SIYAVULA-BONA Mind-maps

CAPSTONE PROJECT

## Abstract

First you should have an executive summary (or abstract) just a single paragraph saying what the results of the project are (at most 200 words).

Please note in this document the words such as Node(Nodes) and Vertex(Vertices) will be used interchangeably.

# Introduction

Your introduction provides the context for the project and should contain the statement of the scope of the project (which may have changed since you first wrote it). Someone reading your introduction must have clear idea of what the system is intended for. If you think there is something special about the kind of problem you tackled that your reader needs to know up front then this is where you say it.

If you need any survey of other work (you probably don’t) then put it towards the end of the introduction and give suitable references. A case where this is needed is if your project builds on someone else’s project or some published algorithm.

Discuss your approach to solving the problem. Give a short overview of the software engineering methods you used (e.g., traditional analysis followed by design and implementation, typically the case if you did an evolutionary prototype, or a more agile approach where you had a cyclical development process).

The Siyavula project as we named it is a software application that aids our clients at Siyavula to interactively and more efficiently view their concept maps. The idea behind a concept map is that it provides a visual representation of different courses and subjects that they use to teach students. The maps contain dependencies, nodes, and a flow structures that aid in the teaching and learning of various concepts in different areas of knowledge ranging from Chemistry and Physics to Finance etc.  
  
The initial problem with their initial system more so was with having a static, manually created system that unfortunately as the concepts map grew. It became increasingly difficult for them to maintain a comprehensive over-arching visual representation of the map that was easy to read/use but at the same time was efficient. Visualization lies at the crux of this project as the primary goal is to be able to see the concept maps its components and how its components interact with each other.  
  
With regards to solving the problem we implemented an Agile based Software Engineering approach to being able to create the software. The development team was built of 3 members. Each split into 3 roles, the team leader, the communicator and the ...  
Taiga.io a free project management software tool was used extensively. It allowed us to create various user stories that would better enable us to interact as a team and gave us a platform to be able to generate issues, enhancements and additions to the code that was based solely off of the requirements that the client had initially stated(see link). Taiga also allowed us to generate Sprints that would allow us to use Burn-down charts to monitor and track our progress with regards to the backlog and monitor our velocity. The software also provides taskboards that enabled us to see who is doing what and at what stage each of the tasks are in. We decided to also implement the use of version control software such as git. We chose the github platform to be able ot better interact with ourselves as well as the fact github is well integrated with taiga they both provided seamless software engineering capabilities when coupled with one another.

Our approach to the creation of the working solution involved continual client interaction with both supervisor and immediate stakeholders.

# Requirements Captured

The next section deals with the analysis of your system. Cover the functional, non-functional and usability requirements. This is where you present your use case narratives and diagrams.

## Functional Requirements

Non-functional Requirements

Accessibility

The system is an open system and is therefore accessible to anyone who owns it and knows how to operate the system. Majority of the access restriction is provided by the client

Capacity, current and forecast

Compliance

Documentation

Disaster recovery

Efficiency

The software should also implement the map file and display it within 3s. All the data within the respective map files needs to be visualized within the 3s gap

Effectiveness

Extensibility

Fault tolerance

Interoperability

Maintainability

Privacy

Portability

The software si written in java. For it to work the computer wll need to have Java installed.

Quality

Reliability

The software should be reliable in that it should provide the data

Resilience

Response time

Input of map file:0.05s

Display of Graph: 1.2s

Graph Processing: 0.300s

Robustness

Scalability

With regards to the size of the map files, a map file of size xxx should be the upper limit with regards to size achieving the performance stated before.

Security

Stability

Supportability

Testability

### Use Case Narrative & Diagrams

Discuss the major analysis artefacts that you produced. We will expect you to produce at least one overall description of the architecture used in your system as a diagram, either here or below (see Section ). You may also want to include an analysis class hierarchy diagram.

# Design Overview

The next section is an overview of your design. The system design has to be justified in terms of the expected behaviour of the final product.

If you produced a design class diagram put it here.

You must present the overall architecture of the system together with an architecture diagram. You may choose what kind of diagram best suits your project but we would expect a layered architecture diagram (see Figure 1) unless there is a good reason for some other kind of diagram. It need not be a formal UML diagram as long as it conveys all the necessary information clearly.

You should then (in subsections) cover the algorithms and the data organisation used and why they were considered the best.

