Table of Contents

Introduction

Wikipedia is at a crossroads; with over 4,985,000 articles on the main English site[1], the focus has shifted from expanding the breadth of its coverage and onto ensuring that it is accessible to all, regardless of language. This continues the philosophy of the non-profit Wikimedia Foundation who run Wikipedia with the aim of creating a "world in which every single person on the planet is given free access to the sum of all human knowledge."[2] Hence, the multilingual expansion of Wikipedia reflects the Foundation's ethical values that information should be freely available to all. To ensure this, the Wikimedia Foundation recently began Wikidata: a project to centralise the information across Wikipedia in a queryable database that can be accessed by both humans and computers. For example, the article for Auckland on the English Wikipedia notes that, "Auckland has a population of 1,413,700…"[3] while the German Wikipedia includes "Der Ballungsraum Auckland ...1,4 Millionen Einwohnern."[4] Figure 1 shows that under Wikidata, the assertion that Auckland has a population of 1.4 million people is made once in a database, allowing this information to be viewed in a many languages without further editing. Hence it serves to fulfil the Wikimedia Foundation's goal of making information accessible to all.

01 The Wikidata Approach to Storing Population

Source: *screenshot of Wikidata*

Wikidata also fulfils another strategic goal of the Wikimedia Foundation: consolidating data in a way that makes information dynamic and more broadly useable. For example, the addition of "{{#property:P1082}}" to the source of the Auckland Wikipedia article causes its population to be added to the text of the article when viewed by the reader. This centralisation means that a single change on the Wikidata page causes the population of Auckland to be updated across all 96 'Auckland' pages in the various languages that Wikipedia supports. This synchronises all of these articles, which is an essential way to ensure that even the sparsely edited articles of minor languages are kept up to date. The importance of this was noted by a recent paper in the JMIR eHealth journal which described how Wikidata could be used as a central repository for information on Drug-Drug interactions, which are essential to preventing potentially lethal side-effects. Since impoverished regions often lacked this information, I feel that this represents a valuable way to provide this life-saving information to third-world doctors in their native language.[5] While updates to Wikipedia articles are typically made by deleting the old text and replacing it with the updated figure, Wikidata can retain previous values, leading allowing for situations like that shown in Figure 2 where change over time can be seen. Due to its recent founding, Wikidata has only a limited record of how this value has changed but as the project expands, so will the information it holds. This chronological record makes it easier for vandalism and incorrect values to be found while also providing a broader snapshot of the information, hence furthering the Foundation's goal of representing the totality of human knowledge.

02 Population of the U.S. over time (via Wikidata)

Source: *screenshot of Wikidata*

Besides making data accessible to all humans, another key consideration of Wikidata was that its strictly-structured dataset would also be machine readable. In this way, the Wikidata project is part of the broader technological trend towards a "semantic web" in which data within a webpage is made queryable, allowing it to be dynamically linked both within and between websites. For example, a library website could integrate information from Wikipedia (via Wikidata) and Amazon to give patrons the ability to find out more about the book and to get an estimated purchase price and delivery date if they wish to own it. In this way, we see how an implementation of the semantic web allows the library to offer its users more relevant data than they themselves hold. It therefore shows how the semantic web ties in with the Wikimedia Foundation's philosophy that information is is most useful when it is not in isolation.

This idea of a semantic web ties into what I believe is one of the most fundamentally important trends in modern technology: the transition from Web 2.0 to Web 3.0. My aforementioned library example highlights the difference between these models. The web began with Web 1.0 in which static pages were delivered from a server, providing only limited interactivity to the user. For example, a library page may have an alphabetical directory of books, each linking to a page containing general information about that book. Web 2.0 marks the proliferation of user interactivity, leading to more user-driven sites such as social media sites and YouTube. Continuing my example of a library website, under Web 2.0 users could be able to leave book reviews and tag books with keywords. In this way the information that the website shows is dynamic and interactive but this information isn't personalised or integrated with other sources. Web 3.0 is based on the semantic web and so will allow sites to pool from a diverse range of data sources. In addition to the above features of the library site, a Web 3.0 version could use Wikidata to return search suggestions for books by the same author and access the user's Facebook profile to access books that they have already reads that similar books can be recommended. This highlights that the power of Web 3.0 and the semantic web lies in its ability to access information from a range of sources to give more relevant and personalised user experiences. In this way, I feel that the most important distinction between these model of web development lies in the second row of Figure 3: 'content'. In Web 1.0 and Web 2.0, the onus is on the web developer to host and manage their own data, limiting the scope that it can cover and exponentially increasing cost as new datasets are developed. Conversely, under a Web 3.0 model the developer acts in a large part as a 'curator': they merely interconnect a unique collection of data sources in a meaningful way. This allows web developers and users to rapidly integrate diverse sets of information, a concept termed 'data fluidity.' Note that Figure 3 lists RDF as a core technology of Web 3.0. This will further explored later in my report. This report will also make frequent reference to SVG. SVG, or 'Scalable Vector Graphic' is an open XML based file format that is used across Wikipedia to store vector images that are resolution independent.

03 **Trends in the Development of the Internet**

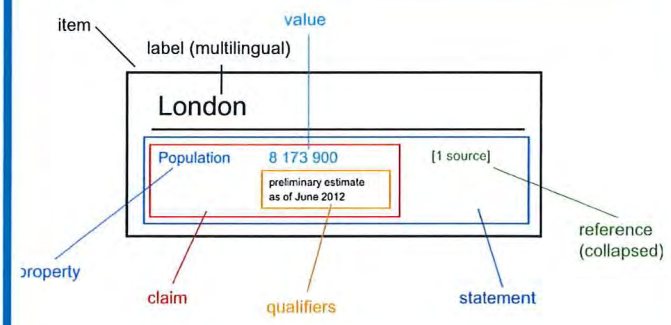
	Web 1.0	Web 2.0	Web 3.0
Information	Static	Interactive	Portable & Personal
Content	Owned	Shared	Curated
Interaction	Web Forms	Web Applications	Smart Apps
Search	Directories	Keywords / Tags	Context/Relevance
Research	Britannica Online	Wikipedia	The Semantic Web
Technologies	HTML / FTP	Flash/HTML5	RDF/RDFS/OWL

Source: based on https://cayugahospitality.com/review/b2b-social-computing

Throughout this report, I will use various terms related specifically to Wikipedia and Wikidata; some of these will be explained in this section. Firstly, Wikipedia articles are written in a markup language called 'wikitext' or 'wiki markup.' Editors use this to specify the links, images and templates etc. that

the reader will see.[6] Wikitext is generally static but there is limited support for scripting within pages using the Lua scripting language.[7] Bots are scripts that run on either an individual's computer or Wikimedia Foundation servers. They require special permission to run, allowing them to make automated edits to Wikidata/Wikipedia. Wikiprojects are projects where groups of editors collaborate on specific tasks such as to improving articles of a particular topic/region. Wikidata represents all conceivable objects and concepts as *Items*. Items are referenced by their *entity ID*, which is a unique number prefixed with "Q". Each item corresponds to a tangible object so the British and Canadian cities both called London would have different items whereas the German and English Wikipedia pages for Auckland both point to the same item (Q37100). Hence it is important that items are multilingual. This is fulfilled through the *Label* (aka *name*) and *Description* which are stored as strings for each language that Wikidata supports. As shown in Figure 4, information is contained within an Item as a series of *Statements*. Statements contain a *Claim* and optionally a source to reference that claim. Claims act as an assertion that a specific *Property* of the item has a particular *Value*. A large but discrete set of properties are used, with each uniquely having an entity id prefixed with "P." Datatypes that can be used for values include numbers, strings, coordinates and dates. Other items can also be used as values such as in the New Zealand item where the P36 (capital) property has a value of Q23661 (the entity id of Wellington). As shown in the Figures 1 and 2, the user interface for Wikidata displays the titles of the items/properties in the user's language, not their entity id's, for the sake of readability. However due to their uniqueness and immutably, entity id's are used for behind the scenes for data storage and in machine accessible interfaces. The use of *Qualifiers* can also be seen in Figures 1 and 2 where they provide metadata about the claim i.e when the value was found and how it was determined.[8]

04 **Key Features of a Wikidata statement**



Source: modified from http://www.jmir.org/2015/5/e110/

Issue

As an active member of the WikiProject Maps on Wikipedia, I am often involved with creating maps to accompany articles. For example, Figure 5 shows a map that I made for the China Southern Airlines page, showing the countries it flies to. It is during the course of making this maps that I begin to identify several key issues. Firstly, these maps were all constructed by hand by individually shading countries in an SVG template using Adobe Illustrator. This process was very time-consuming, with each map taking me on average an hour to make. This manual nature also lead to frequent errors that were difficult to find. For example, in one case I had accidentally shaded Myanmar instead of Thailand on a map that appeared on a highly-visited page for over 7 months. In this way, the current approach conflicts with the philosophy of the Wikimedia Foundation, who "want Wikipedia to be a

05 **Map Manually Created by Myself for Wikipedia**

Source: self

reliable resource" by not only correcting errors but also implementing mechanisms to prevent them.[9,10] However, an arguably more serious issue with the current method is that it contradicts one of the founding principles of the Wikimedia projects: the idea of ensuring "the ability of almost anyone to edit." Instead, a highly-specific set of skills that are possessed by few editors are required. This has led to a situation in which the maps soon became out of date, with editors often sending me messages with corrections that had to be made. In effect, this effectively lead to a single editor having control over a key aspect of many articles. This situation is heavily frowned upon by Wikipedia as it contradicts their values of the "democratisation of information" while also meaning that many maps became inaccurate during periods of time where I was unable to find time to maintain them. In reality, a few other editors were able to produce similar maps albeit with differs approaches and thus different outcomes. As shown in Figure 6 (counties served by Air Astana), this detracts from stylistic consistency, a violation of Wikipedia's policies.[11] It also detracts from functional value. For example, the different map projections used in Figure 6 means that it can not be directly overlaid Figure 5 to make comparisons. Another key issue with the current approach was highlighted by the founding of South Sudan in 2011: over 2600 maps had to individually remade to accomodate its new borders.[12,13]

06 **A similar but Inconsistent Map**

Source: *created by user 'Zebra848'*

Based on the above analysis, it seems clear that the status quo of constructing world maps for Wikipedia by hand is not sufficient. I discussed this with other members of the Maps WikiProject and the Graphics Lab and found that this sentiment was shared with many such users. I am also an active member of Wikidata, a recent project by the Wikimedia Foundation which is described in the introduction to this report. It is here that I began to notice another key issue: there is a lack of integration between Wikidata and Wikipedia causing the information on Wikidata to go largely unused. Attempts are currently being made by the developers of the Wikimedia Foundation to improve integration but I have noticed that their efforts have largely focused on the content of articles and infoboxes (specialised information tables) - not the actual maps that can be found within many of these articles. I therefore feel that my perspective as a member of both Wikidata and Wikiproject Maps will allow me to address a key issue I have identified: the missed opportunities that Wikidata could provide to address the time-consuming process of producing maps that are often soon out of date. As will be explored through this report, I feel as though this will allow me to not only give back to the Wikipedia community but also contribute to some very significant movements with broad reaching benefits.

Stakeholder Analysis

As a complex site with millions of unique visitors per month, a major change to Wikipedia like that which I propose would impact a vast number of people. It is therefore essential that I consider the various stakeholders that would be affected while developing a solution to my identified issue. To do this, I have identified several key groups of stakeholders. It is important to note that these stakeholders represent both the social and physical nature of my project: two key and often contrasting aspects I must balance. However, from previous technology projects I have learned that working with large groups of stakeholders can be very difficult as each has slightly different needs and values. As will be later explained, I countered this by selecting a particular Wikipedia editor

(MissAlaskaSunshine) to work with. The key stakeholder groups that I have identified are:

Article Viewers

Article viewers are by far the largest stakeholder group by number. In terms of the social environment, they also have arguably the most to gain. This is because an adequate solution to my issue would provide them with an up-to-date representation of data in a more meaningful way. The impacts of this will be broad reaching, but only if it is made accessible to the greatest possible number of readers. However, ensuring this will be difficult due to their diversity. The need for a multilingual approach due to their diverse ethnic backgrounds has already been previously discussed as key motivation for the movement towards Wikidata. Universal accessibility also requires allowances for physical disabilities. For example, 'alt text', a written description of a visual webpage element, needs to be implemented so that vision-impaired users can hear a spoken description of the maps.[14] As demonstrated by Figure 7, colourblindness also needs to be a key consideration of the social context. Figure 7a shows a map that was not constructed in accordance with Wikipedia's accessibility policies, namely its procedures to ensure ease of access by colour-blind individuals. To construct Figure 7b I used the 'Coblis' simulator to show how that map appears to a person suffering from a common form of colourblindness called Deuteranopia.[15] For example, Oranges and Yellows from the original map become almost indistinguishable and so information presented by the map cannot be fully understood. Conversely, Figure 7c shows a map that I constructed in accordance with Wikipedia's colourblindness policies. For example, it uses the colour purple which contains both red and blue, making it easier to be seen by individuals who can't distinguish red light (Protanopia) or those that can't distinguish blue light (Tritanopia). Figure 7d shows how an individual with either of these conditions would perceive the differences in shading, showing that while more difficult, it is still possible. Note that the black background of Figure 7d is due to a bug in the colourblindness simulator that makes it unable to handle the transparent background of the original. An automated approach, like that I propose, could force map creators to use colours from a fixed list. This will improve my project's relation with the social environment not only by providing stylistic consistency but also by ensuring that only colourblind-safe colours, such as the purple of Figure 7c, are used. I will also later consider the technological feasibility of allowing individuals to disclose any impairments to their colour vision. This would, for example, allow the map creation engine to avoid using blues for maps displayed to individuals who are unable to distinguish blue hues.

07 Demonstration of the Importance of Colourblindness Policies

7a: Non-compliant (original)

7c: Compliant (original)

7b: Non-compliant (simulation)

7d: Compliant (simulation)

Source: *7a by 'Tounsimentounes', 7c by self, simulation by Coblis*

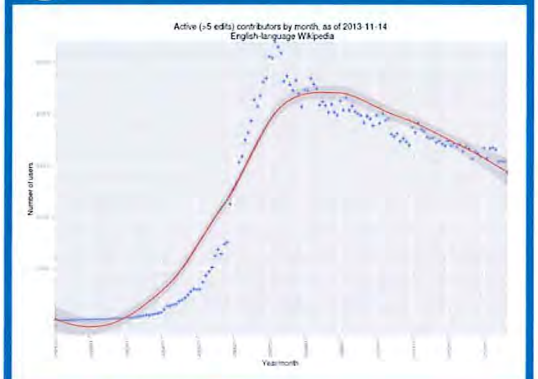
A consideration of the physical environment in which my project must act is also essential to ensuring that it fulfils the needs of the article-viewing stakeholders. The most recent comprehensive survey of Wikipedia readers found that in 2011 over 21% had accessed Wikipedia from a mobile device, with a similar percentage using tablet devices.[16] Due to the current technological trend towards larger phones with more powerful processors, as well as the increased market penetration of tablets, I

anticipate that these figure are even higher today. Therefore, my solution must be suitable for this form factor. Considerations that this requires include the ability to run on many different screen sizes, and low bandwidth usage to enable usage over 3G connections. Complex graphical rendering tasks must also be avoided as they can quickly drain the battery and may face performance issues on less-powerful smartphones. Another key technological trend is the avoidance of browser plugins in modern websites. This is important to ensure compatibility with mobile devices while also affording performance and security benefits. I should therefore aim to avoid these plugins in my solution so that the largest possible number of article readers can benefit from the advantages of my project.

