groups which have multiple random requests. This allows for testing different algorithms across a significant amount of data. Future builds might incorporate different aspects of routes as noted in documentation, below.

***Algorithms***

This is central to the project and extra care taken in understanding the logic of the algorithms and what they mean. Key is noting how and what the different algorithms are optimizing on. *Masoud, M. & Belkasim, S. (2018, Oct.)* is clear on how time windows work which can be leveraged to create inheritance where a base class can have general scheduling functionality while each algorithm inherits and does a specific algorithm. *Agrawal, A., Jain, V., & Sheikh, M. (2016)* brings in different factors and how they effect ride scheduling. Part of the project is exploring how different algorithms can be implemented. For example, using a window of + or – 45 minutes and taking the options in that window, does sorting them by closest to requested time or by which one is able to fit the most rides in?

***Statistics***

The final area is connecting the different epochs of random grouped requests and the different routes they create to generate clear tables and graphs comparing algorithms. The first part is collecting and computing the routes after each request group is processed. Once each request group is processed, the different routes generated from the algorithms need to be evaluated for how many rides are rejected, how actual time differs from requested time, how the rides effect distances traveled, etc. Once these individual statistics are collected, they should form a pool of data across epochs of which the means (average) can be computed. The statistics module is also responsible for final printout of data in both table and graphic form as well as possible future adjustments that might include standard deviation and other factors. These statistics would, as noted in different algorithms, calculate which algorithm, on average, best schedules rides, least issues when deleting rides, etc.

**Final Output**

The final output displays a random set of group generated routes to show how the routes are different under different algorithms as well as different markers averaged over the full epoch run of ride request groups. Both should be shown in table form, raw data, as well captured in graphical format as shown in Appendix G, Figure 9. In the graph for the route, pick-up and drop-off are denoted with specific symbols and the individual ride is tracked with a unique color. This shows how the route, as scheduled by different algorithms, is traversed by the driver as well as what the riders experience. The statistical graphs should have one for each feature of importance with an example shown for ride validity, also in Appendix G, Figure 9 where it shows how different algorithms rate against each other. One of the aspects of comparing these numbers in the complexity of how a feature is effected by which request it is on. In the graph, ride 1 is always accepted. In request two, time-based reject more rides then vect-based. As the number of requests are processed over a period of time, different systems will clearly show a higher average of rejecting rides. This concept needs to be computed and compared for all features across all algorithms.

**Testing**

Testing is broken into three different areas as part of both the phase of development and specific to certain areas. The three different areas are inline, hand and final. The full depth of these different testings is explored below. What should be noted in the early stage is that testing needs to happen early, often and during every aspect of the process where possible. For example, even as code is being developed, the code can be tested with inline prints, variable tracking, code stepping and other tools. Even though this is assumed in coding, it should be documented as part of the production processes as well as clean up and documentation of testing done as applicable.

**Documentation**

Documentation becomes paramount. Even the most simple changes, a single line altered, can crash a whole system. Going back through different modules and lines of code can take hours to track a bug or needed upgrade if not properly documented. In this project there are the following main areas of documentation: Inline, system, algorithm, and tracking.

***Inline Documentation***

Easy reading and changes to code need to be viewed in the documentation so if a module needs to replaced or updated, it is clear how it works. This is really secondary to the project and centers on good coding. For example, ten lines of code without comments could take hours to traverse the logic used while a few lines of comments can spell out the process and flow for easy updates and bug testing. A formal commented intro can then mirror more formal system documentation.

***System Documentation***

At any one time, a final production, either in development or in use, needs a “flash” of what is going on so that it can be viewed and used for current and future development as well as bug tracking, maintenance and tutorials. Clear dated version numbers and recent changes made available for quick reference. It should be noted that even as documentation is locked in as a point of reference, new versions can replace it and the last build archived. This documentation is also a history of production.

***Algorithm Logic***

Since the project centers on algorithms, clear documentation of the logic used and how it functions is important. This will be necessary for the use of the final output when embedding in reports. Given the complexity, care should be taken to make sure it is clear.

***Project Tracking***

In the process of completing the project, in time, a clear path and steps need to be taken. In this case recording each module, process and function to work on with status and completion notes. Additionally a document clearly noting what the different processes are. For example, a code built with inline testing followed by building into main product and hand testing. Specific order of steps needs to be clearly stated for modules as well as tracking the status of each part. Added to this could be bug tracking, change requests, etc. In a full project, tracking is its own major documentation and, in many cases, works closely with systems documentation even as it is done separate.

**Software Design Conclusion**

The final project should be completed within the confines of the time constraint given by the 8 week length of the class with the help of closely monitoring which modules are being completed and that they are on time. Additionally, not shown here, a document to track steps required in creating and posting the code has been created to aid with the process and for future modules.

**System Project Plan**

In developing the software project plan, software design information is incorporated into classes and mapped interactions. In previous experience, I’ve found the two tend to go back and forth between each other and that it is rarely a clear one step after another process. Moving from the planning stage to development stage, the following UML diagrams will highlight classes, interactions, message interaction and a state machine.