With regards to the requirements stated earlier the the visualization of the system is an integral part of the system. In addition due to the massive data set the system should also be relatively efficient due to the fact the system is constantly growing and the data currently at hand is particularly massive. Therefore with regards to the design of the system careful attention must be paid on optimizing how fast the data from the .map files are parsed and how fast the graph can be constructed.

The final product should allow the users to visualize the graphs effortlessly, zoom in and out of certain areas in the graph as well as scroll left and right, up and down depending on their preferences. The user should in addition be able to load hte map od their choice and have it displayed .

From a visualization perspective, we realize that just displaying the vertices and their dependencies is not enought to attain a satisfatory out come from the project. Due to the complexity of the interacitons between vertices, edges and groups. Different visualization techniques will be employed to ensure accurate information is being presented at all times.

## Visualization Techniques

### Dynamic Positioning of Vertices and Edges

Edges and Vertices will be nonstatic and can be moved around if need be. This feature will be important in that will allow the user to arrange nodes and edges in manners that may make the most sense and spur learning.

### Shading

If certain nodes intersect or have a mutual relationship then shading will allow for the node to display characteristics associated with the two relationships.

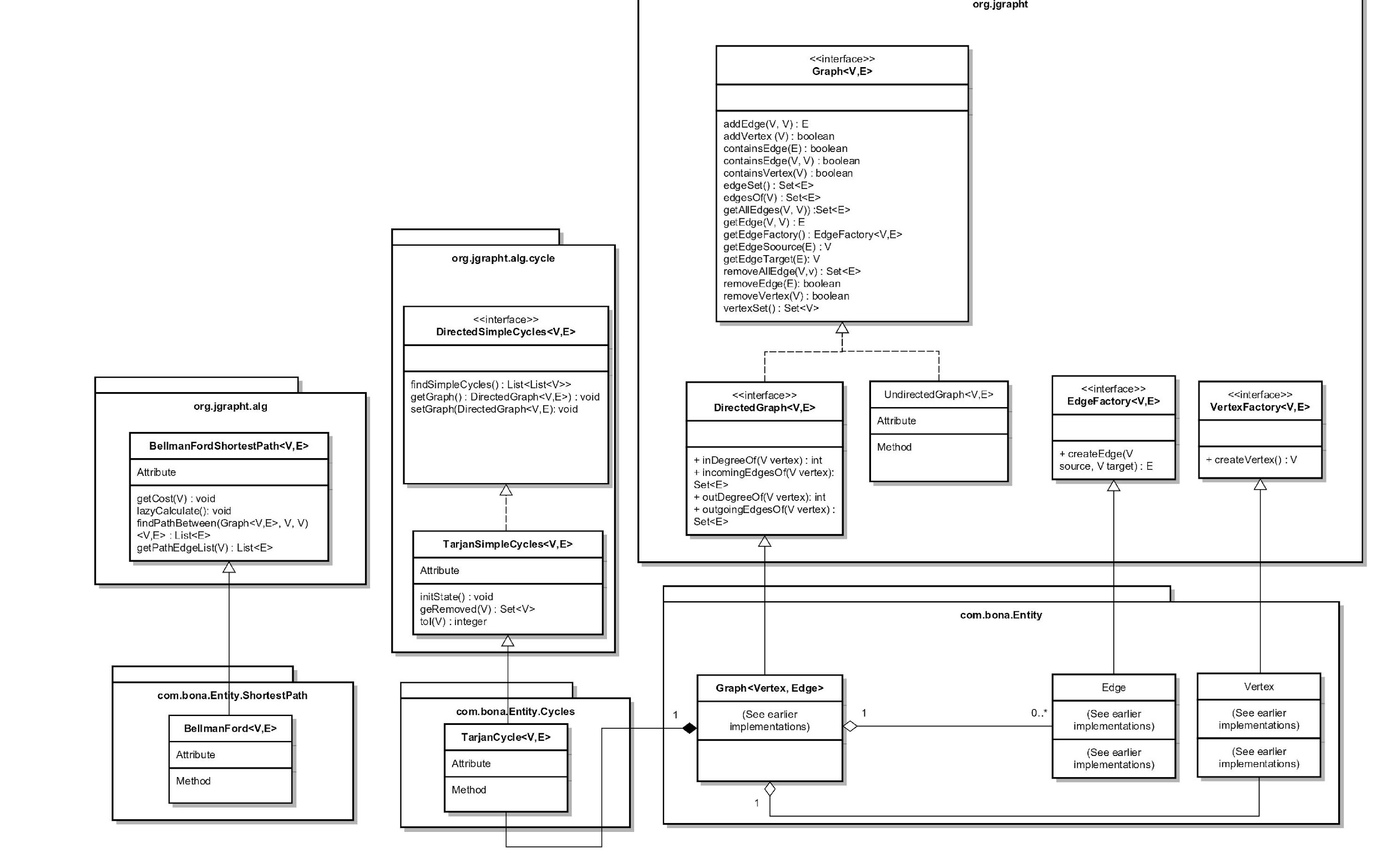
### Different Colors

Different groups and vertices that have different interdependencies or relationships may be highlighted or presented in different colours. These colours will allow the user to better distinguish either different groups, subject areas or types of nodes.

### Color Intensity

Different color intensities may be used to show children of a particular vertex. The deeper down the relationship, the less the color intensity.

### Highlighting Edges (Children)



## Internal Architecture

(Design Class Diagram)

Figure (xxx) represents an illustration of the design structure within the Siyavula System.  
  
The choice to opt for using JGrapht an open source graphing library proved to be a great one. JGrapht provides access to several graphing components from algorithms, to cycles to shortest path algorithms as well as a variety of graph types. E.g. Directed/Undirected Graphs. Furthermore majority of the architecture was stabilized around generic abstract classes or interfaces. These give us full control over the implementations that we would like. E.g. interface DirectedGraph<V,E> allowed us to implement the general directed graph implementations as well as define the V & E classes the way we would like to.