Wikipedia Editors

A key attribute that defines my stakeholder analysis for Wikipedia Editors is that they are volunteers who are under no obligation to edit the site. It is therefore essential that my project does not act to disincentivise their efforts such as through a poor editing experience or an even-more time consuming workflow. The importance of this is shown in Figure 8 which shows the current trend of the number of Wikipedia editors declining since a peak in 2007. This threatens the sustainability of the site and its ability to fulfil the goals and values of the Foundation such a comprehensiveness and accuracy. Determining the cause for this decline was a key motivation for the aforementioned 2011 survey, which found that as few as 6% of readers to the site had ever edited a Wikipedia article. Among the reasons given, 18% stated they "do not know how," 6% were "not comfortable with tech" and 3% outright stated that the editing tools were "not user friendly." (respondents could select multiple responses). To address this, the Wikipedia VisualEditor was developed in 2013 to provide a WYSIWYG ('What You See Is What You Get') editor instead of requiring new users to learn the often-cumbersome wikitext markup language.[17] This example fundamentally shows that to fit into the social environment and Foundation's goals, my projects solution for automatically creating maps must be both intuitive and approachable. This idea of ensuring user-friendliness with therefore play a major role in choosing my solution and evaluating the fitness for purpose of my prototypes. In terms of the physical environment, the Wikipedia Editors have similar requirements to those of the Article Viewers: my solution must be cross-platform and free of performance and security issues.

08 **Trend of Active Wikipedia Editors**



Source: *Okeyes (WMF)*

It is important to note the unique context of the editing culture on Wikipedia: editors tend to specialise at specific tasks. This attitude arose to ensure the consistent implementation of Wikipedia's policies, and also as a way to best utilise a user's particular skills. Therefore, while my project needs to be accessible to the greatest possible number of users, it will likely be most used by a select group of individuals. A key consequence of this narrower scope is that it would be feasible to provide support on the discussion threads of Wikipedia. This is in contrast to the Visual Editor which is utilised on such a scale that users must be able to use it independently. While this doesn't preclude the need to ensure user friendliness, it does allow me to shift the balance between simplicity and functionality to provide more advanced features. This is important to this context of specialisation as it means that a large proportion of my users will use my project repeatedly. In this way the provision of advanced features is important to ensuring I meet my stakeholders' needs as it will allow those who invest the time to learn how to use my project to gain significant advantages in their productivity.

The values and views of my article editing stakeholders will likely be most influential in shaping my project as they will spend the most time using my project and interact most directly with it. I therefore decided to identify a specific editor who I considered generally representative of this wider stakeholder group. I contacted MrsAlaskaSunshine, an editor I have worked with previously, and found that she shared my sentiment that the current approach for creating maps on Wikipedia is not sufficient. She is experienced at editing Wikipedia, somewhat proficient at Wikidata but most importantly, has no coding experience. This is important if I choose implement a scripted approach because it means that she is representative of the broader editor community in which around 11% have programming experience. I am aware that her status as an experienced editor does mean that she is unsuitable for directly discussing whether my project will be helpful at fulfilling the Foundations goals of attracting and retaining new editors; I never the less feel that her stakeholder feedback will be beneficial to guiding my project's develop and so I will use her as my primary stakeholder and client throughout this report. (Note that MrsAlaskaSunshine changed her name during the course of my project; 'MrsAlaskaSunshine' is her current username that will be used throughout this report for consistency.)

Developers

The developers of MediaWiki are a mixture of MediaWiki employees and volunteers who work together to implement new feature and maintain existing functionality.[18] While smaller in number than article readers and editors, developers are never the less a critical stakeholder group for me to consider as I will require their permission to make certain modifications to the behaviour of the site. They also have extensive knowledge of the physical environment that my project will operate and so I anticipate that they will be a useful source of information on the security and stability concerns that may arise throughout my project.

Wikimedia Foundation

The values of the Wikimedia Foundation were first explored in the Background section of this report. These values are representative of both the social context of my report and the metric by which its fitness for purpose will be evaluated; They will therefore be a major focus throughout my report. Besides its values, the Foundation are also a major stakeholder in a pragmatic sense. For example, they provide the servers that both Wikidata and Wikipedia run on and so are financially impacted if my project introduces performance issues that require additional allocations of server resources. It is also to important to the context in which the Foundation is ran: they are a non-profit organisation headquartered in the United States but involved in advocacy across the World. Their status as non-profit therefore acts as a constraint to my project as their limited financial resources mean additional server resources is likely not an option. As a non-profit, the Foundation is largely funded through donations (some comes from grants etc.) and so they indirectly act as a stakeholder as my project's effect on user experience could potentially have an impact on donations; I anticipate that this is most likely to be a negative relationship: a serious error in my project temporarily brings down the site and hence the source of donations. This will therefore have to be seriously considered and actively avoided throughout the development and evaluation of my project.

Possible Solutions

My above analysis of the stakeholders' needs is crucial to my determination of which solutions will meet both the social and physical needs of my target audience. For example, while I have experience in using Adobe Flash (using the Adobe Flex SDK) to create web applications, the aforementioned needs of readers, and the general web trend away from plugins, precludes a Flash based solution. Instead, I must adopt one of the following possible solutions:

09 **Map Maker Previously Created by Self**

Source: *created by self*

Semi-Automated Approach (Invalid)

This project is actually not my first attempt at trying to find a better way to create maps on Wikipedia. Figure 06 shows a program I created two years ago. Since it was for personal use, it didn't consider the needs of others and so it is not fit for purpose within the context of this project. Never the less, it is still useful for consideration as a case study to show some lessons that I can apply to this project. For example, the original versions I made did not have a live preview. This made it extremely difficult to spot errors as I found created the map and so I later implemented one. Another key limitation was that only the raw SVG map was outputted, not a list of countries. It was therefore not possible to update the map once created and so changes to the map had to be manually made using a graphics editing software such as Adobe Illustrator. This meant that its use would have to be limited to the editors who posses this skill. The pool of potential users would be further constrained by the fact that it was written in Objective C and so could run only on Mac OS X. This example therefore highlights two key needs that my project must meet: cross-platform support and maintainability through the an ease in updating the map once constructed. It is reflection on this latter lesson that was the initial inspiration for basing my report off an integration with Wikidata.

Scripted Templates

My initial thoughts on how best to implement my solution were to utilise templates embedded within the actual articles on Wikipedia. The wikitext syntax to include a template is *{{template_name}}* with parameters specified using the a vertical bar, such as *{{template_name | parameter1 | parameter2}}*. Templates can in turn execute behind-the-scenes Lua scripts to provide dynamic functionality. I only have limited experience with writing Lua scripts for Wikipedia so I completed a few tutorials to gain a better sense of whether it would be feasible to implement my solution in this language. Overall, I found the the language was fairly intuitive and well documented and so felt comfortable that I would able to learn it to a sufficient level within the requisite time. It is also well integrated with both Wikipedia and Wikidata, furthering its suitability in this physical context.

While technically feasible, I began to worry that the syntax for templates would be too restrictive for my purposes. To evaluate this, I brainstormed several possible syntaxes. Figure 10 shows my first attempt as both simulated wikitext template syntax and pseudocode of how it could be implemented. Under this approach, the context of the map is first selected, for example the World. The behind-the-scenes mechanisms of the template would then iterate through each item of the context, such as every country in the world, and find the value of that item for a given property. Possible values are given along with colours that should be shaded if that particular country has that value for that property. Note how as per prior stakeholder analysis, this pre-prototype uses a colour palette to ensure consistency and accessibility to colour vision impaired readers. (Also note that the U.S. spelling of

10 Attempt 1 at a Template Syntax Based Approach

Template Syntax:

```
{{map|world|default=MediumGrey|property=
governmentType|case1=presidential|color1=blue|
case2=monarchy|color2=mediumGreen}}
```

Pseudocode:

```
func createMap(context,default,property,[cases])
 create a new map object called 'map'
 set the default colour of 'map' to 'default'
 for each item in map.allItems
  get value of property for item
  store this result as value
  //Cases is a dictionary of value and colour
  for each case in cases
   if case[0] equals value
   map.setColorForItem(item,case[1])
        break the loop
  if map.colorForItem(item) is not yet set
   map.setColorForItem(item,default)
 render the map
 display the map
```

'color' is used in accordance with Wikipedia's policies.) I showed my Template Syntax for Attempt 1 to MissAlaskaSunshine and other editors of Wikipedia. These stakeholders agreed that this syntax appeared to be fairly intuitive. However, as I thought more about Attempt 1, I began to fear that this syntax is highly restrictive. For example, it can only handle one property at a time so couldn't, for example, find all countries in which *either* the head of state *or* the head of government is female. It also completely lacks the ability to handle date based or numerical data, such as population. This precludes economical maps showing aspects such as countries by GDP or quantity of exports. Since a large proportion of maps on Wikipedia fit into this category, I believe that Attempt 1 would not meet my stakeholders' needs.

Figure 11 shows Attempt 2 at creating a template based syntax based on what I had learned from the first attempt. A key difference in this case is that it implements a fall-through condition matching model. Under this approach, if the first condition holds true, it is shaded that colour. Otherwise, the next condition is executed and so on until either a true condition has been found or it reaches the end, in which case the default colour is used. As I continued to consider Attempt 2, a key issue I kept coming across was how to deal with the fact that Wikidata items often have multiple values for a single property. For example, a large country like Russia would have multiple values for the 'Time Zone' property. To overcome this, I researched vector programming languages such as MATLAB and R as they already have built in operations for comparing vector and scalar variables.[19,20] I could not find any pre-existing solutions that would fulfil my stakeholders' needs: the approach needs to be both versatile and easy to understand. I therefore determined that I would need to develop my own set of comparison operators. I initially feared that creating a whole new set of comparison operators would confuse any of my stakeholders who already have programming experience. However, after further reflection, I recalled that in this context the vast majority of users do not fit into this category. I therefore do not anticipate that this will cause

11 Attempt 2 of a Template Syntax Based Approach

Comparison Operators:

Operator	Name	Purpose
->	Commonality	One object is common to both sides
->>	Total Inclusion	All objects on the left are in the right
<<-		All objects on the right are in the left
<<->>	Strict Equality	Both sides contain the same objects

Template Syntax (sample):

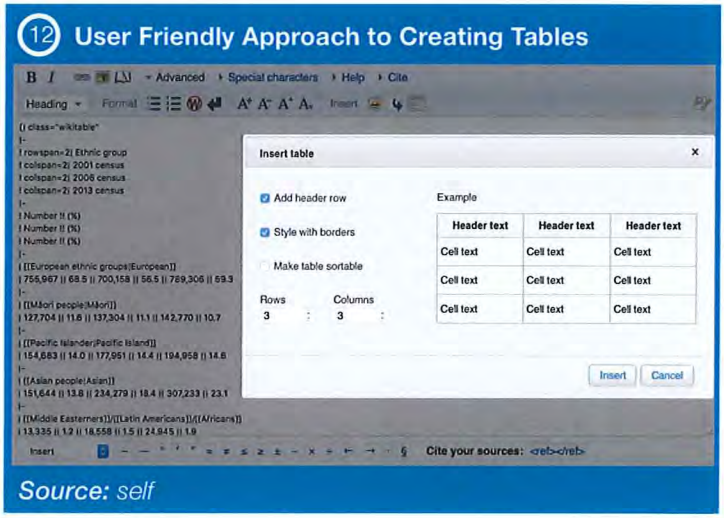
```
{{map|world|mediumGrey|Countries by Form of
Government|item.governmentType->[presidential
republic,semi-presidential republic];blue|
item.governmentType->Parliamentary republic;red|
item.governmentType->[absolute
monarchy,constitutional monarchy];purple]
```

any major issues. As shown in the table of Figure 11, I propose that '—>' could be used as the commonality operator which would be true when one object can be found on both sides. For example, *"item.treatiesSigned —> newZealand.treatiesSigned."* would be true for each country that has signed at least treaty that has also been signed by New Zealand. Note that this example also shows another convention of this proposed query language: like Attempt 1 it iterates through all the countries etc. in the map, using 'item' as a placeholder for each; a '.' is used to access an item's properties. A '—>>' would indicate total inclusion meaning that all the objects on the left are found on the right hand side, but not necessarily vice versa. For example, *"[1,2]—>>[1,2,3]"* would be true but *"[1,2,3]—>>[1,2]"* would not be. A useful example of this includes "item.treatiesSigned <<—*[First Geneva Convention, Second Geneva Convention, Third Geneva Convention, Fourth Geneva Convention]"* would return the countries that have signed all four Geneva Conventions. Note how the direction of the arrow can be reversed for convenience. Standard mathematical conditions would be used for numerical properties. For example, "item.population >= 100000000" would return all countries with populations of at least 100,000,000. Using these conventions, Figure 11 also shows how multiple conditional expressions could be used together to generate a complex map. This is done by implementing each case as two arguments separated by semicolons; the first term gives the condition and the second gives the colour to shade that country if true. Parameters also specify the type of map and its default colour.

While more versatile than Attempt 1, my second attempt is clearly extremely convoluted and confusing. The stakeholders I showed it to also felt this way, with MissAlaskaSunshine noting how she felt as that it would be extremely difficult to spot any mistakes that she may have made. For example, it is hard to quickly see which colour corresponds to which condition. I also feel that the use of semicolons would likely confuse many editors as they do not normally appear in template syntax. The preceding examples all use simple English names for properties and items e.g. "absolute monarchy" and "governmentType." In reality, this could not be the case due to multilingual concerns and the fact that multiple distinct items may have the same name. Instead, the entity id values (e.g Q184558) would have to be used. This severely detracts from ease of use as users would have to individually look up each code, a very time intensive task. In other cases, the user would be tempted to recall memorised codes. These codes were not designed to be easily memorised (there is no particular order to their number etc.) and so this is very likely to lead to errors. Based on these factors, I feel that Attempt 2 does not meet my stakeholders' needs and so I will have to consider alternatives.

Web-Based GUI

Generally speaking, the deficiencies of the above template-based approach lie not directly in the syntax but rather the way in which users would have to type it. This reminded me of a similar situation on Wikipedia in which the syntax for creating tables was found to have an extremely steep learning curve. As mentioned previously, Wikipedia has been progressively valuing more intuitive in order to reverse the trend seen in Figure 08 of a rapidly decreasing editor population. Figure 12 shows the solution that was reached for tables: a graphical pop-up box that allows users to graphically edit the basic parameters using a live preview. Pressing the "Insert" button automatically



adds the corresponding wikitext to the article. (The background of Figure 12 shows the corresponding wikitext of an actual table.) This seemed to be the perfect approach to implementing my table syntax as it fulfils both the key requirements I had determined through stakeholder analysis and feedback: it is both user friendly and error-resistant. The graphical user interface fulfils the former need while the live preview allows errors to be seen and corrected before the edit is made publicly visible. I therefore began researching how I could implement a similar solution to my context. I found that MediaWiki, the engine that Wikipedia runs on, is largely built in PHP (a server-side web scripting language.)[21] I do not have any experience with PHP but research showed that the creation of MediaWiki extensions is very well documented. The open source nature of the project also means that I would easily be able to use similar extensions as examples.