**Class UML Diagrams (Appendix C and D)**

***Main, Main\_Script\_Tester, and Mapper***

The primary run class, Figure 1, evolves from the options in the planning phase, where it becomes clear that a main code iterates through request group, iterating through algorithms which iterate through route instances, triggering a statistical module after each group. This is the “Main” for this project and, if this module was inserted into a full system, this would be the complete test module. The Main\_Script\_Tester, Appendix D, is added in to contain inline code testing. The inline code testing is produced during the initial build of the classes and then copied into the script tester for easy future reference. Once copied over and finalized, the inline testing script is deleted and the class is set, built, for use in other modules. The tester should mirror aspects of the “Main” class for easier production and testing through out the coding process.

Finally, part of the over all package, is the mapper, Appendix D. This was added in to accommodate information between two addresses. One of the issues arising is the licensing and cost of a mapping software. To not incur added licensing expenses, a file was created from Google Maps, *Google (n.d.)*. Most mappers allow a certain amount of downloads before flagging for commercial use. This allows the download and routes for a select group of random addresses.

Central to the production are the rider request classes and the routing classes shown in the case diagram, Appendix C. Rider request classes are random simulated address encased in groups which are encased in epochs generated at the start. Route classes, on the other hand are created during the processing of a request group, with one route per algorithm during request group processing. The pick-up and drop-off addresses in each request become reflected in separate route stops. Time options for the stops are calculated from the request time.

***Statistics and Algorithms***

The statistics class where already noted in the previous planning stage in Appendix D. The statistics class takes a route, calculates the statistics from individual routes and stores that information for each algorithm. All algorithms have a base process that creates a window of optional places to insert a stop. The algorithms then orders the list, prioritizing which stop should be tested first. Once created, “Statistics” is given each completed test route and calculates the statistics for that route and finally displays information for the means over all the different algorithms and different markers. The most common marker according to *Agrawal, A., Jain, V., & Sheikh, M. (2016)*, being distance. It is also important to look at time windows, *Masoud, M. & Belkasim, S. (2018, Oct.)* for understanding logistics and routing. The final markers are then incorporated in method of “addRoute” to statistics for individual groups and the final calculation.

**Interaction Diagram**

The interaction diagram was the most interesting in the planning stage. In fact, it was redrawn four times and, each time, options where moved around. This translated back into the class diagrams making for interesting editing. The interaction diagram shown in Appendix E, Figure 5 allows for a view of how the different data is being moved between different classes. The many redraws come from the many different options. The final plan centered on a Main class that controls most of what goes on. Without the interaction diagram, small details in how to interact between classes and which classes should handle what, would have been missed.

**State Machines**

A state machine of the main class, Appendix E, Figure 6, with route instances included, helps to denote routes don’t need to be instances for each route group, but could be reused. This would use less resources and could simplify the program. For this project, a state machine diagram was not as impact-full as other diagrams but still useful.

**Conclusion, System Project Plan**

The project plan aligns closely with the project design. There where alterations with the most predominant one being the use of a main processing code to contain the full of the process and how to deal with a mapping module. This still tracks closely with the time line for development of the full project.

**Testing**

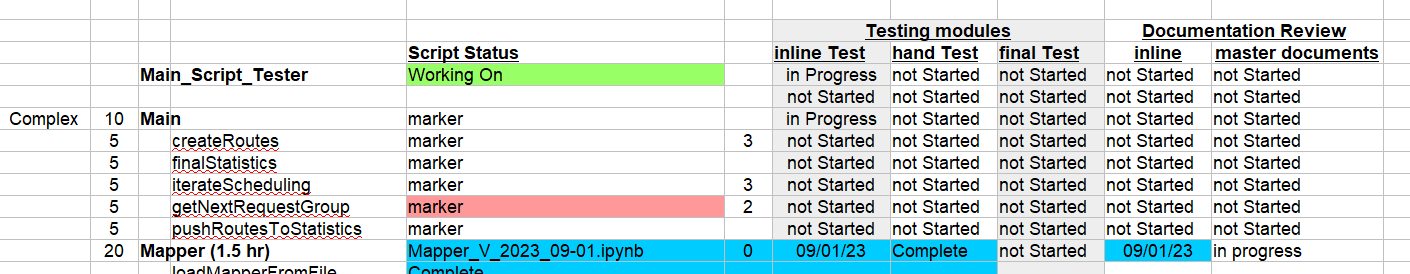
Software testing should encompass any and all critical aspects of the end product as well as leverage concepts, in the case of human interaction, that go beyond functionality and into the human experience. It is noted in the final evaluation that if an Agile or Spiral methodology had been used, the first module build would have generated test data for all other modules instead of hand creating basic data. The bottom line is the product has to do what it needs to and do so in a way, in the case of this specific project, has an end product the looks good. Testing will look at an overview, specific testing and test data.

**Overview**

Central to the testing is doing it throughout the project, integrated as part of the whole process, *Bijlsma, A., Passier, H. J. M., Pootjes, H. J., & Stuurman, S. (2018, Oct.)*. Different aspects of testing feed into or are part of other aspects of testing as noted in Appendix A, Figure 4. The very test code used to test each module as it is produced in a white box type testing mode, where the code might even have test prints to make sure the data flows correctly, is then cleaned and copied into a black box testing setup. Each one of these processes is linked to the development cycle as a whole and tracked in documentation. The individual testing parts are done continually throughout the project and, if there is a failure or needed changes, original testing should be available to easily go back and test different parts. For example, if hand testing fails, the inline testing needs to be available. Same with the final testing, hand testing steps and outputs need to be saved for easier tracking of errors. Again, central to all of this, is the fact that testing is done through out the project as to identify major errors that could affect the design and plan, early, and correct alter modules or parts of the system instead of having an error cascade backward causing larger fixes and alterations to multiple modules.