## Overall Architecture(Package Diagram)

## Algorithms

### Graphing Algorithms

#### Vertex-Edge Relationship Mapper

### Cycles

#### Tarjan Cycles

Tarjan Cycles are the cycle method that we chose to implement. They form the basis of the cycles we chose to implement and are one of the several cycles offered by the JGrapht open source library. The algorithm behind Tarjan Cycles makes them light-weight. According to (xxx) they are an improved version of Kosaraju’s algorithm They perform with a worst case O(|V|+ |E|) where V and E represent vertices and edges respectively  
Tarjan Cycles.

### Search Algorithms

#### Parallel Sequential Search

Due to the potentially massive size of the map file that will be imported. Several hundred or thousands of nodes may be required to be created. Our graph data-structure houses two lists that contain all the vertices and their associated edges. When it comes to checking if a vertex or edge exists, those lists need to be traversed. The lists are not in order and therefore binary search is not helpful here, furthermore the overhead involved in sorting these lists then implementing binary search is far greater than that of having to search them sequentially. Although not the best option, by implementing the runnable interface and using parallelism and the fork-join architecture to allow us to search the tree a lot faster.

it can be seen as an improved version of [Kosaraju's algorithm](https://en.wikipedia.org/wiki/Kosaraju%27s_algorithm), and is comparable in efficiency to the[path-based strong component algorithm](https://en.wikipedia.org/wiki/Path-based_strong_component_algorithm). Tarjan's Algorithm is named for its discoverer, [Robert Tarjan](https://en.wikipedia.org/wiki/Robert_Tarjan).[[1]](https://en.wikipedia.org/wiki/Tarjan%27s_strongly_connected_components_algorithm#cite_note-1)

According to … Tarjan Cycles take

Shortest Path Algorithm  
Bellman-Ford Algorithm is a … algorithm. Its implementation within the software primaroly served the purpose of locating the shortest paths between selected nodes

# Implementation

Now we get to the details.

* Describe your data structures and be sure to illustrate them with a diagram.
* If your user interface was a key feature describe how that was implemented.
* Discuss the function of the most significant methods in each class. This may well require flowcharts, or sequence diagrams, in some cases.
* Any special relationship between the classes (e.g. friends) and why they exist.
* A description of any special programming techniques or libraries used.

The primary data structure used within the implementation of the Siyavula Concept Map application is the graph. In fact the project revolved almost entirely off of graph theory and how graphs may be used to illustrate relationships between various Vertices and Edges.   
  
Special consideration was taken with regards to the visualization of the vertices and their dependencies. A variety of visualization techniques such as colors, shading, shapes and textures were implemented to allow the user to uniquely identify vertices, edges and groups with similar or differing characteristics.

