Table of Contents

Table of Figures i

Summary ii

Glossary iii

1. Introduction 1

1.1 History of the Linked Modernisms Project 1

1.2 Software Technologies 2

1.3 Features of the Linked Modernisms Project 3

2. Software Development Life Cycle 7

2.1 Iterative and Incremental 7

2.1.1 Requirements Phase 8

2.1.2 Analysis Phase 8

2.1.3 Design Phase 9

2.1.4 Implementation Phase 9

2.1.5 Test Phase 10

2.1.6 Iteration and Incremental 10

2.2 Extreme Programming 11

2.3 Pair programming 12

3. Website Security 13

4. Streamline Optimizations 14

5. Conclusion 14

6. Recommendations 15

References 16

# Table of Figures

Figure 1.1 Linked Modernisms Search Page with Visualizations Selections 4

Figure 1.2 Examples of Visualizations and Visualization of Degrees of Separation 5

Figure 2.1 Iterative and Incremental Software Development Life Cycle [4] 7

Figure 2.2 Extreme Programming Development Life Cycle [5] 12

# Summary

The Linked Modernisms Project desires the study of modernism to be a global phenomenon, by making modernist culture accessible to the humanities communities, and scholars around the world. The software development life cycle of this project was mostly an iterative and incremental life cycle. The beginning of development was staged in gathering requirements, and an analysis of those requirements by refining those details to prepare the project. Following those details, designs were constructed and implementation of those designs took place. Then once implementation was finished, testing would occur. The cycle would then start all over again as new challenges arose and new features were added to the project. Within these developing iterations, one of those added features were user contribution to the data. Therefore, with all forms, security issues arise as trusting user input is never safe and thus precautions needed to be taken, such as validation of user data input. With the web site functional and live, automation was a key requirement as the developers were finishing their contract and the project would need to be able to sustain itself. Therefore, makefiles were made along with bash scripts for addition and deletion of data in the database, and could be run whenever a significant amount of new data had been acquired from user contributions. With the project running smoothly in a pipeline, the project is able to start a new iteration of it’s life cycle, requirements, analysis, design, implementation, testing, and easily extensible for any possible new features that may want to be implemented.

# Glossary

MVC MVC stands for Model View Controller. This allows for a design where the Model stores the data. The View is the presentation, and what the users may be viewing of the software. Finally, the Controller is what generates the changes within a Model, which in turn would update the View for a user.

RDF Resource Description Framework. RDF is contained within the family of the World Wide Web Consortium (W3C), and is used within the project as a data format, or more specifically, a metadata model.

SPARQL SPARQL is a recursive acronym for SPARQL Protocol and RDF Query Language, thus being suited for querying and manipulating RDF supported databases.

Turtle Turtle is an RDF triple language format, where a triple is equivalent of a tuple that is of the order three, such as a 3-tuple. In this case a triple would be of the format subject-predicate-object. Thus, since Turtle is an RDF data model, Turtle data may be accessible via SPARQL queries.

VSP Virtuoso Server Pages (VSP) allows HTTP connections between the web page and the Virtuoso server. It is Virtuoso’s in house server side language created for the Virtuoso server [1].

# Introduction

The Linked Modernisms Project is a project that was started at the University of Victoria under the guidance of Associate Professor of English, Doctor Stephen Ross. The Linked Modernisms Project serves as an important research development within the field of digital humanities, as the project is on the cutting edge of development within the field. Linked Modernisms serves to provide the humanities community and scholars a software resource where users may query for data, and have reliable data presented in a visually appealing and useful manner.

## History of the Linked Modernisms Project

Primary development on the Linked Modernisms Project started in May 2015. The main goal for the project was to develop and provide a fully functional website with a corresponding database that was strongly selective in what type of data to include, as to have reliable and accurate data available for users. The implementation of the Linked Modernisms website before May 2015 was a bare skeletal structure for just displaying data that was gathered by a SurveyMonkey questionnaire from scholars within the humanities field and displayed their expertise. With the work that was shown from that implementation, it was decided to develop from scratch a new and improved website that would then implement the features the project had planned for its users. However, the technologies and work on the backend with the database was carried over, and was settled that the project would still use Virtuoso technology as the server, and would host our database. However, this separation of the backend and the frontend gave the advantage of looking at the project from a Model View Controller (MVC) perspective, which was ideal as the development team now consisted of two developers, and thus one could work on the backend while one would be working on the frontend, therefore allowing constant development and progress for the project.