***Tracking***

Documentation on a project like this one, centers on a testing sheet, Figure 1, and testing code. When producing a package as a one-person process, this becomes the main part of development. Documentation on what each testing phase means for different modules is also important. In the

**Figure 1**

main tracking sheet, script status includes the build version, when inline test is completed, hand test completed which, in some cases is part of the inline testing and when final testing is done. It is also important to identify all the classes and procedures that need testing, *Ibrahim, R., Amin, A. A. B., Jamel, S., & Wahab, J. A. (2020)*.

***Incremental Testing***

One of the columns in Figure 1notes a number. This will need to be upgraded with a tag of order or level. Level 0, as Mapper is noted as, means that when integrating with the software packages, it needs to be designed and tested first. Same with Main processes. This allows for errors in the planning to be corrected for in developing related aspects. Although this should be handled during the planning phase, not everything can be planned for.

**Testing Parts in Project**

Note testing diagram in Appendix X shows how they interact as well as the tracking document above in Figure 1. Even though additional testing can be applied as needed, the main testing fall into three main parts: Inline, hand, and final test.

***Inline Test***

Inline test deals with test code being written to test different aspects of a module or process. For instance, a code needs test data sent and output data displayed. Even code can be added inline to print specific parts of a procedure can be placed in the inline testing. Once the inline testing is done, a copy with the testing goes into a version folder, the main test code is copied to a testing module for future access in final testing and a build is done from iPython into a module Python folder.

***Hand Test***

As an extension of the inline testing, data is produced. This data needs to be printed and walked through to make sure it is being processed correctly. In more basic procedures, this is part of the inline print test. For complex procedures, specific to algorithms and possible request options, this needs to be done with a full walk through of hand writing the math and double checking the output.

***Final Test***

The final test has a number of aspects all of which are black box testings. One, it integrates all the parts which is really the Main build. Two, it runs the test\_main, which is designed to mimic the main build but have test procedures generated from the inline testing done with each module. The final output has it’s own challenges as visual representation requires additional care, *Fulcini, T., Coppola, R., Ardito, L., & Torchiano, M. (2023).* In this it is a visual output of graphs and data frames which have to meet an easy, comprehensible and enjoyable view. Having worked in other graphical outputs, this can take a number of iterations of the output and altering the code to maximize the experience and validate the information.

**Test Data**

***Route For Testing***

Creating an array of different stops mirroring the data structure, was required for testing different aspects of the code. As can be seen below, these are the stops used in a generic test of making sure the code is working correctly and no errors are thrown.

stops for testing

0 --> [3, -80, -80, -6, '2114 Lincoln St']

1 --> [1, 31, 0, 10, '879 Neon Forest Circle']

2 --> [1, 33, 5, 33, '1819 Sunlight Dr']

3 --> [2, 35, 30, 45, '1715 Iron Horse Dr']

4 --> [3, 45, 45, 60, '2114 Lincoln St']

5 --> [2, 80, 80, 90, '1819 Sunlight Dr'].

As seen above, there is the ID number, pickup time, minimal time, maximum time and an address. This allows for testing how the algorithms processes and existing route and, the above test data, can easily be altered to test for boundaries. In this case a time of 80 was added in for ‘1819 Sunlight Dr’ to see how the software acted if a time was way beyond push down range. In addition, this data is submitted to the final statistics tracker to compute different aspects of route.

***Requests for Testing***

Central to each of the components are requests being scheduled. This requires a pick-up address and a drop-off address along with the requested time as shown here.

Request group

['1225 Ken Pratt Blvd', '225 E 8th Ave', 35]

['2051 Sumac Str', '28 University Circle', 0]

['2016 Mount Sneffels St', '422 Overbrook Ln', 25]

['2051 Sumac Str', '420 Kimbark St', 15]

These apply to any aspect of the scheduler except the final statistics. It should be noted that part of the mapper generates these values randomly from a list and in later testing can replace this locked test data. The bonus of set data is once hand calculated, it does not have to be recalculated.

**Testing Summary**

All testing should center and be tied back to the original proposal and project scope. Is the quality of the project, in this case centering on the logic and algorithms, working? How will it be used and viewed as an end project, in this case data and graphics outputs? Additional questions that might arise in Computer Engineering projects are security, run time, critical failures, etc. All of these should be noted as to their importance and testing implemented as needed.

**Final Project Evaluation**

The short question is did the software generate statistical analysis of different route algorithms which can be used to asses future scheduling builds? The short answer is, Yes. As a logic exploration prototype it functions as advertised. Additional features necessary for a full presentation to organizations or individuals which are part of the inspiration for the project, had to be left out of the original build to keep it within a realistic time-frame. As part of the complete project, there future implementation will be noted in parts of the architectural evaluation. Central to the project is to evaluate it from the Computer Engineering architectural perspective which includes the SDLC (Software Design Life Cycle) and methodology; quality attributes; and viewing additional perspectives (asking the tough questions).

**Software Design Life Cycle And Methodology**

Two aspects of the SDLC stand out in the project. One being the focus on individual parts, on planning, analysis, design, implementation, and maintenance. The second being the choice of over all methodology.