Extensions are run on the central Wikipedia servers and hence their creation is closely monitored by the core development team. As previously mentioned, this is necessary to ensure they do not inadvertently cause performance and security issues to users of the site. I therefore reached out to my developer stakeholders to describe my approach and ask for advice on how to implement it. They were extremely concerned by the fact that I would be creating an extension with limited knowledge of PHP as they believe that this means it is more likely that my code will be unstable and error-prone. I understand these concerns as there have apparently been instances of poorly-written extensions crashing entire servers and leading to website downtime. This is detrimental to all of my stakeholders: readers will not be able to view the site, editors cannot perform their tasks and the developers would have to implement reversion procedures. The impacts on the Wikimedia Foundation, including contradiction of they values of and loss of donation revenue, have already been discussed. To prevent this, one individual developer pointed me to some useful documentation on common pitfalls of PHP programming, key bug reports showing known issues and information on how to properly debug Wikipedia extensions by running a local test MediaWiki server on my home computer as well as on Test Wikipedia, a special shadow version of Wikipedia that is used for testing new software features.[22] He also offered to answer any specific technical questions I may have.

After establishing that the creation of a Wikipedia extension is a possibility, I now needed to determine the approach that I would take. Further research into the MediaWiki template documentation lead me to realise that I would actually have to create two separate extensions: a User Interface extension to handle the graphical construction of the map by the user and a Parser Extension to interpret my template syntax and output the corresponding result.[23] (Technically these could be combined into one extension with a common codebase but this would likely impede maintainability). My previous experience with dynamically creating maps, such as that of Figure 09, involved using a vector image format called SVG which can be directly embedded within an HTML page.[24] However, this requires that the reader is on a device that supports HTML5, a relatively new standard.[25] In my opinion, forcing this as a requirement would be contrary to the values Wikimedia Foundation as their goal to ensure that every human being has access to "the sum of all knowledge" includes accessibility across various computing platforms. For example, the Wikimedia Foundation made a major effort to support the One Laptop Per Child (OLPC) initiative in which low-cost laptops were freely distributed to developing nations.[26] The software used by these devices and their limited processing power means that they would not be able to render these maps if they were based on HTML5 SVG. This reflects a key paradox while developing any technological project: the broader the target audience, the less suitable the approach is to any single individual. Therefore, while an HTML5 SVG could perhaps allow more novel features, I feel that it is important that I do not restrict my target audience to users with capable devices as I feel that this goes against the values of the Foundation while also would negating my desires to align this project with a broader movement: the democratisation of information through technology. I discussed these grievances with the aforementioned Wikipedia

to create the maps is ran under a desktop environment, with the result then somehow inserted into the corresponding articles. Consultation with my developer stakeholders informed me of the Wikimedia Tool Labs service which hosts these sorts of scripts on Wikimedia servers. However, for the purposes of initial development, these scripts will be ran on my home computer. Besides the advantages of reduced server load by only querying once per map, not per user, a desktop based approach is beneficial as it allows me to use a programming language that I am more familiar with: Python. Python can be ran on Wikimedia Tool Labs and is well-integrated with both Wikipedia and Wikidata through libraries such as Pywikibot.[29] It is also cross-platform and hence is more suitable to my physical environment than other desktop based languages such as Objective C that was used in project detailed in Figure 09. Python is also well regarded as being intuitive and easy to learn and so I feel that using it across project would be beneficial if I end up implementing a solution in which users write their own scripts. I therefore conclude that the use of Python will be most viable in the context of my project.

A desktop-based solution frees my solution from having to fit into the confines of the template syntax. I am therefore able to consider pre-existing solutions that I could potentially integrate into my project. One such existing solution that I researched was RDF (Resource Description Format), an open file format that provides metadata about an object and hence is used as a part of the broader semantic Web 3.0 movement. Wikidata provides an RDF interface but this simply returns raw information: a querying language is required to actually interpret it and make conclusions. By far the most common query language for RDF is SPARQL, a SQL based querying language designed specifically for RDF. Figure 15 shows an example SPARQL query to return all countries in Africa. However, I feel that this syntax is neither easily readable nor succinct and so its lack of user friendliness would, in my opinion, jeopardise my project's fitness for purpose if I were to implement it. I am aware that a pre-existing solution would improve my ability to interact with other web services as a part for the broader semantic web movement; However, I feel that utilising a pre-existing solution would severely compromise my project's fitness for purpose and so alternative solutions tailored to my project's unique context must instead be considered.

15 Example SPARQL Script

```
PREFIX ex: <http://example.com/exampleOntology#>
SELECT ?capital ?country
WHERE {
  ?x ex:cityname ?capital ;
     ex:isCapitalOf ?y .
  ?y ex:countryname ?country ;
     ex:isInContinent ex:Africa .
}
```

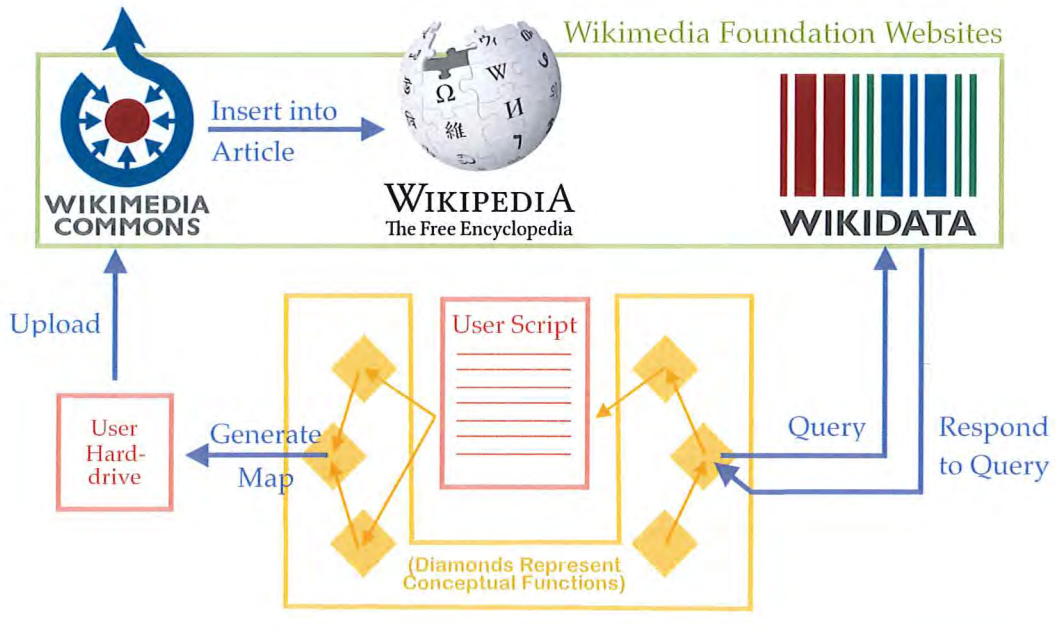
Source: https://en.wikipedia.org/wiki/Resource_Description_Framework

I briefly considered a drag and drop approach but soon reject this as it would be too inflexible. Instead I must implement an approach in which users write scripts that interact with a library written by me to both query Wikidata and generate maps. I do not feel that the need for these scripts to be written in Python will be burdensome to map creators as many of the scripts will be largely similar and the aforementioned context of specialisation should allow assistance and ensure eventual familiarity. This means that good documentation with plentiful examples should suffice. I do not anticipate that requiring users to develop and run the scripts within the built-in Python IDE (IDLE) will be too burdensome; however this will be a major focus of my evaluation where I may consider adding a graphical user interface (GUI) based on stakeholder feedback.

Now that I had decided upon the method by which I would query the data from Wikipedia, the next step was to determine how these maps would actually be presented. I briefly considered utilising the Graphoid service (see Figure 13). However, consultation with my developer stakeholders proved this to be unfeasible as they feared that the large number of Wikipedia articles using maps could mean that the server resources consumed by Graphoid would become unreasonable. Hence, to ensure the

scalability of my project I must find an alternative. After considerable thought I determined that the most feasible solution is actually the most similar to the current solution: simply uploading SVG maps to Wikimedia Commons. These maps would be automatically created by a user's script and then uploaded to this central repository. In terms of the social environment, I do not feel the need to manually upload these maps will be burdensome as my target audience of map editors are already familiar with this process, which implements various batch uploading provisions to save time. This proposition also avoids the need to make an entirely new syntax just for adding maps to articles as the existing syntax to insert images from Wikimedia Commons would be used. This reduces the learning curve for all editors, including those who are not interested in creating new maps but are never the less involved in maintaining them such as by moving them and changing their size. By utilising the mature mechanisms of Wikimedia Commons, features such as categorisation are also inherently available as a way to meaningfully store maps. Due to this, this approach is arguably more in line with the values of the Wikimedia Foundation as this reuse makes it easier for one map to be reused across the many different language versions of Wikipedia. Manual upload could even be potentially beneficial in terms of quality as it forces each editor to at least glance over their map before it is submitted, providing one last chance to spot glaring errors. This approach also bypasses the need for the Wikimedia Tool Labs as the code could instead script could instead be ran on the computers of individual editors. While the Wikimedia Tool Labs could optionally be used at a later stage, their avoidance makes my project more feasible as it prevents the security issues that would be introduced if scripts created by users were allowed to run on Wikimedia Foundation servers without individual approval. While my developer stakeholders did feel as though this issue could be addressed by restricting access to approved editors only, I feel that this contradicts the values of the Wikimedia Foundation in terms of democratisation and openness. The existing syntax for inserting images already includes provisions for vision impairment, such as the aforementioned alt text, and so I feel that this approach is also inline with my needs to ensure accessibility to users. I will therefore proceed with implementing this approach as Prototype 1 with Figure 16 providing a conceptual overview of this solution.

16 **Conceptual Overview of My Solution**



Source: *created by self, logos courtesy of Wikimedia Foundation*

Brief

By on my stakeholder analysis and consultation with my stakeholders I have identified the key requirements that my project must meet. This brief was presented to MrsAlaskaSunshine who agreed to it in her role as my primary client.

Physical:

- Must not introduce any significant performance, security or stability issues to the site
- The editing tools and the mechanism for viewing maps must be available on a broad range of computing platforms
    - Noting the trend towards smartphones and the trend away from web plugins, both low-performance and native clients must be compatible
- The maps produced must promote currency through automatic updating by querying Wikidata

Social:

- Must be accessible to as many editors and article viewers as possible. This requires considerations for:
    - Vision impairments and limited colour vision
    - Multilingualism
    - Computing platforms of developing countries
- Maps should be consistent both stylistically and with regard to Wikipedia's policies
- The editing tools must be both:
    - User friendly so as to maximise user participation
    - Comprehensive so that experienced users can maximise their efficiency through advanced features
- The values of the Wikipedia Foundation, such as openness and transparency, must be maintained
- My project should fit into the broader technological trend towards the semantic web through curation and interactivity
    - However, the use of information from other sites is beyond the scope of this project

Initial Prototype Development

Production

After all the research done in the preceding section, I felt as though I was ready to begin implementing my project. A key issue that continually came up in my previous analysis was the difficulty in balancing the difficulty of remembering entity id's and their advantages in terms of uniqueness and multilingualism. I ultimately decided that a sole reliance on entity id's would not suit my stakeholders' needs in terms of ensuring that my project is user-friendly. However, an approach based solely on natural language terms would not suit the physical environment as technological limitations mean that ambiguities could not be resolved. After much thought, I felt as though the only possible compromise was to hard-code a list of commonly used items and properties that could be referenced by name. Items and properties not on that list could be referenced using their entity id's by manually looking up those values on the Wikidata website. While not ideal, MrsAlaskaSunshine and I both feel that the vast majority of maps will be based on only a few properties so this should not be an issue. In accordance with prior stakeholder analysis and feedback, I implemented a similar list for colours, ensuring the consistency and accessibility for colour-blind individuals as previously described. As I created this list, I noticed that many items were referred to by multiple names and so I later amended the syntax to add aliases to this list so that multiple items can refer to a single entity id. Figure 17 shows an excerpt from these three lists to demonstrate this basic syntax. Within each line, a the first term is the name of that item / property / colour followed by a colon and then either its entity id in the case of items and properties or a hex code in the case of colours. Aliases are specified are separated by commas. Lines beginning with a '#' are ignored by the interpreter and so can be used as comments to organise the lists. I then created functions to parse these lists so that, for example, *"i('male')"* would call the parser function with the 'friendly' name of the item, causing it to return the corresponding entity id for the male item (Q6581097). I also created a similar function called 'p' to return properties so that *"p(country)"* would return the entity id (P17) of the country property. To facilitate multilingualism, multiple versions of these files can be created, with the filename of each reflects its Wikipedia language code. For example, "items-en.txt" includes a list of friendly names in English e.g. "male:Q6581097" whereas the "items-it.txt" would include a list of friendly names in Italian such as "maschio:Q6581097." I considered creating a set of scripts to managed these lists but later decided that that syntax used is simple enough for editors to simply use a text editor such as Notepad or TextEdit (the Mac equivalent). Therefore, while my project will come with a pre-existing set of items etc., users will easily be able to modify them if they choose. Also note that the colours are aliased with names that denote function as well as hue. In this way the colour palette is implemented in such a way that consistent adherence to conventions is enforced. For example, as shown in Figure 05, maps of airline destinations are all shaded in light blue so 'airlineBlue' is listed as a colour to remind users of this convention. Ultimately this need for stylistic consistency was a key discovery of my stakeholder analysis and so I feel that this will help improve my project's fitness for purpose. The actual function to access colours is called 'color(…)' and performs similarly to p(…) and i(…). It is however aliased as colour(…) so that both

17 Excerpt of Lists Showing Basic Syntax

Items:

```
male:Q6581097
female:Q6581072
english:Q1860
landlocked country:Q123480
#airlines
turkish airlines:Q4548
lot polish airlines,lot:Q204268
air canada:Q185339
all nippon airways:Q204284
air europa:Q332980
eva air,eva:Q715113
avianca honduras:Q846922
aeromexico:Q381563
scandinavian airlines:Q187854
china southern airlines:Q291090
thai airways:Q188710
emirates (airline),emirates:Q180432
```

Properties:

```
#general
isa:P31
instanceof:P31
operates:P121
country:P17
#country
headofgovernment:P6
headofstate:P35
officiallanguage:P37
language:P37
currency:P38
continent:P30
memberof:P463
#person
sex:P21
gender:P21
sex or gender:P21
religion:P140
```

Colours:

```
purple, female:#ad7fa8
default:#b3b3b3
purple:#ad7fa8
blue,male,airlineBlue:#729fcf
orange,airportOrange:#f57900
```

Source: self

names can be used interchangeably.

Now that a reliable and convenient method of retrieving entity id's had been developed, I began writing the library that provides the high-level functions called by user's scripts; these functions then in turn call lower-level functions that I wrote to authenticate the connection with Wikipedia and work with the aforementioned Pywikibot to call the Wikidata API and interpret the result. As shown in Figure 18a, I only found the need to implement three high-level functions that are called directly by the user. The function named 'query' is by far the most important as it takes an item and property as its parameters and returns the corresponding value(s). These values are then interpreted by the user's script to determine how the map should be shaded. 'itemFromPage' is a function that returns the entity id of an item based on the name of its Wikipedia article. The language can optionally be passed as a parameter, otherwise it defaults to English. Due to the previously-described limitations of interactions between Wikidata and Wikimedia, this function is relatively computationally-intensive. Its use should therefore be limited to a couple of calls per script. For example, Figure 18b shows its use to obtain a reference to the African Union item which is compared against every country that is being tested. A final function called 'getName' returns the name of an item based on an entity id. The language can optionally be specified, otherwise it defaults to English. This function is only intended to be used for testing purposes such as validating that entity id values are correct.