## Software Technologies

As the project had settled on using Virtuoso as its server, it was most convenient for the project to use a SPARQL endpoint, as Virtuoso has full support for managing a SPARQL queries and requests to a database. The Virtuoso database that would hold RDF data was known as the Quad Store, and it accepted RDF formats of only RDF/XML, N-Triples (nt), and Turtle (ttl) formats of data into the Quad Store. Therefore, the backend of the project had been settled upon. However, with the frontend starting from scratch, the project was wide open to what kind of technologies would be used for the interface of the website. The resulting decision was determined with the goal to have a visually pleasing interface for the user. Along with basic HTML and CSS, which are needed as the basics for layout and styling web pages, JavaScript was used to control visual elements for the users to interact with. In particular, the D3sparql.js JavaScript library was used to complete the task of visualizations for the web site. Now with the backend and frontend technologies decided upon, the data still had to be dealt with. With the Virtuoso server only accepting certain formats, the project lead suggested the use of Turtle format, as it was easiest to read. However, pre-existing data from the survey questionnaire needed to be converted into Turtle. To overcome this obstacle, Python scripts were used to convert the existing data into Turtle, and this allowed for the data to be uploaded into the Virtuoso database. This left the project complete for components in development, however software was also needed and used frequently to manage progress and output of the project. Git version control software was used to track progression changes with the code base, while GitHub was used as a free hosting service for the project’s Git repository.

## Features of the Linked Modernisms Project

For the Linked Modernisms Project to reach its goals and full potential, a list of functionalities were set out as guidelines for the project at the start of development. Almost all features are fully functional and complete or have plans for improvement in the future and maintenance plans.

The feature with the top priority for the Linked Modernisms Project was to have RDF searchability. This would involve a user that would be able to query the projects database with the subject-predicate-object format that is used by RDF data. The user would then be able to filter searches for whatever they may want to specify for their search.