***SDLC Individual Parts***

The individual parts of the SDLC where mostly right on with adjustments needed throughout the project but no change that was so out of scope as to be considered major project drift. Most conspicuous was in the design mode where utilizing Python as the code with it’s more loosely defined variable structures left out the need for extreme typed and defined use. In a distributed software this would greatly increase the chance for security issues but in a research and development, R&D, throwaway prototype environment, this is well within scope and use. Over all the full cycle, minus maintenance which will happen post project, where well defined and completed.

***Development Methodology***

Methodology is significantly more interesting and the greater learning experience. The over all concept going in was that a primarily Waterfall setup would suffice as most logic and concepts appeared well defined with certain aspects, like different statistics and algorithms to use, could be implemented in a more Agile format compartmentalized in the Waterfall methodology. So the original methodology was a hybrid Waterfall/Agile process. In most data science R&D projects this rarely works and, because of their nature and how new organized software development in them is, most are currently done ad-hock, *Data Science Process Alliance (t.d.)*. Code is thrown together, see if it works, go back and then solidify it into something more traditional which causes problems with teams and larger projects. This is because the aspect of research has a huge affect of the over all process. *Data Science Process Alliance (t.d.)* notes a new concept, R&D methodology which *Kumar, M. (2022)* goes into more depth on. This becomes an Agile project alternating with ad-hoc break out sessions. In starting the process, defining the iteration of test data should have been the first stage which would have been an automatic testing of all other objects and parts of the project in an Agile methodology with each part being explored ad-hock then brought back into a formal production and documentation. In not identifying that aspect in the methodology, the timeline was a little heavy, running slightly behind, but still close to predicted. A proposed Research & Development Agile methodology can be found in Appendix F. In addition much time could have been saved. The end methodology should have been an R&D methodology loosely defined as an Agile/Ad-hock hybrid combination.

In combining the individual parts and seeing how they fit into the overall software methodology, the project, from a post completion evaluation perspective would have been completed faster and with a higher of quality would have moved towards the hybrid Agile/Ad-hock or R&D methodology. This also aligns with many of the Data Science projects I’ve already been part of. There is still little documentation or examples of its use showing this is in a development state of it’s own.

**Quality Attributes**

In looking at the quality attributes, four rise to the top, flexibility, test-ability, commercial development, and end use. Table 2 shows these listed with a score.

**Table 2**

|  |  |  |
| --- | --- | --- |
| **Quality Attribute** | **Scenario** | **Score** |
| **Flexibility** | Able to add new test algorithms and alter special test case data configuration. | 4 |
| **Test-ability** | Able to see the how the algorithms effect routes. (Time restrictions limited this) | 2 |
| **Commercial Development** | Able to use the logic and concepts for developing employable software. | 3 |
| **End Use** | Able to use data and graphics produced for final report to submit to entities. | 3 |

***The score is from 1 to 4.***

1. Failed to meet requirements.
2. Functions, meets minimal requirements.
3. Requirements met and functions with high quality.
4. Greatly exceeds requirements.

Central to the project was the flexibility to add and remove different algorithms which greatly exceeds the expected proposal plan. The end use of the project, the use of graphs and DataFrames for use in a publication, as well as the ability to follow the logic of the different algorithms for commercial development, both work with a high quality and within expectations. The lowest grade is for test-ability meaning that the algorithms should function as designated but additional testing should be included prior to publishing a formal report. Over all the project turned out expected results.

***Additional Perspectives (Asking the Difficult Questions)***

There are a lot of different areas to look at but a number of them really didn’t apply as much as others. Risk is lower as it is a throwaway prototype as is security and even performance (as related to processing speed of the routine.) What are central are drift in the production, problems that arose and gaps in knowledge which includes validity.

Drift in any production can be challenging and with an exploratory data science software development, it is practically baked into the project! In this case, keeping on track was fairly well managed. If a more flexible model was used, necessary drift for this kind of research software development would have been better accommodated. In short, drift was overly kept under control and an Agile R&D development cycle would have better addressed this.

Problems that arose included the original plans for classes and processes having to shift as they did not make since, once the development started, for certain functionality to be under certain class umbrellas. This would have been better addressed with an Agile or Spiral methodology embedded with the R&D add-hock mixed in. An outline for an Agile R&D methodology can be found in Appendix F. Having more incremental reviews of the whole project would have better addressed these problems. This leads to an interesting question of how to develop an Agile software, in this methodology, maximizing a team setup. When this problem comes up again, and it will in the work I’ve been doing, a more in depth look at it and the team involved will be necessary.

Gaps in knowledge put testing the final output for logic flaws and double checking a code walk through behind getting the project out and will need to be addressed before and if a full report is published on the data produced. These gaps directly affect the possible validity of the end project. This is fairly easy to remedy and should be accomplished in a few days. In future planning, additional time needs to be allocated in final validity of complex processes that have an additional component of logic and/or statistics to them.

**Conclusion**

From both the perspective of this project being the first full cycle in development of a software package from a computer engineering perspective that I’ve done, the project, as to be expected, was educational and had a few wrong turns. That said, the final project worked as advertised, producing the information which can go into a report to be filed to the Longmont City Council as well as other organizations that could benefit from it. In addition, posting of the code to a public get hub repository makes for an additional reference for contract work or job hunting. In short, it is a valid step forward in education and vocational possibilities in computer engineering and data science.