18 High-Level Query Functions

a) Conceptual:

Name	Inputs	Output
query	entity id of item and property	List of values, 'None' if not found
itemFromPage	name of article, optionally language	entity id of corresponding item
getName	entity id of item, optionally language	Name in given language or English

b) In Practise:

```
from automap import *
from api.query import *
map = dataMap('world','africanUnion',colour('default'))

africanUnion = itemFromPage('African Union')
for country in map:
        if africanUnion in query(country,p('memberof')):
                map[country] = colour('blue')
map.generate()
```

Source: self

Another key aspect of my project is the library that provides high-level functions to generate SVG maps. 'map' is a reserved keyword in Python so I had to codename this library "AutoMap." My previous attempt at a map generator that I detailed in Figure 09 used a highly modified SVG file with placeholder text that was replaced based on the colour it was to be shaded. For example, the entire file would be searched for the string "@NZ" so that it could be replaced with the colour that was to be shaded; if New Zealand was to be shaded in with the colour #ad7fa8, "...fill=@NZ..." in the original SVG file would be replaced with "...fill=#ad7fa8..."While adequate, this requires that a 7726 line file is searched through 247 times (once for each country) leading to a each map taking a long time to generate. Figure 19 shows the alternative approach I found: using CSS embedded within the SVG to modify the stylings, and hence colour, of the countries defined below. Note how this figure also shows that large and small nations are handled differently as countries smaller than 20,000km^2 are rendered with circles around them so that they can be more easily discerned (this can be seen in Figure 05). When developing AutoMap, I initially implemented its functions as high-level on a

19 Extract of Test Map showing CSS within SVG

```
.gd, .as, .mh, .sg, .tt, .bm, .fk, .jm, .mp, .ki, .ky, .bs, .ws, .nu, .tv,
.lc, .fj, .gi, .to, .pr, .kn, .sc, .vi, .gg, .dm, .ag, .vc, .ms, .ai, .im,
.mt, .gm, .je, .bb, .vu
{
opacity:1;
stroke:#ad7fa8;
fill-opacity:0;
}
.zw, .ss, .in, .bw, .sl, .ls, .cm, .pg, .pw, .ie, .hn, .na, .gy, .sz, .tz,
.sd, .lr, .pk, .ca, .ph, .nz, .au, .sb, .gb, .ug, .ng, .zm, .za, .mw, .gh,
.bz, .rw, .ke, .er
{
fill:#ad7fa8;
}
```

Source: self

custom class called dataMap. Examples of functions included one to return all the countries etc. included on the map and another to set the colour of a specific country. However, as I tried implementing a few queries I found that most followed a basic pattern: iterating through all the countries in a map, querying Wikidata for that country based on a property, and then shading a map a particular colour given the result of this query. Based on this, I begin considering implementations of AutoMap that would streamline this interaction. Research into special types of functions and classes led to me to the concept of operator overloading in which basic operations of a programming language are customised.[30] Extensive usage of this to implement custom behaviour is generally frowned upon as it can confuse experienced programmers who are used to particular outcomes. However, based on my stakeholder analysis, it is reasonable to assume that the majority of my article-creating stakeholders will not be Python programmers and so this usage of operator overloading should not impact on my project's success. Overriding the __iter__ function provides a convenient way to iterate though all of the countries in the map simply with *"for country in map:"* where map is an instance of the dataMap class. The code within this loop is executed once per country, with the 'country' variable containing the entity id of that country for use in the 'query' function. *"map[country] = colour['blue']"* (by overriding __getitem__ and __setitem__) can then be used to set that country a particular colour. Note the use of the friendly colour names described in the first paragraph of this section.

20 Difficulties in Installing the 'Requests' Module

a) Error due to to missing 'requests' module

```
>>>
ImportError: No module named 'requests'
Python module requests is required.
Try running 'pip install requests'.
>>>
```

b) Difficulties installing 'requests'

```
Last login: Tue Oct 20 14:12:19 on console
Marricks-MacBook-Pro:~ Marrick$ pip install requests
You are using pip version 7.1.0, however version 7.1.2 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
Requirement already satisfied (use --upgrade to upgrade): requests in /Library/P
ython/2.7/site-packages
Marricks-MacBook-Pro:~ Marrick$ pip install requests --upgrade
You are using pip version 7.1.0, however version 7.1.2 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
Collecting requests
  Downloading requests-2.8.1-py2.py3-none-any.whl (497kB)
    100% |████████████████████████████████| 499kB 826kB/s
Installing collected packages: requests
  Found existing installation: requests 2.7.0
    Uninstalling requests-2.7.0:
      Successfully uninstalled requests-2.7.0
Successfully installed requests-2.8.1
Marricks-MacBook-Pro:~ Marrick$
```

Source: self

The first test of my project yielded the error message shown in Figure 20a. I tried repeatedly installing this module using the terminal command given but even though the terminal stated it was already installed, the error remained. Figure 20b shows that no combination of forcibly updating the module or uninstalling then reinstalling it fixed the issue. Research into the issue showed me that the module had been inadvertently installed to the Python 2 that is built into Mac OS X, not the Python 3.4 that I was using. Further research found that this issue could be addressed by using *"python3.4 -m pip install requests"* instead of just *"pip install requests"* as suggested in the error message.[31] I noted this terminal command as it will need to be ran by all editors using my project and hence should be added to the documentation I provide. The next notable error I faced in the development of this prototype is shown in Figure 21a. This error confused me as all the documentation and example code I looked at used "pywikibot" and so I could not understand why it was not defined. Since the pywikibot library is open-source, I was able to look through the original source code to find where pywikibot it defined and hence determine situations in which it would fail. Figure 21b show an extract of pwb.py, a wrapper through which the main pywikibot library is called. This shows that pwb, which will eventually hold a reference to the pywikibot library, is initially set to None. The try except block then attempts to initialise the pywikibot library and, if successful, assigns it to the pwb library. The try except block silences the error so that the program can continues to run as opposed to crashing. While this is is beneficial in the case of the prototype that I give to me stakeholders, it makes debugging more difficult as this suppresses the error codes that would ordinarily give a detailed description of what exactly is the problem. To uncover this, I manually removed the try catch block from within the pywikibot library. This yielded the error messages given in Figure 21c. This highlights another key

21 Difficulties in Interfacing with Pywikibot

a) Original error message

```
>>> ================================ RESTART ================================
>>>
Traceback (most recent call last):
  File "/Users/Marrick/Documents/AutoMap/scriptE copy.py", line 2, in <module>
    from api.query import *
  File "/Users/Marrick/Documents/AutoMap/api/query.py", line 3, in <module>
    site = pywikibot.Site('en','wikipedia')
NameError: name 'pywikibot' is not defined
>>>
```

b) Extract of pwb.py showing the excepted import statements

```
pwb = None
def tryimport_pwb():
    """Try to import pywikibot.

    If so, we need to patch pwb.argvu, too.
    If pywikibot is not available, we create a mock object to remove the
    need for if statements further on.
    """
    global pwb
    try:
        import pywikibot  # noqa
        pwb = pywikibot
    except RuntimeError:
        pwb = lambda: None
        pwb.argvu = []
```

c) Error message after modifying pwb (shows root cause)

```
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    query('Q30','Q31')
  File "/Users/Marrick/Documents/AutoMap/api/query.py", line 32, in query
    item = pywikibot.ItemPage(repo,code)
  File "/Users/Marrick/Documents/AutoMap/api/pywikibot/page.py", line 3457, in __init__
    ns = site.item_namespace
  File "/Users/Marrick/Documents/AutoMap/api/pywikibot/site.py", line 5537, in item_namespace
    self._cache_entity_namespaces()
  File "/Users/Marrick/Documents/AutoMap/api/pywikibot/site.py", line 5518, in _cache_entity_nam
espaces
    for namespace in self.namespaces.values():
  File "/Users/Marrick/Documents/AutoMap/api/pywikibot/site.py", line 806, in namespaces
    self._namespaces = _NamespacesDict(self._build_namespaces())
  File "/Users/Marrick/Documents/AutoMap/api/pywikibot/site.py", line 2158, in _build_namespaces
    is_mw114 = MediaWikiVersion(self.version()) >= MediaWikiVersion('1.14')
  File "/Library/Frameworks/Python.framework/Versions/3.4/lib/python3.4/distutils/version.py", l
ine 40, in __init__
    self.parse(vstring)
  File "/Users/Marrick/Documents/AutoMap/api/pywikibot/tools/__init__.py", line 331, in parse
    raise ValueError('Invalid version number "{0}"'.format(vstring))
ValueError: Invalid version number "1.27.0-wmf.3"
>>>
                                                                    Ln: 84  Col: 4
```

Source: self

22 Test of the AutoMap Library

```
from automap import *
from query import *

map = dataMap('world','test1',color('default'))

map[i('New Zealand')] = color('blue')
map[i('Singapore')] = color('orange')
map[i('Canada')] = color('purple')

map.generate()
```

Outcome:

Source: self

benefit of debugging interfaces with open-source libraries: small temporary modifications can be made for testing purposes. Ultimately the error showed me that the version of pywikibot I was using was not compatible with the current version of MediaWiki used on Wikidata. While I was able to simply update pywikibot, this error was significant as it demonstrated that I would need to mention the need to periodically update pywikibot in the documentation that I will provide to my stakeholders.

After overcoming the above issues, I created a few test maps to validate that the libraries were operating as they should. Figure 22 shows some a test script I made to validate AutoMap by shading three countries: New Zealand, Singapore and Canada. This proved that particular countries could be shaded in particular colours. Singapore was chosen as a test case to ensure that small islands are correctly circled whereas Canada was used as its many islands made it a good way to validate that the engine correctly colours the entirety of a country. Note how Figure 22 shows this map within a web browser. This is because the generated map is automatically opened once created, providing an implementation of the live preview that I previously identified as being essential to the success of my project. To validate the querying aspect of my project, I created a script to list each country along side its head of state as queried from Wikidata. This is documented in Figure 23 which also demonstrates that names using Unicode, such as that of Nebojša

23 Test of the Querying Library

Script:

```
from automap import *
from query import *

map = dataMap('world','test',colour('default'))
for country in map:
    print(getName(country)+': '+getName(query(country,p('headOfState'))[0]))
```

Outcome (extract):

```
Bosnia and Herzegovina: Nebojša Radmanović
Barbados: Elizabeth II
Bangladesh: Abdul Hamid
Belgium: Philippe of Belgium
Burkina Faso: Blaise Compaoré
Bulgaria: Rosen Plevneliev
Bahrain: Hamad bin Isa Al Khalifa
Burundi: Pierre Nkurunziza
Benin: Yayi Boni
Saint-Barthélemy:
Bermuda:
```

Source: self

Radmanović, are correctly handled. This shows that my project will correctly supports multilingualism, a key requirement of my stakeholders. This script took a while to run, suggesting that this prototype may have performance issues (this was later confirmed in Initial Prototype evaluation section). Another notable feature of Figure 23 is that blank spaces can be seen next to some countries such as Bermuda. Examination of these Wikidata pages uncovered the fact that they had a value of the Head of Government property but none for the Head of State. I changed my script to display the value for Head of Government but this only lead to different countries showing blanks as while they had values for Head of State, they did not have a value for Head of Government. (Note that the difference between Head of State and Head of Government is very nuanced but based on the system of government. In many countries one individual is both whereas other countries, including New Zealand, use it to distinguish ceremonial leaders e.g. the Queen from elected leaders such as the Prime Minister.) This sort of situation in which two properties are used for similar purposes is unfortunately common across Wikidata so I modified my query function so that it can handle either a single property or an array of properties. When an array is given, it queries the item for the first property in the array. If a value exists for that property it is returned. Otherwise the item is queried against the second property and so on. To validate this implementation, well as to test the combined use of both my query library and AutoMap together, I created the script shown in Figure 24. This script shades countries based on whether their Head of Government or Head of State (in that order of precedence) is male or female. It successfully shaded all nations with the exception of Switzerland. This is not an error as Switzerland's head of state is technically not an individual but a group called the Federal Council. However, the President of the Federal Council is female and so the penultimate line of code in Figure 24 manually shades Switzerland purple.

24 Testing the Combined use of Query and AutoMap

Script:

```
from automap import *
from query import *

map = dataMap('world','Countries by HOS Gender',colour('default'))

for country in map:
    values = query(country,[p('headOfGovernment'),p('headOfState')])
    gender = query(values[0],p('sex'))
    if item('male') in gender:
        map[country] = colour('male')
    elif item('female') in values:
        map[country] = colour('female')

map[i('Switzerland')] = colour('female')
map.generate()
```

Outcome:

Source: self

While the approach of iterating through the countries in a map is useful in most cases, there are situations in which it would not be desirable. For example, each airline's item includes a property

called Equipment Operated which lists the aircraft models that they fly. To shade a map based on which countries have airlines that operate a particular model, one would have to iterate through all the airlines, check whether they operate that model and then lookup their 'country' property to shade that country a particular colour. To facilitate this, I added a 'collection' function to AutoMap which acts in a similar way to the 'item','property' and 'color' function previously described: it parses a text file that contains a list of named collections of items and then returns an array of their entity id's. This implementation was tested in Figure 25 which generates a map of all countries whose airlines operate the Boeing 787 Dreamliner.