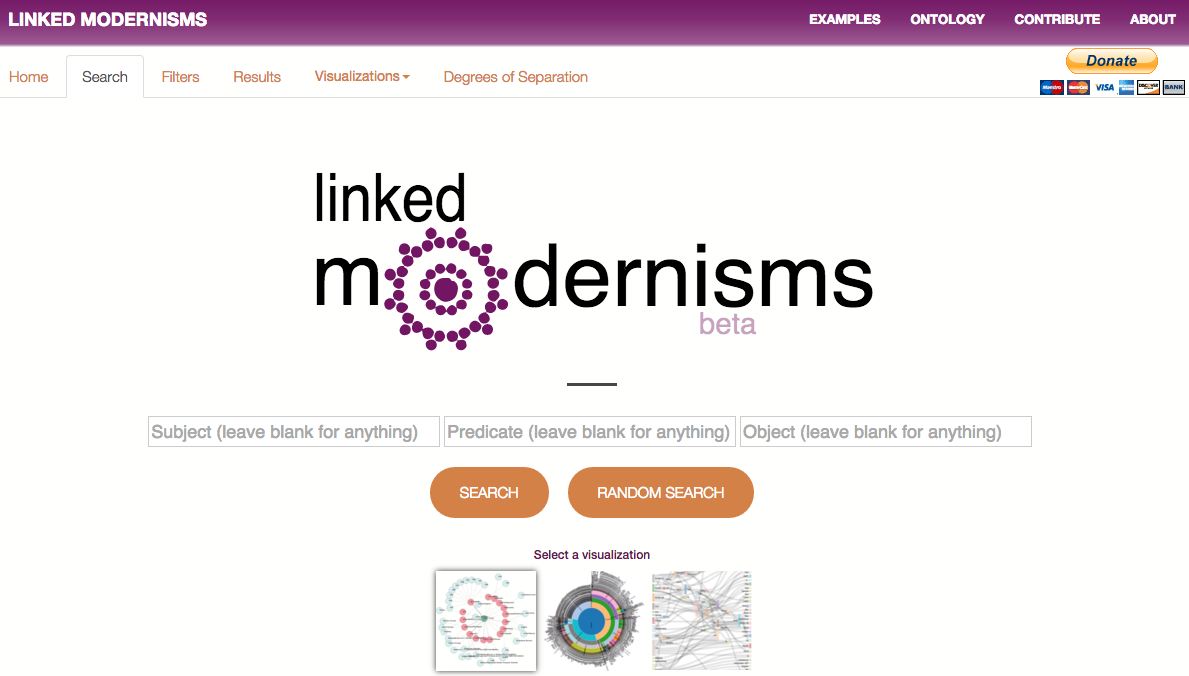


Figure 1.1 Linked Modernisms Search Page with Visualizations Selections

Visualizations of search results were the second priority for the project. This allowed the users to search for their results through search boxes and as a result of the search. The data returned would be one of the visualizations used from D3sparql.js, depending on the selection the user chose, whether it be a Force Graph, Sunburst, or a Sankey Graph [2].

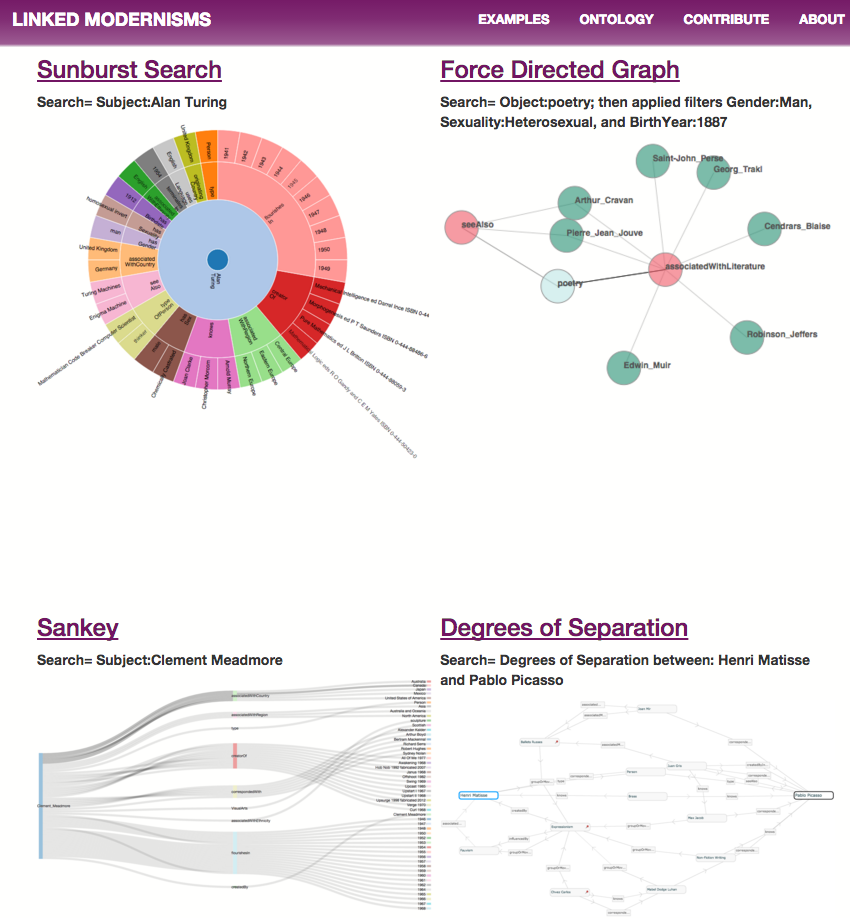


Figure 1.2 Examples of Visualizations and Visualization of Degrees of Separation

The third highest priority is similar to the visualizations, where instead of just a visualization of the results given, the degrees of separation were shown. Degrees of separation demonstrates and displays the waypoints between entities within the metadata, along with how many other connections may be on route while tracing connection paths to the other entity. An open-source project by the name of RelFinder was used to demonstrate and produce an interactive relationship graph that displayed these connections between different entities within our RDF data [3].