**Data Structures**  
As shown in figure … The open source JGrapht Library provided us with the abstract classes and interfaces that we would implement to get the graph working properly. The DirectedGraph<V,E> class implements the Graph<V,E> interface that provides the underlying skeleton for the development for the graph that we used. Both these classes were housed in the com.jgrapht.core package. Our graph implemented the DirectedGraph<V,E> interface which was contained the necessary methods that we needed. Our clients requirements stated that the user should be able to view the dependencies and links between nodes and show the flow of information. Therefore the choice of using a DirectedGraph corroborated this. The com.bona.Entity.Graph class which we created contained a variety of methods.

addVertex(String id, String type,String label)

## User Interface

## Methods and Functions

## Open Source Libraries

### JGrapht

The JGrapht Open Source Library as mentioned previously in this report boasts a comprehensive set o fgraphing tools from shortest path algorithms to cycle algorithms to various type of graph structures such as Directed, Listenable Undirected Graphs. A major benefit of using this library is the fact that it incorporated many abstract classes and interfaces that allowed for effortless extensibility as well as customization. In addition a lot of these classes are generic allowing for further customization of the classes.

License

### VL-Jung

VL-Jung is an open library that facilitates majority of the UI functionality. This library incorporates several visualization techniques for visualizing graphs.

License

License

# Program Validation and Verification

Tell us how you tested the system and why you believe it works. Describe the Quality Management Plan for your project, that is, software testing plan. The plan should indicate the types of testing that was performed and detail how they were done. This must include the reasons on why the chosen testing protocol was considered effective.

Create a table that summarizes the testing plan (see Table 1).

*Table 1: Summary Testing Plan. A table caption goes above the table.*

|  |  |
| --- | --- |
| **Process** | **Technique** |
| 1. Class Testing: test methods and state behaviour of classes | Random, Partition and White-Box Tests |
| 1. Integration Testing: test the interaction of sets of classes | Random and Behavioural Testing |
| 1. Validation Testing: test whether customer requirements are satisfied | Use-case based black box and Acceptance tests |
| 1. System Testing: test the behaviour of the system as part of a larger environment | Recovery, security, stress and performance tests |

Describe all the steps taken to validate the correctness of the program.

If you had user tests then say what you did and what the results were. Describe why these test data were chosen (what test conditions the data was testing). Table 2 provides an example of the sorts of results we are looking for. The full detail of the test runs should be appended to the report.

*Table 2: Summary of tests carried out. A table caption goes above the table.*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Data Set and reason for its choice** | **Test Cases** | | |  |  |
|  | *Normal Functioning* | *Extreme boundary cases* | *Invalid Data (program should not crash)* |
| Preliminary test (see Appendix 3) | Passed | n/a | Fell over |
|  |  |  |  |
|  |  |  |  |

## Follow your table of results with a discussions of them highlighting how useful and usable your system is for its intended purpose.Conclusion

Your report must have a clear conclusion where you revisit the aims set out in the beginning and discuss how well you met them. Did you achieve the objective of creating a well-structured, modular, and robust system? Please summarize the design features and test results that show this.

## User Manual

Your system must have a user manual. Append this to your report (make it Appendix A) or bind it separately if it is big. If your system is interactive and has a good user interface with context dependent help then this can be just a cheat sheet. Discuss the level at which your user manual is to be pitched with your client. If your system is to be extended then you might want to include a technical API manual.

# Conclusion

This document has covered the major sections needed for your report. You will probably have each of the subsections 2.1–2.7 as major section in the report each with its own subsections.

A marking guide for the report will be provided later.

# Appendix A — Code Legibility and Output

This is not strictly part of the report but is a requirement for the final hand-in.

* Each method should start wide a brief description of its function.
* Use indentation to display the structure within a method.
* Comments should be used extensively. They are best used to describe logical blocks of code rather than individual statements. Line-by-line comments have the drawbacks of not providing any overview and of decreasing readability.
* Meaningful identifiers should be chosen.
* Output should be pleasingly formatted and easy to read.

# References

If you need to give references ensure that the following information is included:

Journal article: Author’s surname, Author’s initial. (Year of publication) Title of paper. Title of journal, volume number, page numbers.

Book: Author’s surname, author’s initial. (Year of publication) Title of book. Publisher, publisher location.

Chapter: Author’s surname, author’s initial. (Year of publication) Title of chapter. In editors (eds), Title of book. Publisher, publisher location.

Conference: Author’s surname, author’s initial. (Year of publication) Title of paper. Title of Conference, conference location, conference date, page numbers. Publisher, publisher location.

Technical documents: Document number (Year of publication) Document title. Publisher, publisher location.

Internet source: Author’s surname, author’s initial. (Year of publication) Title of source. url (retrieved date).