**References**

**Agrawal, A., Jain, V., & Sheikh,** M. (2016) *Quantitative Estimation of Cost Drivers for Intermediate COCOMO towards Traditional and Cloud Based Software Development*. Proceedings of the 9th Annual ACM India Conference, 85–95. https://doi.org/10.1145/2998476.2998488

**Bijlsma, A., Passier, H. J. M., Pootj**es, H. J., & Stuurman, S. (2018) Integrated Test Development : An integrated and incremental approach to write software of high quality. *Proceedings of the 7th Computer Science Education Research Conference, 9–20.*

***Brahim, R., Amin,*** *A. A. B., Jamel, S., & Wahab, J. A. (2020). EPiT :* A Software Testing Tool for Generation of Test Cases Automatically.  *arXiv:2007.11197 [cs.SE].*

**Cadey, D.** (2023, April) *Longmont City Council to Help Pursue Microtransit System for City.* Daily Times-Call, Longmont, Colo. Pulled from <https://www.msn.com/en-us/news/other/longmont-city-council-to-help-pursue-microtransit-system-for-city/ar-AA1cQCGV>

C**oester, C.** & Koutsoupias, E. (2018, Nov.) T*he Online $k$-Taxi Problem*. Department of Computer Science, University of Oxford.

**Data Science Process Alliance** (t.d.) Data Science Methodologies and Frameworks Guide. Data Science Process Alliance. Pulled from https://www.datascience-pm.com/data-science-methodologies/

**Google (n.**d.)  *Google Maps directions for Driving, API.* Google.

**Kumar, M. (2022)** Data Science in Research and Development Approach (R&D). *Medium.com*.

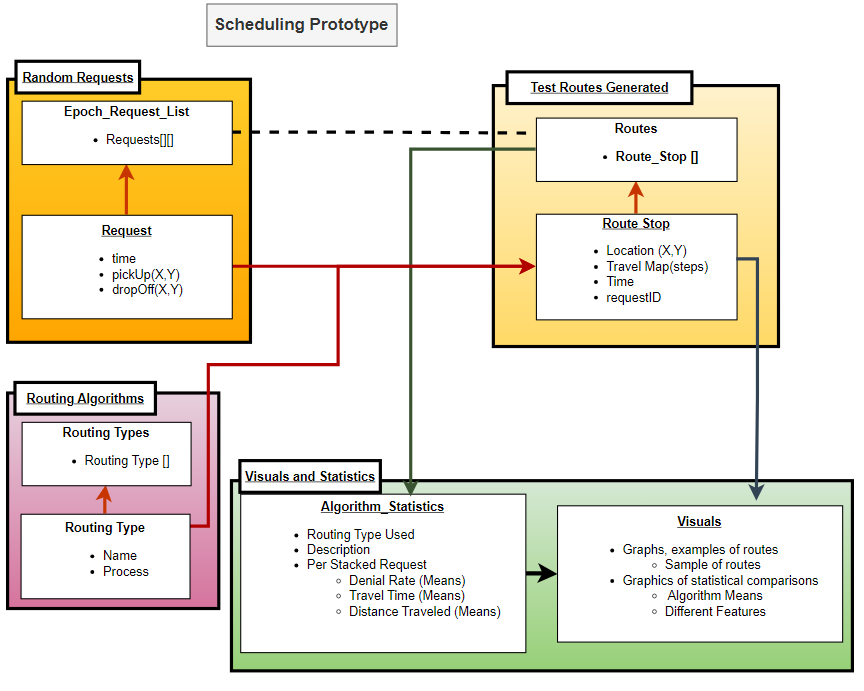
**Masoud, M. & Belkasim**, S. (2018, Oct.) *Optimizing Capacitated Vehicle Scheduling with Time Windows: A Case Study of RMC Delivery*. Department of Computer Science, Georgia State University.

**Appendix A**

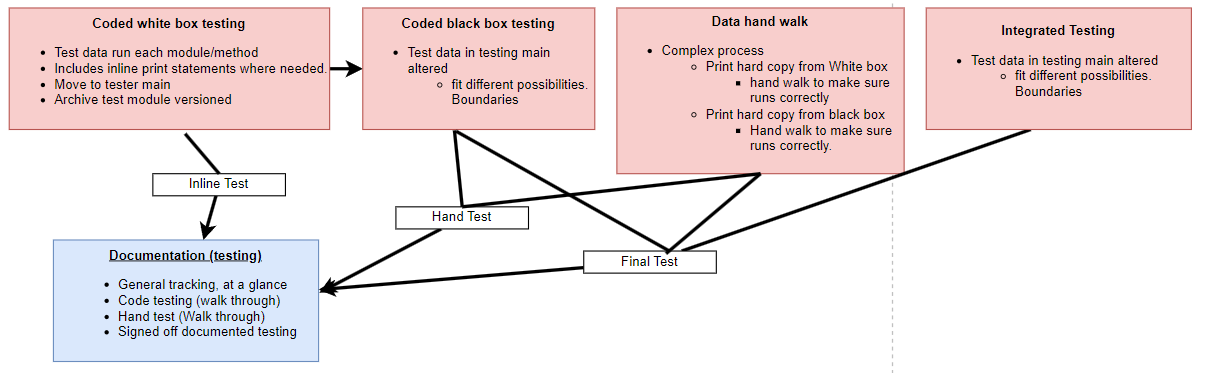
View of Connections in Proposed System and Testing Diagrams