As the previous three features were implemented, the rest of the list of features was equivalent to each other in priority of importance. One of those features was to have a point where users could start with a randomizer function, such as a random button to generate random data, to display random results and demonstrate what the Linked Modernism Project was capable of producing for users. Saving and being able to re-access previous searches was another feature that was asked for. This was so users would not have to redo searches that they may have previously searched for, or that they may have forgotten what they searched for in the past, and could quickly access interesting search results they may have found using the website. This could lead to users wanting to have files of the Linked Modernisms data, which was also implemented as another feature that they would be able to download as a specific file format, the projects ontology data. However, users may not want our data, but may want to contribute to our data as well. This gave rise to another feature, a page where users could contribute to the project’s metadata. Then finally, the web site provides the features in an overall intuitive and quickly accessible interface design.

# Software Development Life Cycle

For software development to flourish, there needs to be planning with regard to any project. There also needs to be a controlled flow of how operations and work is managed and performed.

## 2.1 Iterative and Incremental

As development for the project was essentially starting from scratch, development had to start with a plan. As software development is difficult to work on continuously with no plan, an iterative and incremental approach had become the choice of how to plan and layout the project’s goals.

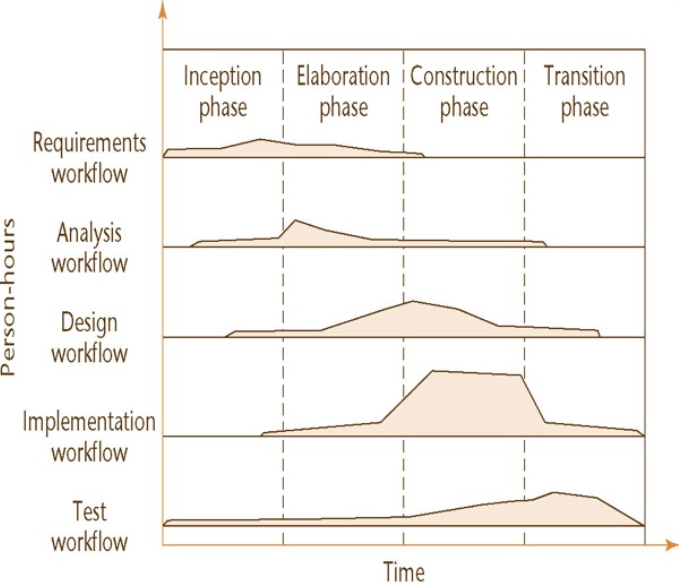


Figure 2.1 Iterative and Incremental Software Development Life Cycle [4]

### 2.1.1 Requirements Phase

As part of a Software Developer’s job, one of the first steps would be to find out what were the requirements of the client, in this case the Project Lead’s. The first step with gathering and identifying these requirements was to understand the difference between what the client wanted and asked for, and what the client was in need of for their project to succeed [4]. Requirements that were identified were from the features list given to the developers, along with understanding the purpose and business model of the Linked Modernisms Project. However, the business model was unlike a normal business, whereas this project is an open-source project and will be run and maintained through donations and support for the digital humanities.

### 2.1.2 Analysis Phase

The next step in the project was to completely analyze the requirements that were given by the Project Lead. Analyzing the requirements and refining them meant a clearer picture for the Project Lead, as the client must fully understand what the project plans are, as opposed to just what the client wants and believes is possible [4]. Therefore, when analyzing the requirements and summing up the results, all the requirements must be expressed to the Project Lead in natural language [4]. Especially in this case, the Project Lead was an English Professor and not well-versed in technical software terminology or any programming experience, thus careful plans needed to be descriptive and detailed so as an average computer user would be able to follow the guidelines.

### 2.1.3 Design Phase

Once most of the details from the analysis phase had been evaluated, the information and material gathered from the previous phases were used to form design choices. During the design phase, non-functional requirements are to be finalized for the project [4]. This included deciding on the main programming languages that would be used due to the requirements of the project, reusability of other software to improve development time, as well as portability issues between work stations and the server. Once the non-functional requirements had been decided, decisions on what visualizations would be well suited for the RDF triple data that we had was the next step by breaking the architectural design of the project up into modules based on the visualizations [4]. According to the preference of the Project Lead and functionality of the visualizations, the visualizations that were decided upon were a Force Graph, a Sunburst tree, or a Sankey graph [2].