25 Use of Collections to Iterate through Airlines, not Countries

Script:

```
from automap import *
from api.query import *

map = dataMap('world','787',colour('default'))

boeing787 = itemFromPage('Boeing 787 Dreamliner')
for airline in c('airlines'):
    if boeing787 in query(airline,p('operates')):
        countries = query(airline,p('country'))
        map[country] = color('aircraftOrange')

map.generate()
```

Outcome:

Source: self

Outcome

By this stage I felt as though my prototype was stable and so I sent it to MrsAlaskaSunshine along with the examples that I had written. Since I had not written documentation at this stage, I helped her with a few issues over Skype. The impact of this on whether or not this can be considered a valid test of my prototype will be examined in the Prototype 1 Evaluation section of this report. I gave MrsAlaskaSunshine three specific maps to create: countries that belong to the African Union, landlocked countries and countries in which the national language is English (note that the US does not technically have an official language). She sent me the code for her implementations, shown here as ran from my computer:

26 Scripts Created by MrsAlaskaSunshine with Corresponding Outcomes

a) Countries of the African Union:

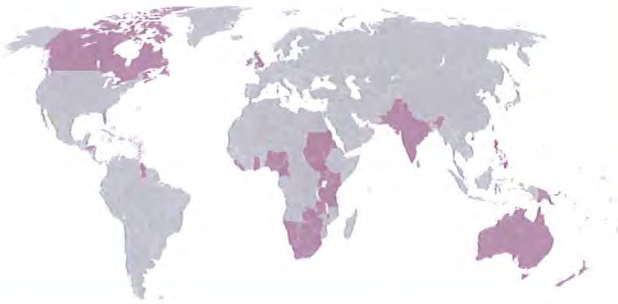
```
from automap import *
from api.query import *
map = dataMap('world','africanUnion',colour('default'))

africanUnion = itemFromPage('African Union')
for country in map:
        if africanUnion in query(country,p('memberof')):
                map[country] = colour('blue')
map.generate()
```

b) Landlocked Countries:

```
from automap import *
from api.query import *
map = dataMap('world','landlocked',colour('default'))

for country in map:
    if item('landlocked country') in query(country,p('isa')):
        map[country] = colour('purple')
map.generate()
```

c) Countries where English is A National Language:

```
from automap import *
from api.query import *

map = dataMap('world','englishLanguage',colour('default'))

for country in map:
    languages = query(country,p('officialLanguage'))
    if item('english') in languages:
        map[country] = colour('purple')

map.generate()
```



Initial Prototype Evaluation

Client Re-Visitation

After MrsAlaskaSunshine completed the three maps shown above, I asked her a few questions regarding her experiences in installing and using my program. I provided a written list of instructions for installation process which she described as "intricate but not difficult." She said that she found it incredibly easy to implement the African Union and landlocked countries maps (Figures 26a and 26b) as their query was very similar to the example scripts I gave her. She found the map preview to be useful but was confused by the fact that it opened in a web browser. I had told her that the SVG file would be saved to her computer but the web browser gave her the impression that it was actually somewhere on the internet. While I was able to quickly point her to where the actual file was saved, this interaction actually taught me a useful lesson which with hindsight seems obvious: my project will only be user friendly if it acts in ways users will expect based on previous experience. As a developer, I often use web browsers to preview files for compatibility purposes. However, stakeholder analysis showed that my key stakeholder group of editors, who are represented by MrsAlaskaSunshine, do not necessarily have much experience with computing and so it is unreasonable to assume that they will be familiar with the idea that web browsers can be used to open local files on their own computer. MrsAlaskaSunshine found it difficult to implement Figure 26c as none of the example scripts I gave her showed how to deal with properties that had multiple values, such as countries with more than one official language. I wrote her a few examples that handled multiple values but in different contexts. She was quickly able to modify these examples into scripts that completed the assigned task and so I am confident that this approach is suitable to my target audience. However, this does emphasis the importance of including adequate documentation in my Final Prototype. I also feel that MrsAlaskaSunshine's feedback shows that the current requirement for editors to create and run their own scripts in IDLE is not fit for purpose; I will therefore strongly consider the feasibility of implementing a graphical user interface.

While my client re-visitation does imply that my Initial Prototype was valid in terms of the social environment, I fear that was not an accurate reflection of how all editors would find it. Firstly, I consulted with MrsAlaskaSunshine throughout the production of my prototype and so she was generally well aware of its functionality and its key patterns. As described in my stakeholder analysis, a key strategic aim of my project is to ensure it is inline with the Wikipedia values of being editor friendly in an attempt to mitigate the trend of decreased editorship. Since I to chose work with MrsAlaskaSunshine as my primary stakeholder due to her enthusiasm to find a way to use Wikidata with Wikipedia, she is likely not representative of this trend which is more to do with less-motivated editors and inexperienced ones who have only just began editing the site. While unavoidable due to the lack of documentation, the direct help that I provided her meant that this test was not representative of a real-world case. However, I believe that this did not have any effect on the outcome as an actual deployment would not only include adequate documentation but also the help and discussion pages that are common across Wikipedia, allowing editors to receive support about specific issues. After considering the responses of MrsAlaskaSunshine, and my analysis of its validity, I believe that my project is probably suitable for the social context. I however believe that I have sufficient feedback to develop my Final Prototype. Lessons from testing this prototype will be applied to ensuring that my tests of the Final Prototype are more representative of my stakeholder groups. This will allow my to definitively determine whether my project is fit for purpose with regard to social environment.

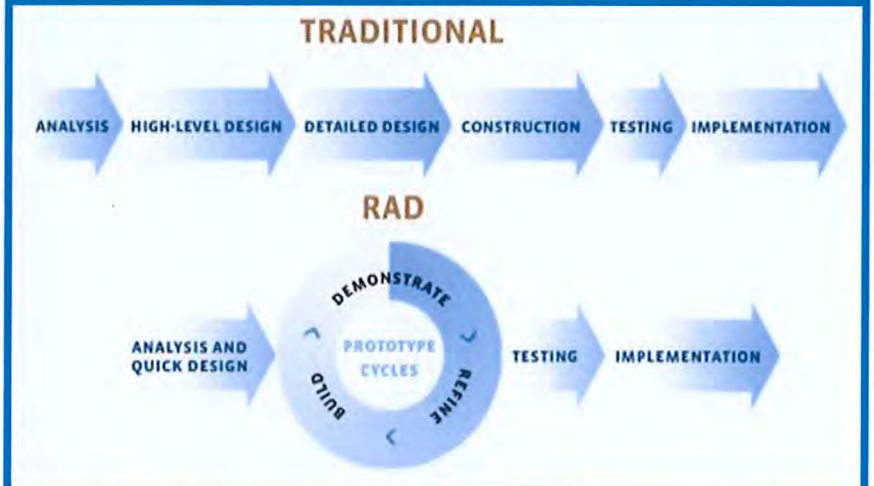
Personal Reflection

While MrsAlaskaSunshine's results may not accurately reflect all editors, they certainly do not reflect

the social needs of all of my stakeholders. I can therefore only fully analysis the social fitness for purpose of my project if I also consider my stakeholder analysis. In terms of article-viewing stakeholders, a key requirement that I noted was that it must be accessible to as many individuals as possible both to widen its potential use and to fulfil the values of the Wikimedia Foundation. By using Wikimedia Commons, the need for multilingualism was fulfilled as the maps are stored in a language-independent repository. The approach I chose unfortunately means that all users must see the same map and so I was not able to implement my previous idea of letting colour blind users specify which colours they can't discern and serving them with customised maps based on these needs. However, by requiring map-creators to choose from a fixed list of colours, I was able to indirectly force compliance with Wikipedia's colourblindness policies so that certain colours are avoided. While these two examples demonstrate ways in which this prototype was developed to address identified social requirements, these efforts were not exhaustive. For example, while alt text can be added manually to images inserted from Wikimedia Commons, this process is not automatic and so may be inconsistent. Therefore, at least in this regard, my project is not fully fit for social purpose.

A key decision that I made after deciding upon a desktop-based solution was to implement it using Python. It is therefore important to reflect upon this decision to determine whether I should continue to use it in my Final Prototype. Besides its tight integration with Wikipedia and Wikidata, a key motivation behind my choice of Python is that it is a very high-level programming language meaning that new features can be rapidly implemented by building upon pre-existing solutions. For example, I was able to the use the built-in XML processing module to edit the SVG while the webbrowser module allowed me to implement the preview in a single line of code. My use of this webbrowser module represents the key distinction between a high-level and low-level approaches to opening a file in a new browser tab; under a low-level approach, the developer would have to individually develop separate functions to interface with each major browser, with differences due to operating systems likely having to also be considered. Conversely, under a high-level language this task is abstracted to be the same regardless of platform. For example, in Python the command is simply webbroswer.open_new_tab(url). In this way, a low-level approach is useful in cases where fine control control over the way a task is executed, such as how it handles different browsers, is required. However, this comes as a tradeoff in terms of implementation time and so a high-level approach is useful for when new methods are to be quickly implemented and evaluated. In this way, my choice of Python allowed me to follow the Rapid Application Development ("RAD") programming philosophy. As shown in Figure 27, under this approach the outcome is continually refined as it is produced so that it is progressively more inline with the needs of stakeholders. The Dynamic Typing of Python and its avoidance of Header Files means that the architecture of a solution can be rapidly changed, furthering Python's compatibility with a RAD approach. It is this RAD approach that allowed me to continually consider fitness for purpose throughout the development

27 Traditional versus RAD Programming Paradigms



Source: http://www-old.novulo.com/Rad.aspx

process, a key reason that even this Initial Prototype meets many of my brief requirements and is largely fit for purpose. In terms of the physical context of my project, I therefore conclude that my choice of Python and adherence to a RAD model was beneficial and so I will continue to use these for my Final Prototype.

Functional Modelling

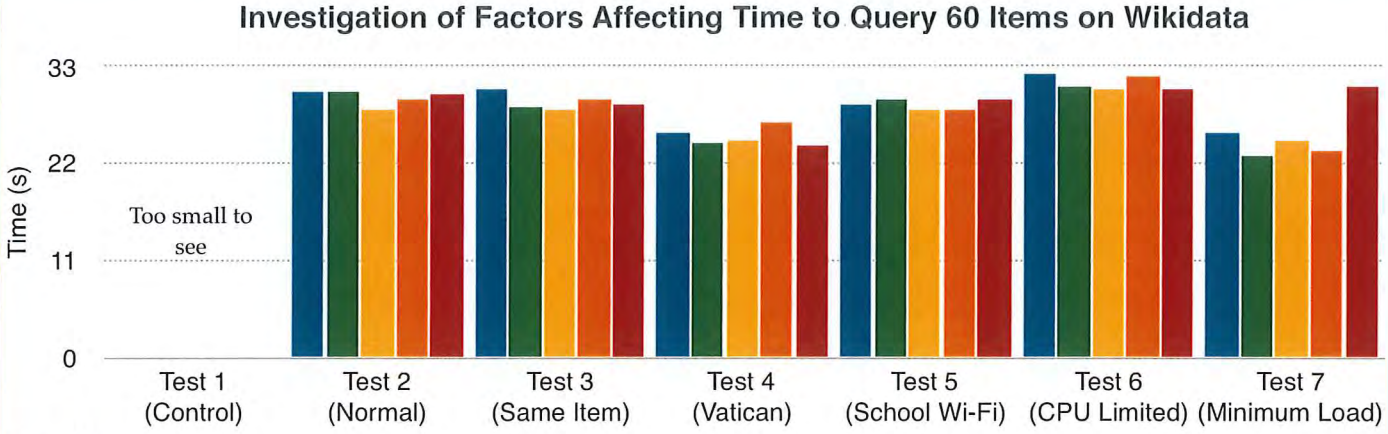
MrsAlaskaSunshine and I both observed that these scripts took a long time to run, suggesting a performance deficiency. To quantify this issue, and hence whether this prototype is fit for purpose in the physical environment, I constructed the script shown in Figure 28. This uses the builtin 'timeit' Python module that my programming teacher showed me as a way to accurately time the execution time of a piece of code. This is another example of how a high-level language and the RAD programming philosophy are beneficial to prototype development, in this case testing. Note how just 60 items are iterated instead of every single country: this meant I had to spend less time waiting for the tests to runs. The number 60 was chosen so that a total time of half a minute would correspond to each query taking half a second etc. Also note that the 'timeit' module can be used to automatically run a block of code multiple times and return just the total time; I instead executed this command multiple times so that I could see each individual result. This is important as it will allow me to gauge the variability as well as any patterns such as the first query taking longer (which would indicate caching). Different modifications of the code in Figure 28 were used to investigate different possible causes and solutions. These results are shown in Figure 29:

28 Script Used for Functional Modelling

```
import timeit

from query import *
from automap import *

print('Test X: (Description)\n')

def testQuery():
    for country in c('@world')[0:60]:
        query(country,p('headOfGovernment'))

for i in range(0,5):
    time = timeit.timeit('testQuery()',number=1,setup='from __main__ import testQuery')
    print('Test completed in '+str(time)+' seconds.')
```

Source: self

29 Results of Performance Investigation



Raw Results:

```
Test 1: Control

Test completed in 7.959499998833053e-05 seconds.
Test completed in 5.3977999414200895e-05 seconds.
Test completed in 5.3789999583868816e-05 seconds.
Test completed in 5.364500066207256e-05 seconds.
Test completed in 5.334100023901556e-05 seconds.
>>> ================================ RESTART =====
>>>
Test 2: Normal

Test completed in 30.038472333000755 seconds.
Test completed in 30.023262726999747 seconds.
Test completed in 27.946945772000618 seconds.
Test completed in 29.08143840000048 seconds.
Test completed in 29.638918208000177 seconds.
>>>
```

```
Test 3: Same Country 60 Times

Test completed in 30.309909686999163 seconds.
Test completed in 28.436261241000466 seconds.
Test completed in 28.168050512000264 seconds.
Test completed in 29.250616731000264 seconds.
Test completed in 28.59451249599988 seconds.


Test 4: Vatican City 60 Times

Test completed in 25.30789580399869 seconds.
Test completed in 24.144127325998852 seconds.
Test completed in 24.430031376999978 seconds.
Test completed in 26.54324856600033 seconds.
Test completed in 23.927143336999507 seconds.
```

```
>>>
Test 5: School Wi-Fi

Test completed in 28.57353010100269 seconds.
Test completed in 29.09114646099988 seconds.
Test completed in 28.048455029400167 seconds.
Test completed in 28.113109971000085 seconds.
Test completed in 29.054406692001066 seconds.
>>>
```

(for Test 6 see below Figure)

```
Test 7: Minimum Load

Test completed in 25.269595366476363 seconds.
Test completed in 22.77919887117391 seconds.
Test completed in 24.50836719293661 seconds.
Test completed in 23.28921618352658 seconds.
Test completed in 30.63636541411945 seconds.
```

Test 1 was a control test with all the same code as the other tests except that the call to the 'query' function was removed. It executed in under a ten thousandth of a second, proving that the performance issues do not lie in the AutoMap functions used to iterate the 60 countries or get the entity id of the 'headOfGovernment' property. This test was important to ensure the validity of the test. Note that the trials for Test 1 took so little time to execute that they cannot be seen in the graph. Test 2 represents a 'normal' test in which the code from Figure 28 was ran unmodified. While the first two trials took longer than the rest, I do not feel that this is statically significant enough to suggest a pattern. This suggests that caching does not occur. To confirm this suspicion, I ran Test 3 which queried the same item 60 times in a row. The difference between the first trial and the last four are in my opinion not statistically significant which strongly suggests that caching does not occur. Further consideration of Test 3 made me wonder whether the country that is called, in this case Andorra, affects the time taken. The only reason I could give for this is that some countries have values for their head of government whereas others do not. To test this, I performed Test 4 in which the Vatican City item was queried 60 times in a row for its head of government property, each time returning 'None'. This took a statistically-less time to complete, showing that a returned value does in fact slow down the query, albeit only slightly. This prompted me to look over my code to handle a returned value. This section merely uses an 'if statement' to determine whether a value exists and if so, decodes it from a byte array to a string. I know from experience that 'if statements' have a negligible impact on computation time. Figure 30 shows a test I used to show that the conversion of a byte array to a string is also computationally negligible. This proves that the reduced time due to an absence of a return type is not due to my program not having to process one. It must therefore either due to less data being transferred or some difference within the Wikimedia servers. To test whether the rate of data transfer is in fact the limiting factor, I ran the code used in Test 2 at my school where Figure 31 shows the average upload and download speeds to be 489 and 632 megabits per second respectively, with a ping of 12ms seconds (measured by speedtest.net). The results were are shown in the graph as Test 5. The other tests were conducted from my home computer where the internet speed averages just 13 megabits per second with a ping of 9ms. There is no statistical difference between Tests 2 and 5, suggesting that internet connection is not the factor leading to these performance issues. As shown in Figure 32, I used a script from OS X Daily[32] to run the CPU at 50% while Test 6 was performed. This resulted in a slight increase in time but I do not believe that these results are conclusive. Instead, other factors such as decreased resources to the Python interpreter could have caused the results. While not definitive, this does at least suggest that

30 Time Taken to Decode Byte String

```
>>> ================================ RESTART ================================
>>> import timeit
>>> timeit.timeit("b'Q12345'.decode('UTF-8')",number=1000)/1000
3.772839991142973e-07
>>>
```

Source: self

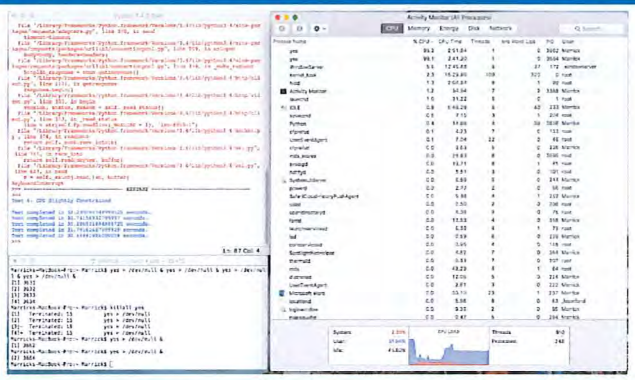
31 Internet Speeds at Home and at School

At School:

At Home:

Source: speedtest.net

32 Tests Conducted Under CPU Stress Conditions



Source: self

the performance issues are not due to computationally inefficient code. Based on research, I discovered that the Wikimedia servers have the lowest demand around 2:30am UTC, on a Monday.[33] This time corresponds with around 3:30pm New Zealand Time on Tuesday and so the next time this occurred (September 29th) I reran the script for Test 2 to yield Test 7. There does appear to be some decrease in the average time per query but this difference is only slight. Therefore, more efficient utilisation of server resources through time-shifting will not suffice alone to address the performance issues of my project.