Figure 3

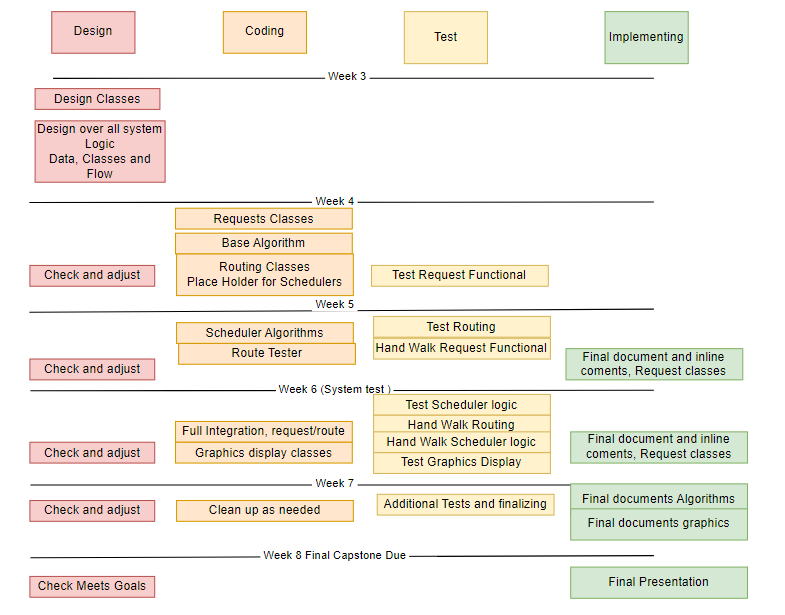
*Proposed System Modules showing Connections*

**Figure 4**

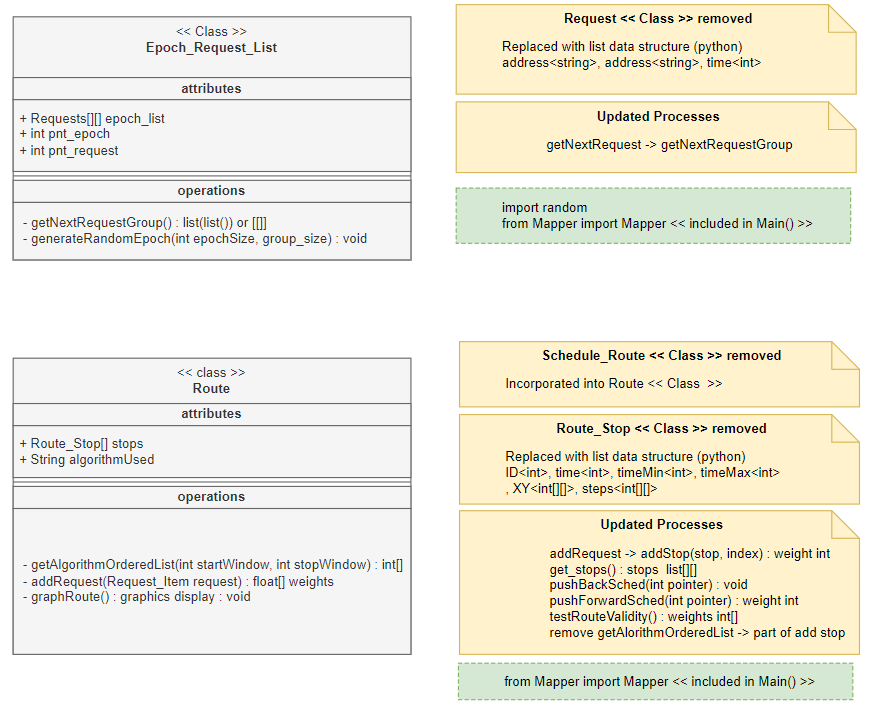
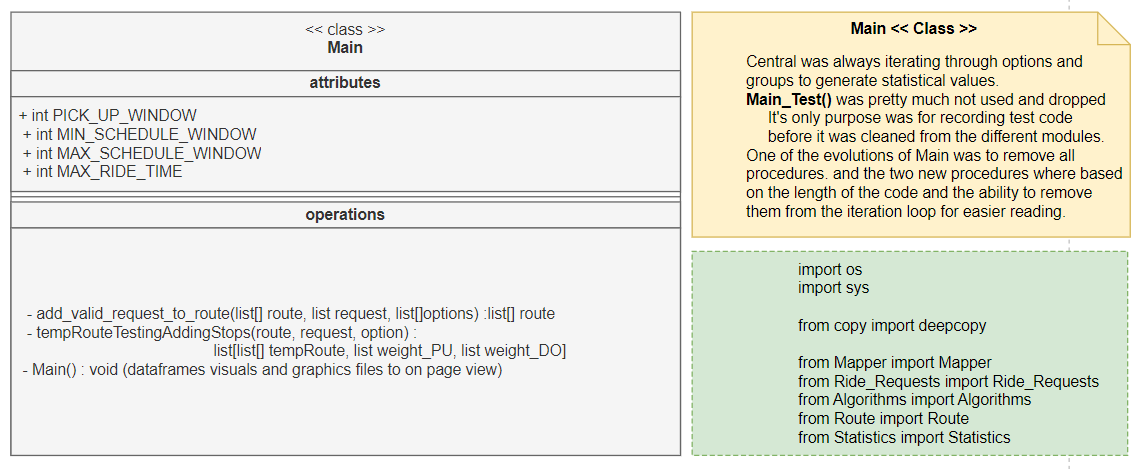
*Testing Concepts with Connections*