### 2.1.4 Implementation Phase

As designs were complete or partially in place for implementation, target goals for the project were to start implementation. Therefore, actual programming and creation of features took place in this phase. As this was a large project to undertake, it is advisable to split large portions of work and partition them into subsystems, thus making the workload for developers easier to manage and maintain [4]. Implementation was reasonably split by what features were to be implemented, and were in the order of highest priority features to the lowest priority. Within the data visualization feature, implementation tasks were also split up between the different visualizations: Force Graph, Sunburst, and Sankey. Once the highest priority tasks were completed, it would continue in a chain downwards in priority until each task was completed.

### 2.1.5 Test Phase

The test phase is as important to developers as any of the other phases, as this determines if the product is implementing the features that were requested correctly. The testing phase serves as quality assurance for the project, therefore having traceability in mind when designing and implementing is an important requirement for testing [4]. With the project in a Git repository hosted on GitHub, concern for traceability was negligible as any changes in functionality or issues could be traced back through Git commits and Git messages of where the last working version of code was within the codebase. Therefore, any issues that occurred during testing were simple to trace through the last few versions of code to find any errors that occurred. Interface and superficial testing was done locally by running local servers on the development machines to see if user interaction with the interface and aesthetics were visually appealing and intuitive to the user. Interaction with the database and SPARQL queries were tested once the index of the website was uploaded onto the Virtuoso server. This allowed for tests of SPARQL queries and the accuracy of search results, and if visualizations were appearing correctly.

### 2.1.6 Iteration and Incremental

As processes started and work began to be developed, it was clear that it was never possible to cover absolutely every detail and thus retrospection into previously accomplished phases is required. An example of a set back was implementation of server side code. In the designing phase for it, PHP was originally going to be the programming language that was to be used for server side code. However, the version of PHP that Virtuoso supports is already a few years dated, so by recommendation from the Research Computing Specialist from Compute Canada the server side language VSP was used as it was fully supported and up to date with Virtuoso software. Therefore, progress in the implementation phase cycled back to the design phase by finding a better solution and continued back into the implementation phase. Unlike a waterfall development life cycle, an iteration and incremental cycle allows for dynamic improvement and will produce a more stable and thus a better product.

## 2.2 Extreme Programming

While an iterative and incremental life cycle was a big part of the project, an extreme programming style approach was also implemented into the workflow of the development team. Extreme programming urges and allowed for close interaction with the customer or client, in this case the Project Lead, and therefore stresses a case of customer satisfaction [5]. As the development team was only two people, meetings and communication were held frequently between the developers and the Project Lead which allowed for honest plans, simplicity, and most importantly the feedback from the Project Lead on what their thoughts were [5]. This gave the chance to implement a feature or module, get fast direct feedback on it, and be able to reiterate and improve on any issues or features in a short amount of time, therefore creating a fast pace and precise software production environment.

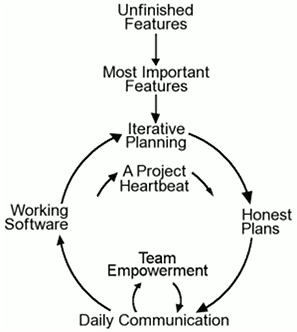


Figure 2.2 Extreme Programming Development Life Cycle [5]

## 2.3 Pair programming

Due to the development team being only a two-man team, this gave the advantage of each knowing the functionality of almost the entire codebase. Along with the extreme programming philosophy, with the stress on communication and interaction with the Project Lead, the high communication between the developers was a great advantage. Pair programming can increase quality in the software without hindering time delivery of the software. It seems counter intuitive, but can result in higher quality code [6]. Instances of this during the Linked Modernisms Project were when solving for issues and tracing back through codebase. Searching for errors together is quicker, whereas one developer may have missed errors. Therefore, once the errors are found, both developers may resume their work and continue on, instead of having one developer work while the other could have spent double the time still searching for an error. On the opposite side of development, new software to both developers was introduced, and therefore pair programming was useful so as to teach each other what the other did not know or had not learned yet.