Based on these experimental results, I feel that my prototype is not fit for physical purpose due to its poor performance under typical computing environments, let alone the low-powered computers my project must also be fit for purpose on to fulfil the Foundation's goals and values. Therefore, improving performance will be a major focus in the production of my Final Prototype; Consideration will also be made as to how the social fitness for purpose can be improved based on my analysis of stakeholder feedback.

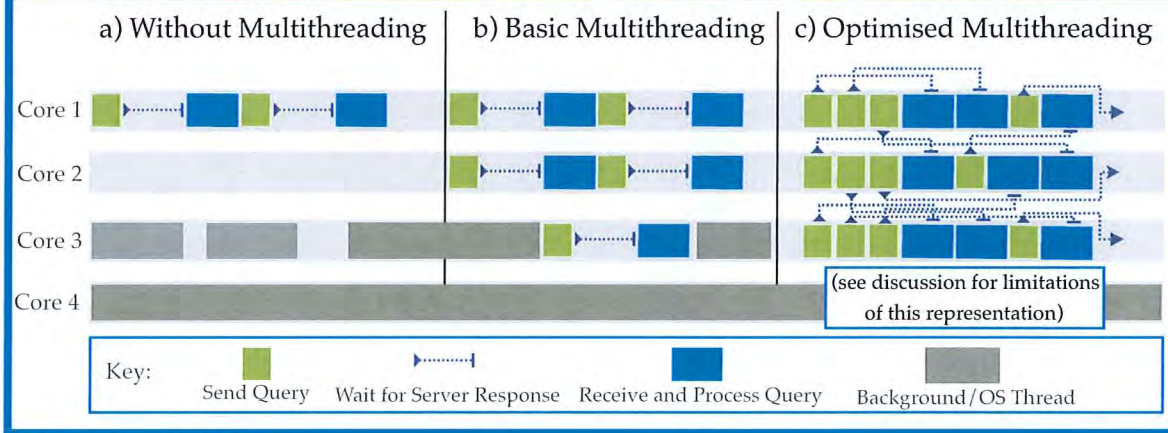
Final Prototype Development

Conceptualisation

After analysing my stakeholder feedback, self-reflection and performance investigation I feel that I have a very good sense of what the major limitations of my project are in terms of the both the physical and social contexts. I believe that the performance issues are the most significant as the lead time between editing a script and seeing the result makes it more time intensive to develop and test scripts. This contributes to the broader issue of a general lack of user-friendliness to the editor, compromising the needs established in my stakeholder analysis to combat the general trend of declining editor numbers. Another key issue that will be addressed for the Final Prototype is that provisions for user accessibility, such as alt text, could not be implemented due to limitations of the Initial Prototype.

The general conclusion that I drew from from my performance investigation was that the issues I faced due to performance were almost exclusively due to factors beyond my control, namely how Wikimedia servers handle Wikidata API requests. It is therefore impossible to optimise the actual queries; I must instead optimise the way in which they are requested. One way to do this would be though caching in which previous results are stored locally for future use. The main reason that I avoided caching in my Initial Prototype is that the Wikidata API does not include any provisions to notify the client when an item is changed and so will be at least partially out of date, even if it is continually refreshed.[34] Therefore, any attempt at caching is a tradeoff as performance increases but accuracy likely decreases due to reduced data currency. Despite this, I feel that the unique context of my project means that caching actually could be viable. Firstly, most maps are constructed by iterating through the countries and so only a finite of items would have to be store. Secondly, most properties about countries such as their heads of states and capital cities rarely change and properties such as population are generally only updated on Wikidata once per year; Therefore, the rate at which cached information becomes out of date is low. Another key feature of my physical context that suggests caching will be viable is the fact that due to the design of the Wikidata API, Pywikibot can only return all of the properties of an item, not just the ones required. Without caching, the required property is found within this result and returned. This means that if the same item is separately queried for different properties, an approach without caching would cause multiple queries to be sent, a key inefficiency in terms of performance. Conversely, a caching approach in this context would simply search through the result from the first query, although this time looking for a different property. Based on these considerations, I have decided that I will implement caching in my final project.

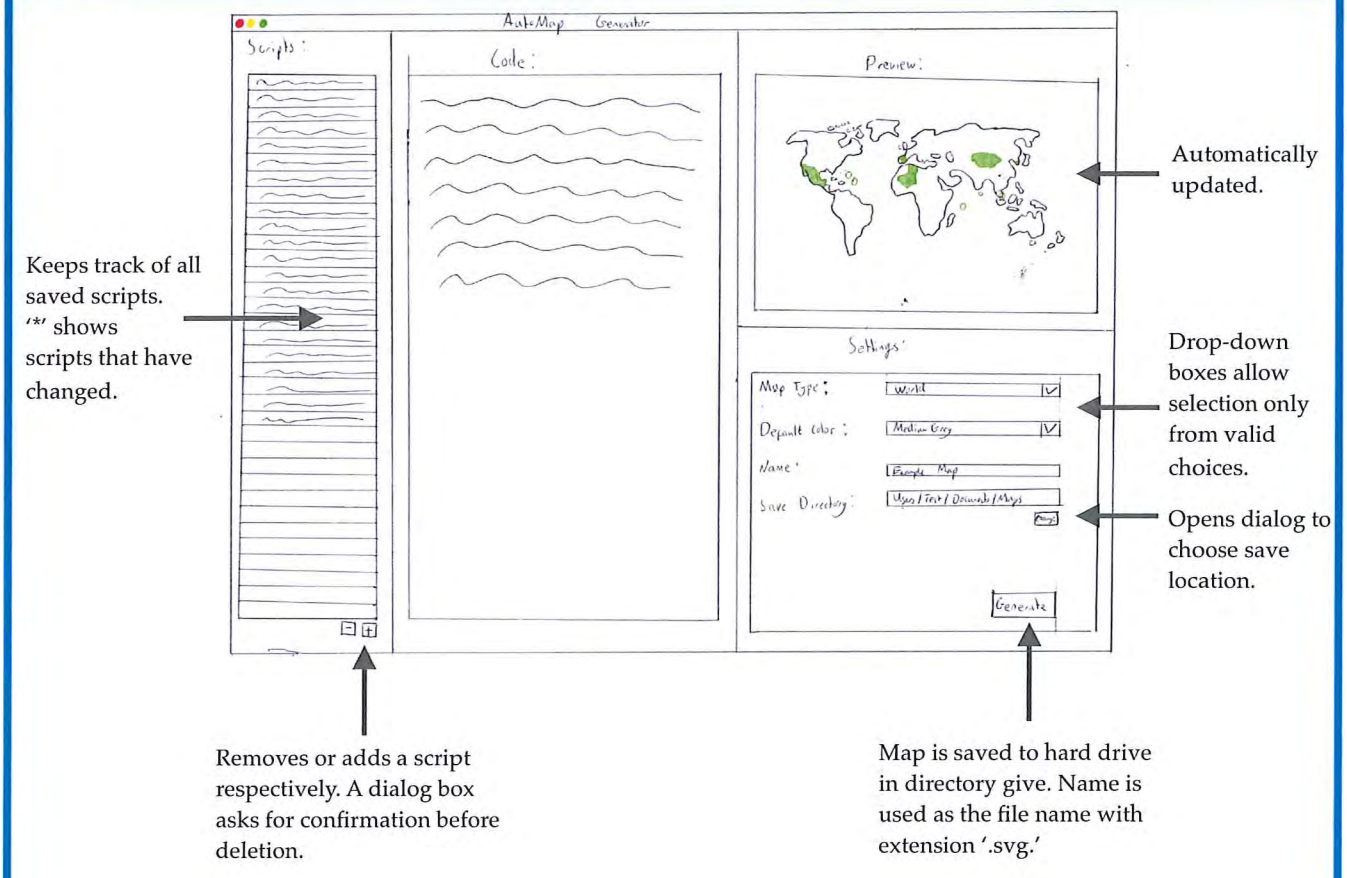
33 **Conceptual Diagram of Possible Multithreaded Caching Approaches**



Source: *created by self*

While caching will likely yield considerable performance gains, an improvement to the way in which I query Wikidata would still be desirable as it would allow me to update the cache more frequently. This would make the data more current and so ensure that my project fulfils the needs of my social environment with regard to providing a up-to-date source of information. Research into multithreading and parallelisation show that there is limited but sufficient support for these paradigms in Python.[35] As conceptually shown in Figure 33b, this will theoretically improve performance by performing multiple queries at once. Figure 33c show an idea I had as to how performance could be improved even further if threads continued sending queries while they were waiting for their response. In reality, this depiction is oversimplified as an entire thread would likely have to be dedicated to checking if a query has been received and to cache any that arrive too soon. However, a key trend of computer science is research into multithreading and the broader concept of parallelism; there is therefore likely a more efficient scheduling algorithm exists to allocate tasks to individual threads. For example, it would aim to minimise the inter-thread communication that I inadvertently overused in Figure 33c as this would lead to thread synchronisation issues.[36] While implementing this Optimised Multithreading would undeniably improve performance, this would require low-level modifications to the inner-workings of the Pywikibot framework. I have neither the time nor the skill necessary to do this but I feel that caching and Basic Multithreading (Figure 33b) will be sufficient to ensure the performance, and hence physical fitness, of my project.

34 GUI Mockup Annotated with Functionality

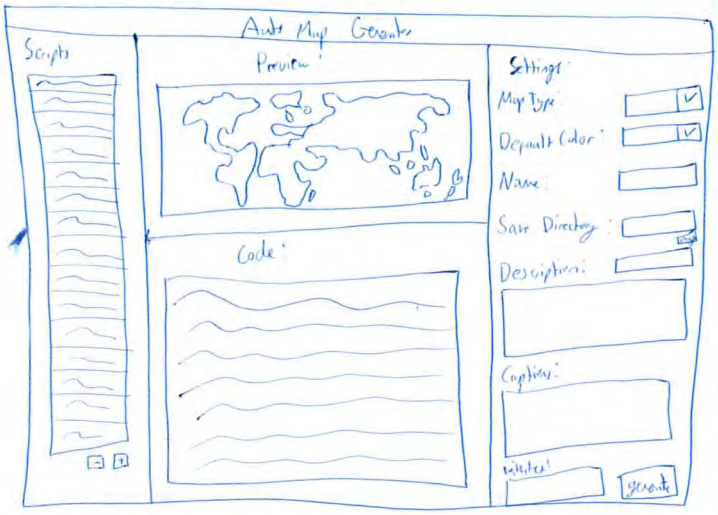


Based on MrsAlaskaSunshine's experiences, I feel that my previous assumption that a graphical user interface ('GUI') was not needed was in fact incorrect; Even after Python had been installed, she struggled with the process as I had to explain to her how to use the built-in Python editor (IDLE),

where to save the scripts and where the final maps get saved. A GUI could bundle all of these tasks into one environment. I anticipate that this would make my project substantially more user-friendly, hence ensuring the needs of my stakeholder. Figure 34 shows an annotated mockup of a potential GUI for my project. Two key features of this are the ability to choose a save directory as well as the 'Scripts:' pane on the left hand side to keep of the scripts together; these features address the issues MrsAlaskaSunshine faced at finding scripts and locating their output while using my Initial Prototype.

Shortly after producing this GUI mockup, I came across a community discussion in which several Wikipedia editors were complaining that the current method of adding captions and keys (like that seen in Figure 05) is cumbersome as it must be manually added to each page that the map is used. Based on this, I sketched a second GUI mockup (Figure 35) in which the user can graphically select both a caption and a key. This is automatically formatted into wikitext which can then be copied into articles. Wikipedia automatically adds the content of an image's caption to its alt text and so this approach will ensure that the maps created are accessible to vision impaired readers, hence ensuring that the values and policies of the Wikimedia Foundation are met.

35 **Second GUI Mockup showing Captions**



Production

To verify that caching will indeed give performance gains, I performed the Functional Modelling investigation shown in Figure 36. This acts as a wrapper to the query function and locally stores the result. Note that cache is not stored upon closing and so this implementation is useful only for the purposes of investigation. Figure 36a shows a copy of the script used for my pervious investigation that has been updated to use this cached query. Note how testQuery() is called once before the timing begins: this ensures that the cache is loaded so that the time results are purely based on querying the local cache. Figure 36b shows that the time this takes is trivial, highlighting the performance gains of caching when that cache is store in the RAM. Figure 36c shows a test I performed to evaluate the time taken to retrieve 60 queries from a cache and then use AutoMap to generate an SVG map and save it to the hard drive. Figure 36d shows that this entire process is completed in under a hundredth of a second. While this doesn't take into account the time to render an SVG map, I believe that this result

36 **Functional Modelling through Caching Investigation**

Purely Queries:

a)

```
import timeit
from query import *
from automap import *

print('Tests With Caches')

def testQuery():
    for country in cc['World']['data']:
        customQuery(country,p['healthGovernment'])

testQuery()

for i in range(0,5):
    time = timeit.timeit('testQuery()',number=1,setup='from __main__ import testQuery')
    print('Test completed in '+str(time)+' seconds.')
```

b)

```
Test: With Cache

Test completed in 1.968099968507886e-05 seconds.
Test completed in 3.627499972935766e-05 seconds.
Test completed in 2.3069995222613215e-05 seconds.
Test completed in 4.4665997847914696e-05 seconds.
Test completed in 3.5464996472001076e-05 seconds.
```

Cached Queries with Map Generation:

c)

```
import timeit
from query import *
from automap import *

print('Tests With Cache and Map')

def testQuery():
    map = dataMap('world','Cache Test',colour='default')
    for country in cc['World']['data']:
        result = cachedQuery(country,p['healthGovernment'])
        map[country] = colCon(result)
    map.generate(True)
testQuery()

for i in range(0,5):
    time = timeit.timeit('testQuery()',number=1,setup='from __main__ import testQuery')
    print('Test completed in '+str(time)+' seconds.')
```

d)

```
Test: With Cache and Map

Test completed in 0.008185480000975076 seconds.
Test completed in 0.007838665995222982 seconds.
Test completed in 0.009492768003838137 seconds.
Test completed in 0.008714952004083898 seconds.
Test completed in 0.009340978002001066 seconds.
>>>
```

shows that it will be feasible to implement a live preview at least in terms of the physical performance needs.