****

**Appendix B**

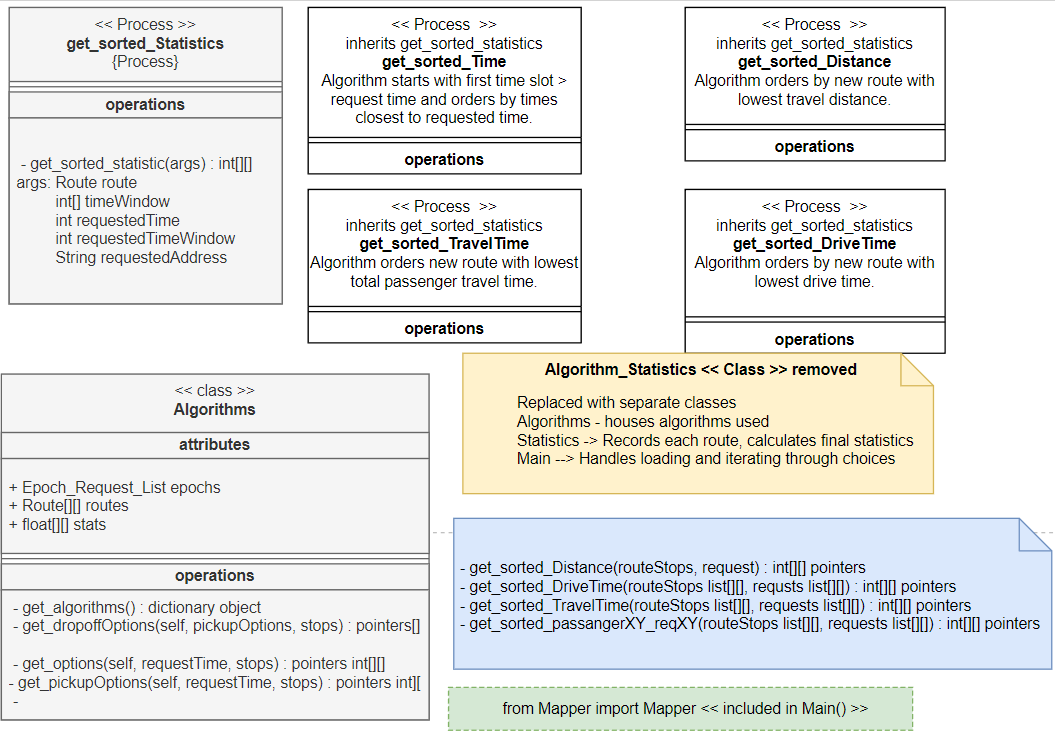
Proposed Timeline for Project (Diagram)

**Appendix C**

Class Diagrams (Main, Epoch\_Request\_ListRoute, Route)

**Appendix D**

Class Diagrams (Algorithms)