# Website Security

Developing the interface and functionality of the Linked Modernisms Project were not the only concerns, especially once functionality of user contribution was introduced. This meant that users could input data into the web pages, and according to many sources including Microsoft; there are two golden rules about network programming. One, never to trust a user’s input; and two, while the data is moving between an untrusted to a trusted domain, always confirm that the data is not malicious [7]. Well known cases were especially protected against, as users may input special characters into the form data, or even write code such as </form> which would allow the user to inject their own malicious code and have it executed [8]. Consideration was taken to how significant the security should have to be that would be implemented. After consideration, a script that prevents any password saving to any file was selected as the security to use [9]. The security measure is called GateKeeper 2. The theory behind GateKeeper 2 is that the page is accessible only if you know the page name, which is the password that is chosen [9]. However, that file is not shown in any developer tools that are in browsers, and therefore cannot been seen unless correct input is submitted.

# Streamline Optimizations

With the Linked Modernisms features implemented and both developers’ Co-op ending, the time for optimizing the existing features had come. As the Project Lead was not well-versed in the software side of the project, a software pipeline was needed for the Project Lead to continue with the project while not having any developers to do any work. Optimizations to features had to be introduced. Trimming of unused code was done so as to keep the code readable to any possible future hire whose job would be to maintain parts of the web site. With the features completed, bash scripts were made to automatically run processes that would filter user contributions data and input them into the database along with validating the data at the same time. A makefile was also created so that all the addition bash scripts could run with just the “make” command, and would run all the processes and scripts in one quick command. There were also deletion scripts with a different makefile that allowed for quick and easy deletion that ran similarly. This allowed for the only maintenance of the project to be handled with the one “make” command, which will streamline the pipeline and create little to no issue for the Project Lead.

# Conclusion

The software life cycle of the Linked Modernisms Project was described. This demonstrated the difficulties and obstacles that occur in software development and the solutions and programming styles that help avoid issues and provide better quality software in the end. Requirements, analysis, design, implementation, and testing phases have all been iterated through while in development of this project and as a result, each iteration of the website and it’s features improved over the previous iterations implementations. Following these iterations allowed for implementing new feature requests from the Project Lead throughout development and the necessary security to sections of the web site. With the web site now running through a pipeline, this allows the web site to be extensible and have any future requests or changes easily implemented and have the web site continue to run smoothly and serve the digital humanities community.

# Recommendations

There is work that remains with the natural language processing. Since the natural language is dependent on the StanfordCoreNLP, it requires Java for the project. There is currently no streamlined section in the pipeline for any of the natural language code. Automation for natural language, processing data, and uploading it to the database needs to be implemented. A method of not having to manually run the “make” command to run the addition bash scripts for the data should also be implemented for a more streamline pipeline. With the interface, improvements could be worked on with the CSS styles and the formatting of the images of project contributors and supporters on the main page. Finally, the only major issue would be the rate at which data is uploaded to the database is too slow with the verification of data needed before adding to the database.

# References

[1] *14. Web Application Development* (2016, Jan. 2). [Online]. Available: <http://docs.openlinksw.com/virtuoso/vsp1.html>

[2] *d3sparql.js* (2015, Dec 28). [Online]. Available: <http://biohackathon.org/d3sparql/>

[3] *RelFinder – Visual Data Web* (2015, Dec 18). [Online]. Available: <http://www.visualdataweb.org/relfinder.php>

[4] *03\_UP\_Workflows.pdf – Google Drive* (2016, Jan 1) [Online]. Available: <https://drive.google.com/file/d/0B0Z3739uxD0jVllBZGRvZ1RKSXc/view?usp=sharing>

[5] D. Wells. (2013, Oct 8). *Extreme Programming: A gentle introduction* [Online]. Available: <http://www.extremeprogramming.org/>

[6] D. Wells. (2013, Oct 8). *Pair Programming* [Online]. Available: <http://www.extremeprogramming.org/rules/pair.html>

[7] Microsoft. (2016, Jan 1). *Do Not Trust User Input Directly* [Online]. Available: <https://msdn.microsoft.com/en-us/library/ee798441(v=cs.20).aspx>

[8] *PHP 5 Form Validation* (2016, Jan 1) [Online]. Available: <http://www.w3schools.com/php/php_form_validation.asp>

[9] *GateKeeper 2 – Javascript Password Protection* (2015, Dec 18) [Online]. Available: <http://www.htmliseasy.com/keeper/index2.html>