To actually implement the cache, I first considered using a database. Advantages of this include greater scalability as well as better handling of situations that could occur when I multithread my program. For example, if one thread attempts to modify the cache while another thread is reading it. However, I have no database programming experience and so found the process confusing, especially since a specialised query language (SQL) has to be learned and used.[37] I therefore instead decided to simply read and write the cache to and from a text file. My implementation of this is shown in Figure 37a. Note how tempCache is used throughout the openCache() function so that the actual 'cache' variable is only changed at the last possible moment. Note that in Python, dictionaries are implemented as primitive types that are passed by value, not by reference; it is therefore not necessary to explicitly set 'tempCache' as a copy of 'cache'. Figure 37b shows the outputted cache; at this stage the function to update the cache of all countries had not yet been implemented and so this merely shows the properties that had been accessed through my functional modelling investigation. Note that the 'None' type (caused when no value exists) and multiple values (some Wikidata items have past and present heads of government listed) are two special cases that are both correctly handled.

37 Actual Implementation of Caching

a) Code:

```
cache = {}
def openCache():
    f = open('cache')
    tempName = ''
    tempCache = {}
    for tempLine in f.readlines():
        line = tempLine.strip()
        if line[0] == '~':
            tempName = line[1:]
        else:
            property = line.split(':')[0]
            values = line.split(':')[1].split(',')
            if values == ['None']:
                tempCache[(tempName,property)] = [None]
            else:
                tempCache[(tempName,property)] = values
    global cache
    cache = tempCache
    f.close()

def updateCache():
    #Not Yet Implemented
    pass

def saveCache():
    f = open('cache','w')
    tempCache = cache

    records = {}
    for record in tempCache.keys():
        if record[0] not in records:
            records[record[0]] = [record[1]]
            print(records)
        else:
            print(records)
            records[record[0]].append(record[1])

    for item in records:
        f.write('~'+item+'\n')
        for property in records[item]:
            line = property+':'
            values = cache[(item,property)]
            for value in values:
                line += (str(value)+',') #str(None) returns 'None'
            f.write(line[:-1]+'\n')
    f.close()
```

b) Cache (Extract):

```
~Q25362
P6:None
~Q225
P6:Q57844
~Q750
P6:None
~Q262
P6:Q57901
~Q25228
P6:Q19831099
~Q241
P6:None
~Q962
P6:Q29035,Q3241966
~Q977
P6:Q57683
~Q916
P6:None
~Q963
P6:Q57449
~Q781
P6:Q57602
~Q96988
P6:None
~Q31
P6:Q950958
~Q35
P6:Q57652,Q182397,Q46052,Q311063,Q327190,Q312881,Q348901,Q
312881,Q441330,Q727237,Q441330,Q918287,Q721358,Q503087,Q71
0950,Q503087,Q1335084,Q664052,Q731504,Q664052,Q367557,Q982
328,Q367557,Q1375094,Q1375737,Q1744787,Q1040851,Q264340,Q1
040851,Q1874069,Q1375094,Q1274220,Q572189,Q1576188,Q163499
2,Q1375050,Q1343877,Q966436,Q1343866,Q782403,Q947911,Q6940
93,Q725118,Q1037599,Q725118,Q947789,Q591189,Q491280,Q94791
1,Q351178,Q182397
~Q1009
P6:Q57840
~Q902
P6:Q52183,Q52183,Q234814,Q234814
```

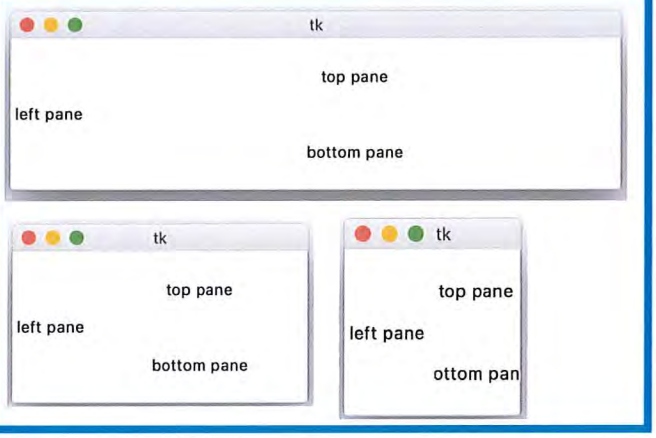
Based on the aforementioned research, I knew that I would have to use the built-in library 'multiprocessing' library in which each thread is ran as a seperate process. By following the offical python documentation, I was able to implement the updateCache() so that it was multithreaded. To test the performance of this new approach, I modified updateCache() so that it only updates the cache of the first sixty countries and hence its results are comparable to that of my investigation (Figure 29) where no multithreading was performed. These results are shown in Figure 38 and suggest that the time taken is between a half and two-thirds of the time without multithreading. This test was performed performed on a quad core computer so theoretically the time decrease by a factor of 4. However, in reality the performance gains are less than the theoretical due to overhead within the 'multiprocessing' library as well as the use of cores by the OS and other processes. This idea that other processes limit the number of cores that can be used by my function is supported by the fact that the data from this test is a lot more variable than in my previous investigation before multithreading was added.

38 Multithreading Test

```
>>>
Test: Multithreading

Test completed in 18.662017446339384 seconds.
Test completed in 17.0837726324683 seconds.
Test completed in 13.651817676465036 seconds.
Test completed in 14.150358832704859 seconds.
Test completed in 15.155450736375293 seconds.
>>>
```

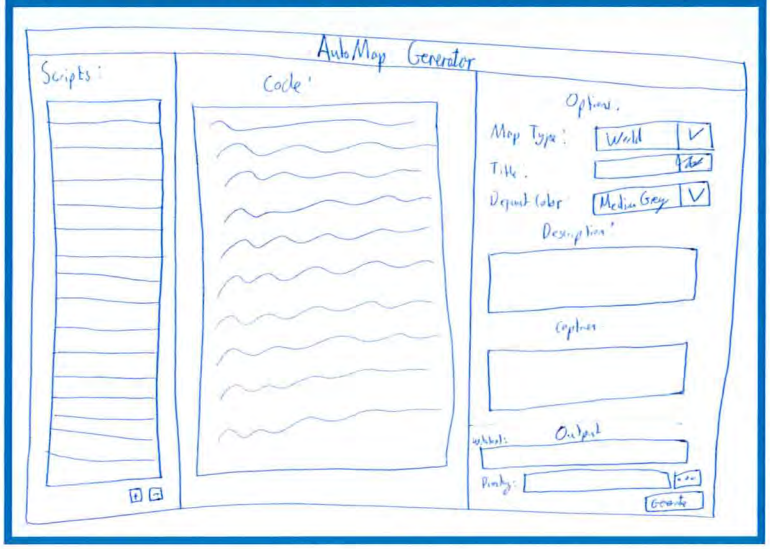
I previously used the tkinter module to create a GUI for a programming internal and found that it was exhaustive and well documented by sources such as at effbot.org.[38] I therefore decided to use this module to implement my GUI design from Figure 35. Note that keys are specified as a set of lines, each containing a '|.' Each line corresponds with an entry in the key whereas the '|' separates the colour and the corresponding text. This approach shows two key considerations for ease of use to ensure fitness: friendly colours can be used while the '|' character was used as it will be familiar to my target audience due to its use in the template syntax. The development of this GUI was largely routine with the only notable issues faced were the need to operate on many different screen sizes plus the actual rendering of an SVG image. I had heard from previous experience that the tkinter includes a PanedWindow widget for managing complex GUIs but I had never actually used it. Using the documentation from effbot.org, I created the test project shown in Figure 39 to learn how to use this widget. Note how the left-most pane always remains the same size; this is achieved by setting "stretch='never' " during m1.add(). (Note that Figure 39 shows that the left-most pane of my final GUI should not resize) While this project deals only with PanedWindows that contain labels, more versatile GUIs are created through nesting. Research into ways to include SVG images within tkinter brought me to the 'canvas2svg' module which extends tkinter's built in canvas widget. Note that this module is freely released under the BSD and this component is suitable in the regard that it can be distributed with my project.[39]. However, as previously described, the SVG maps produced through this project use CSS styling which is a part of the SVG standard that is compatible with MediaWiki (and hence Wikipedia) but not this module. While I could update my map generation code to use standard SVG, this would cause serious performance issues due to the need to iterate multiple times through the entire file (see page 18). I researched other modules that can also display SVG within a tkinter canvas but found none that could handle CSS styling. According to effbot.org, the tkinter canvas can display images through the

39 Functional Modelling of Tkinter Resizing



PIL which in turn is able to handle raster formats such as PNG. This lead to me to consider ways in which I could convert an SVG to a PNG and then open that PNG into tkinter. One possible approach would be to send the SVG to a web service that then sends back the rasterised PNG image. I believe that the need to continuously transfer images across the internet and wait for them to be rasterised would lead to significant performance issues and hence the 'live' preview would not be particularly responsive. It would also require significant server resources, impairing the scalability of my project. I therefore believe that at this stage it is not feasible to implement a live preview directly within my GUI. However, as noted previously, my Initial Prototype was able to provide some visual feedback by opening an web browser to display the produced map. This lead me to consider whether it would be possible to use the same approach to open a new tab and then use Python to refresh that web page when a new preview has been generated. Research into whether this is possible lead me to a StackOverflow discussion thread in which a Python user had a similar issue.[40] One commenter suggested that instead of trying to refresh the html file through Python, Python could be used to open an html file that has a script embedded within it to automatically update its self periodically. I therefore modified my map.generate() function with an additional parameter which, if set to true, causes the SVG map that is generated to be embedded in an html file along with a self-refreshing script I found on another StackOverflow discussion thread.[41] Since the SVG map is embedded as a reference to a hard drive location, updating the SVG map will cause the new map to be displayed once the page is automatically refreshed. Therefore while the live preview is not actually within the GUI, the GUI window can be placed next to the web browser window for a comparable result. Figure 41 shows a rough sketch I made to re-conceptualise my GUI without an internal live preview; the final GUI produced based on this is shown in the 'Outcome' section.

41 Third GUI Mockup without Internal Live Preview



The final step in the production of my Final Prototype was to actually implement the live preview so that the code written in the text field would be executed, causing the map and hence preview to be updated. As noted in my discussion of why I chose to use Python, it is a dynamic interpreted language and so I knew that I could pass the code as a string to the exec() function. Since every valid script must invariably include "from query import *", "from automap import *" and "map.generate()" I decided to automatically add theses to the exec() call so that they do not have to be added by the user. The initialisation of a dataMap object is also automatically added based on the parameters specified in the GUI. I feel that this is important to ensuring that it meets the needs of my editor stakeholders, especially given the fact that the majority do not know how to program and so minimising the amount of code they have to write should reduce error. Since exec() runs the code within the same context of the main application, I must be careful that an error in the user's code will not bring down my entire project. Figure 42 shows some test code that I executed within the Python

42 Error Checking Test

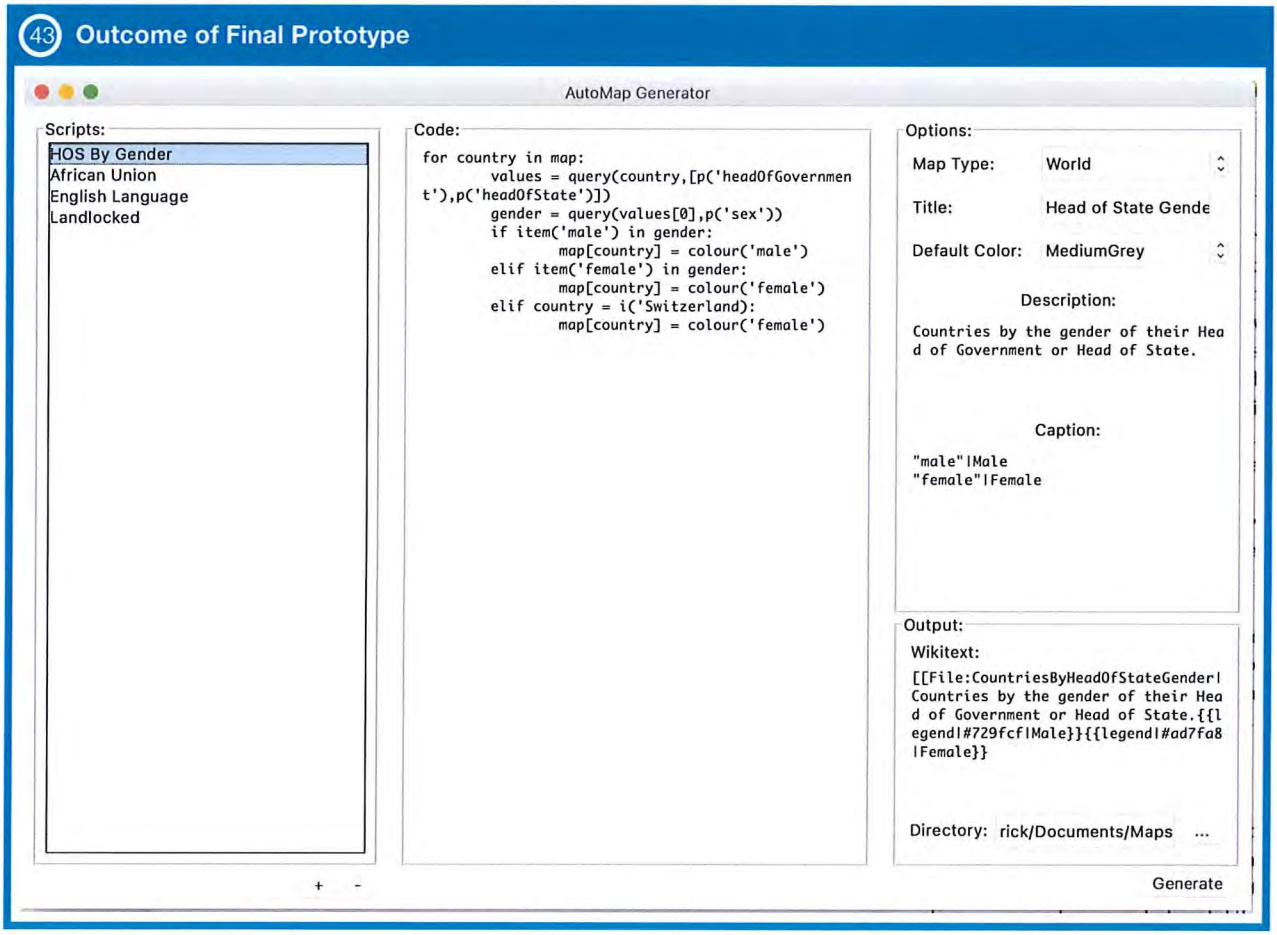
```
>>> try:
        exec('0/0')
except Exception as e:
        print(str(e))


division by zero
```

shell to confirm that errors within the code executed through an exec() command are not only handled but also that a description of the error can be received. I added an extra label to my GUI to show this error message, to providing the user with feedback as to why there script may not be working. Another key issue I faced while implementing this live script execution was that even if a user's script doesn't crash, it could still impair the broader program if the program cannot perform other tasks while it waits for the code to execute. To prevent these performance issues, I placed the call to exec() in a separate thread using the same approach as my parallelised cache updater. Updating the live preview too frequently would lead to frivolously high CPU and disk usage and so I decided that the live preview would be updated every 10 seconds. Note that the call to map.generate() is always the last function to be called and so the map will only be updated if the code actually runs. This means that the last valid map will always shown, improving the user-friendliness since it prevents cases in which the amp is automatically generated while the user is in the middle of typing a line of code, causing no map to be displayed for at least 10 seconds. This would impair the user's ability to perform the very intent of the live preview: make refinements to the code based on issues seen visually. As noted in my documentation, this property of only updating when the code is successfully ran provides a roundabout way to prevent the live preview from updating; placing a line of code that is guaranteed to cause an error, such as "0/0" will 'freeze' the live preview. This would be helpful in cases where the user is making changes but wants the previous preview to remain for reference.