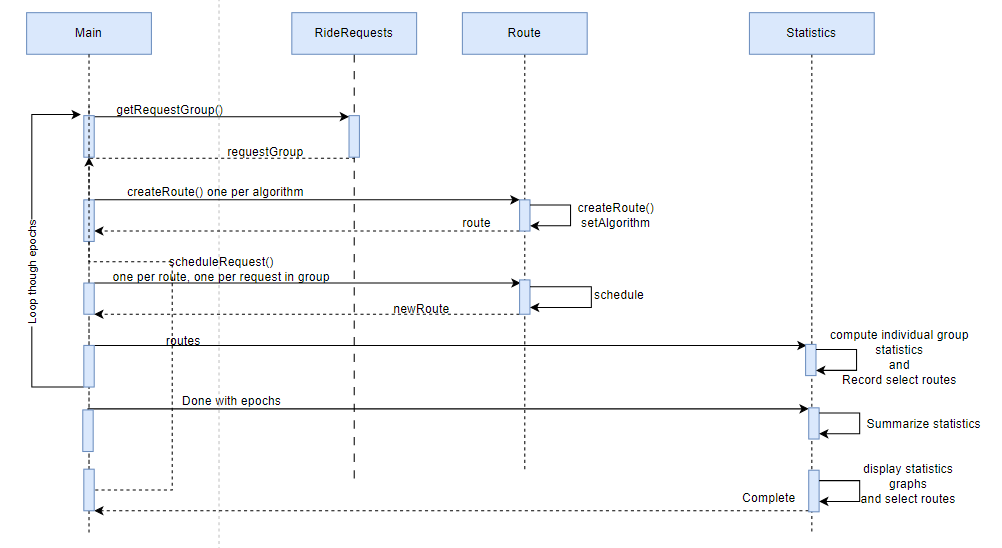


**Appendix E**

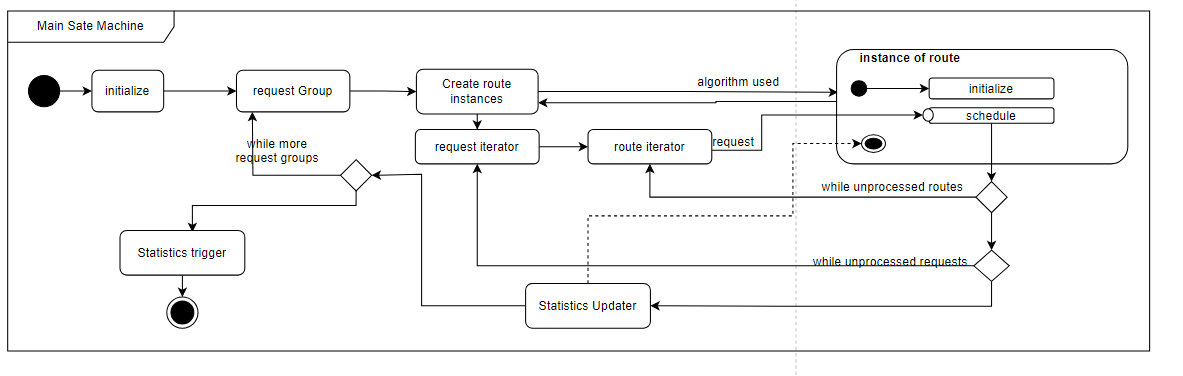
Interactive and State Machine Diagrams

**Figure 5**

*Interactive Diagram*

**Figure 6**

*State Machine Diagram*

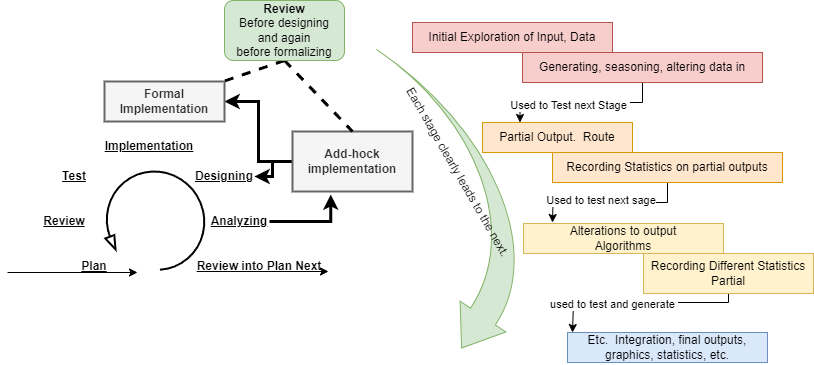


**Appendix F**

Proposed Research & Development Agile Development Module

The concept of adjusting an Agile development methodology, as shown below, to insert add-hock exploration implementation and design is depicted in Figure 7. This shows how it might look in this project as the project cycled to completion. The problem lies in both having and add-hock exploration and then making sure it is reviewed and finalized. For example, in exploring different statistics to track the routes, new questions arose. These different statistics are easy to develop add-hock and some modules do it automatically. This would allow a quick turn around time to test and print them. At this point the review is usually left out in the data science projects I’ve been a part of. The review of the implementation needs to occur to formally make sure they are needed as well as fit them in with the complete system design. Once the first review is done, the design and implementation is half done and needs additional outside review to make sure the design is valid and functional as well as another review to make sure the formal implementation is done correctly. In this project this did happen add-hock generating multiple reviews post module and cost more time then if that had been implemented and enforced during the production. Leaving it also left it out in figuring timelines and estimating code completion.

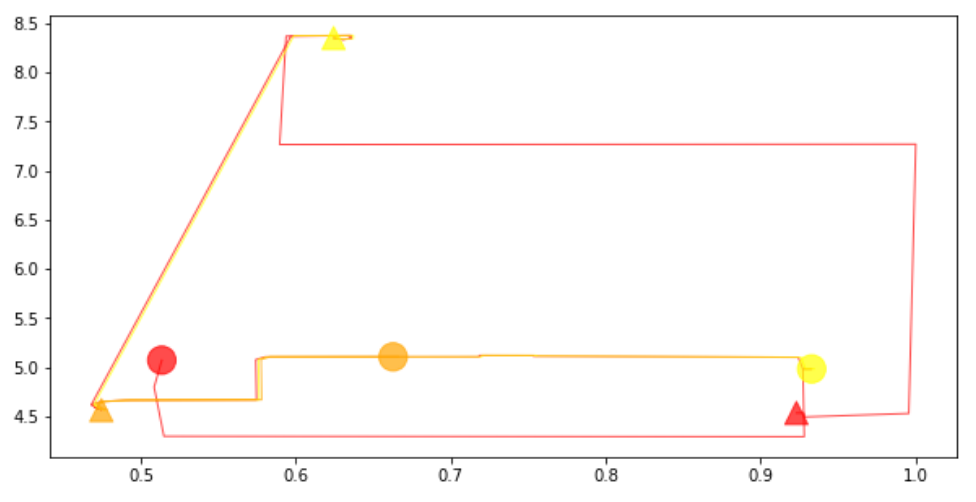
**Figure 7**

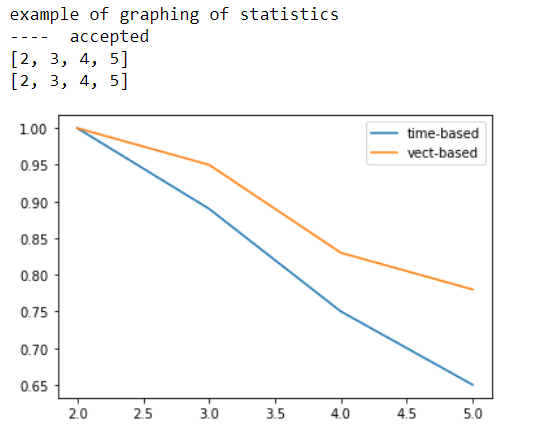


**Appendix G**

Graphic Outputs (Examples)

**Figure 8**

*Initial Example of Graphic Output*



**Figure 9**

*Example of Some of the Final Route Outputs and Request Validated*