Outcome

Figure 43 demonstrates the full implementation of my Final Prototype. In this screenshot, the 'Generate Map' button has just been pressed and so the SVG map was saved to the specified directory and the 'Wikitext:' field was populated with compliant wikitext based on the map's title and caption. For a commentary on what each UI feature does, please refer to my GUI mockups.

43 Outcome of Final Prototype



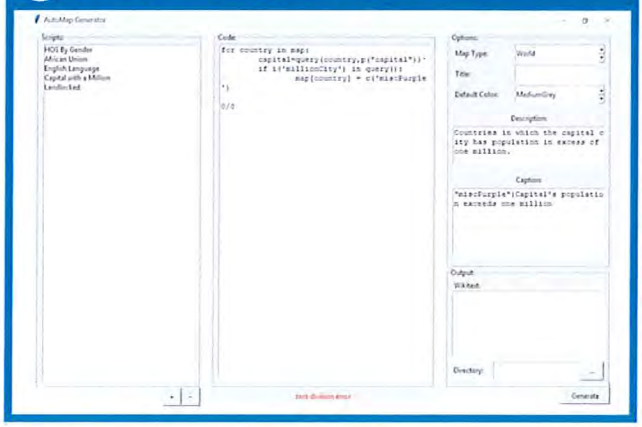
Final Evaluation

Client Re-Visitation

As previously noted, I felt that my previous client re-visitation was fundamentally flawed as its small sample size meant that it was not representative of the entire stakeholder group. I therefore decided that the use of many users with different background would be the only way to fully evaluate the social fitness of this Final Prototype. Since MrsAlaskaSunshine has already had experience with my Initial Prototype, she is not a valid representation of a first-time user and hence her responses are not necessarily reflective of my project's ease of use. However, I was able to ask her to compare my Initial Prototype with my Final Prototype so that I could see whether or not the changes I implemented actually addressed her feedback and the broader stakeholder needs that I identified. When asked whether she felt the GUI would make it easier for first time users she said, "without a doubt. Being able to do things visually makes it so much more easy to learn." When asked about whether to comment on the implications of the live preview being in a web browser, she said that it "required some juggling" to line the browser window with the GUI but besides this, it was fully functional. She did however note that placing the preview in the the web browser did make it harder for her to compare the preview map with other web pages such as to manually check its accuracy by comparing it to a preexisting map on Wikipedia. Despite this, it is of course possible to open two browser windows and so I do not feel that this compromises the fitness of my prototype. When asked whether she feels that it meets the needs of Wikipedia editors who create maps she said "I believe it does." I also asked whether she would continue to use it to which she replied, "I'm on a bit of a hiatus right now but if it's still around when I get back, i'll definitely use it again."

To gain a better sense of how a first time user would react to my prototype, I reached out to editors that I had previously directly worked with. When asked if they had coding experience, one had "dabbled here and there with Python" but has significant JavaScript experience; the other two had no programming experience. By this stage I had already created a brief but comprehensive set of documentation that covered installation, use of the GUI and scripting. I asked all three users to produce countries in which the capital city is a 'city with millions of inhabitants.' I chose to give them this task as it requires them to perform two steps: query Wikidata for the capital city and then query the returned item to determine if it contains 'city with millions of inhabitants as a value'. While somewhat advanced, this is fully documented with example code in the documentation I provided. However, to gauge the true user-friendliness of my approach I opted to set a task that is not documented and hence must be intuited: generate a map of countries based on the number of sister cities its capital has. The user who had 'dabbled' with Python was apparently able to "quickly and easily" implement this whereas the other two had more difficulty. Figure 44 shows a screenshot that he took of his final script. This appears to have been ran on what appears to be either Windows 8 or Windows 10, hence confirming that my approach is indeed cross platform. The dated appearance of this UI is an unavoidable consequence of using TKinter, suggesting that an alternative GUI library could be used for future versions. (Note that this screenshot was originally sent to confirm exec() error handling on Windows hence the '0/0'; The generate button had not yet been pressed so no Wikitext is displayed). After telling the two non-programmers that "len(values) will give the number of values" and that

44 Final Outcome in use by User



"Mathematical comparison signs such as > or >= can be used in the 'if' statements", they were both able to implement the second map. As previously noted, if implemented across Wikipedia there would be discussion pages where users could get help and so I do not feel as though their difficult represents a lapse in fitness for purpose. The user with programming experience disliked the lack of syntax colouring in the editor and noted that the error messages given in the GUI do not show the exact piece of code that caused the error or even the line number on which it appeared. I do not feel that this would have a drastic effect on a user's ability to troubleshoot a script written for my project. However, the option to edit the script in IDLE then reopen it in my GUI remains for any editors who feel that the tools given are constraining. Since the live preview is implemented within the browser, it would still be accessible to users who choose to use IDLE. It is important to note that none of these editors were members of WikiProject Maps. However, as general Wikipedia editors I feel that they still reflect the needs of my stakeholders. I also do not feel that the fact that all three of them are experienced Wikipedia editors had much effect on the results as I do not feel the skills needed to use my project correlate with editing ability. Based on these client re-visitations, I conclude that my project is fit for social purpose at least with regards to fulfilling my stakeholders' needs for ease of use.

Reflection on Stakeholder Needs

Ultimately my project can only be considered to be fit for purpose if it meets the majority of the needs of each of my stakeholders; I do not believe that this is the case. While my client visitation shows that my project largely meets the needs of my editor-stakeholders, this does not necessarily mean that my approach is the most desirable with regard to their needs. This lead me to reflect on whether, my decision to reject a web based solution was a mistake. I feel that if I implemented this successfully, the outcome would have a greater fitness for purpose in terms of the both the social and the physical context. For example, while my Final Prototype does technically meet all the requirements listed in my brief, I feel as though the time taken to install both Python and my project limits its audience to those who are committed to creating a large number of maps and so have researched the various options. Conversely, a web based approach would allow any user to simply click a button on an article they are editing. This would encourage broader participation in the map creation process and also allow minor changes, such as the colours of countries, to be made more easily. However, I stand by my assertion at the time that a web based would not have been feasible due to both time and skill constraints. I therefore must conclude that the choice of a desktop based solution was most suitable in my personal context but should my project be deemed useful, with all current indications suggesting it will be, I would recommend to the developers of the site that a web based implementation should be produced if resources suffice.

Due to time constraints, I wasn't able to internationalise my GUI so that the names of sections appear in a user's native language. While maps can be seen by users of any language, this constrains my audience of editors to those who have at least a basic understanding of English. While I could have easily implemented a language choice option to determine which strings are read from a text file, I opted not to as I had no means to actually know what the names were in different languages. However, as my project is deployed across Wikipedia it will attract users who speak multiple languages; I will therefore be able to approach them for assistance in translating the GUI. I do not have experience with internationalisation in Tkinter but due to its support for Unicode, it should be able to handle non-English characters. However, I am concerned that the need to handle right-to-left languages such as Arabic could require some modification to my layout.

Along with caching, another key tradeoff I made during the development of my solution was to require editors to upload the SVG maps themselves, as opposed to automatically by the program.

Figure 08 where I linked the trend of declining editor numbers to the learning curve of many editing tools on Wikipedia. This need shaped the choice of my solution and formed the basis of my evaluation of my Initial Prototype and Final Prototype. I can therefore definitively conclude that this value has been met to the greatest extent possible and so, in general, I feel that my project meets the values of the Wikimedia Foundation. Therefore, from their perspective, it is fit for social purpose. I therefore do not feel that my project would compromise the strategic efforts of the Foundation such as their attempts to maximise fundraising through voluntary donations. Like the developers of the site, the Foundation also holds a major stake in the physical fitness of my project. However, it has already been established that I do not anticipate a major impact on server load, and any major security and stability concerns that could affect their sites have been prevented through my choice of a desktop based solution. I therefore conclude that my project meets all foreseeable needs of the Wikimedia Foundation and so in this regard is fit for purpose.

Fitness in the Broadest Sense

Besides the direct needs of my stakeholders, it is also important to evaluate my project in terms of its broader purposes based on the key technological trends it is a part of: the push towards a semantic Web 3.0, and the democratisation of information. In terms of Web 3.0, I believe that my project is a merely a small step, albeit one in the right direction; I feel that my project is philosophically in line with this movement as it provides a simple way for users to integrate information from diverse fields. However, in practice is not a successful implementation of Web 3.0 as it doesn't utilise its open standards such as the aforementioned RDF and SPARQL. This means that they project can only be used with Wikidata and hence it misses out on what I believe is the core opportunity of the semantic web: the ability to integrate multiple source of information that are not intrinsically linked. Fitness for purpose in the broadest sense includes how the purpose of a project will change over time and whether it can be applied to other contexts. In one way, my project's close adherence to the values of Wikipedia, as well as its high Sustainability and Maintainability, means that it will likely continue to have a purpose. However, I feel that by not using RDF and SPARQL it will not be compatible with other Web 3.0 movements and so I do not feel that my project is applicable to other contexts such as the spatial presentation of information from other sources. Another key trend that my project is a part of is the democratisation of information. I feel that my project is highly successful in this regard as while it doesn't provide any new information, it allows existing information to be presented in more meaningful ways. This is arguably actually more useful as I feel that the issue with Wikipedia is not a lack of information per se but that it is often presented in a way that overwhelms the reader. In this way my project can also be seen as part of the movement towards solving Big Data: the concept by which the amount of available information exceeds the ability for it to be processes and understood by humans.[42] Based on this, and the considerations of accessibility made throughout my report, I conclude that my project is a highly successful step towards the key value of this movement: the idea that information should have the greatest possible value to the greatest number of people. The importance of these key movements was hinted upon towards the beginning of this report through reference to a medical journal discussing how Wikidata could be used to ensure the universality of health care; While important in on of itself I would like to use this to emphasise why I believe that these movements are so important to not only Wikipedia but also broader society: we cannot anticipate the ways in which data will be used under the semantic web; the power is truly in the hands of individuals to curate data in new was to affect great social change.

An evaluation of Sustainability and Maintainability is important as it reflects whether my project's fitness for purpose will remain in the long term. In terms of Sustainability, the API of Wikidata is in its early stages of development so continually changing as the project grows. As was discovered through Figure 21, this may in turn break the version of Pywikibot that is shipped with my project and so the

need to continuously update Pywikibot could impair the sustainability of my project if new version of Pywikibot are no longer produced. (Note that Pywikibot is under an open licence and so it is permissible to bundle it with my project) However, in this case, an alternative that is compatible would almost certainly be present due to Python's frequent use with Wikipedia. This represents yet another advantage of the RAD approach used throughout my project: integration with a new library would require only modification of my high-level functions, a task made due to the aforementioned advantages Python brings to RAD development. Tkinter is a stable module and while new versions of Python may affect the sustainability of my project, changes are usually small and generally made to additional modules, not core functionality. However, Python's popularity also means that any changes are well documented both officially and by community members; I therefore conclude that my choice is highly-advantageous to the sustainability of my project. Considerations have also been made throughout my project as to who scalability can be assured. For example, this was a key reason behind my rejection of an approach using the Graphoid service. I therefore do not believe that scalability concerns will as my project's number of users increases and so I do not anticipate that this will have an effect on sustainability. As previously noted, Python is a high-level language in which calls to system functions are wrapped in broad, high-level functions. This leads to another sustainability advantage of using Python: my project's code does not need to take into account new operating systems or browsers, such as those used for the live preview, and simply updating a user's Python version will ensure future compatibility. The popularity of Python, particularly among the Wikipedia community, means that my project is highly maintainable as users of the site with programming experience. This is only possible due to the open-source nature of my project, reflecting a key importance in terms of maintainability of the Wikimedia Foundation's value of 'openness.' Another key programming philosophy that I used throughout my project is Object Orientated Programming in which code is logically divided into objects called 'classes.' This is extremely beneficial to maintainability as it means modifications can easily be made through polymorphic inheritance in which behaviours of an original class are modified without completely rewriting it in entirety. For example, the ability to add airport codes (see Figure 09) could be implemented through inheritance by overriding the dataMap.generate(...) function. The functions for querying Wikidata and those for presenting an SVG map are in different classes called Query and AutoMap (of instance type dataMap.) Therefore, a completely different form of data visualisation, such as a graph, could be added through the creation of a new class which would still be able to access the functions of Query. Therefore, I ultimately conclude that my project is both Sustainable and Maintainable and hence I anticipate that it will continue to be fit for purpose in the foreseeable future.

While certain enhancements could of course still be made, I feel as though all the requirements of my brief have been met and the strategic outcomes identified in my stakeholder analysis, such as the need to address the trend of declining editor numbers, have been mostly implemented. Perhaps even more importantly, my discussion of Ultimate Disposal, Sustainability and Maintainability show that this fitness for purpose will continue into the future. I will therefore begin finalising the documentation and introducing my project to more Wikipedia editors so that broad distribution and use by editors can begin shortly.

Future Enhancements

While my project supports this on a code level, time constraints meant that I did not actually implement the ability to maps of specific regions, not just the entire world; this is an obvious future enhancement that could be made. Another logical step would be to implement other forms of information presentation such as graphs. I also feel that there are many opportunities to improve the editing experience afforded by my GUI. For example, one of my stakeholders noted the lack of line numbers while I feel that syntax highlighting would be another major improvement. Figure 44 shows

that neither indentation nor indentation are well handled; I anticipate that addressing these would be difficult as it would likely require either significant modification of the existing 'Text' widget or the creation of an entirely new one. I also feel that handling errors simply by giving a generic description is not enough: showing which line the error occurred on and potentially linking to specific documentation as to common causes would be significantly more user friendly and hence improve the future fitness of my project. I however anticipate that this would be very difficult to actually implement as the exec() command takes a block of code and executes it at once, returning the result and outputting a generic description of the error. A possible approach would be to automatically break the code into independent 'chunks' that do not depend on each other. The exec() command could then be called separately on each 'chunk,' proving more context as to where specifically an error may have occurred.

In terms of the broader context of my report, I feel that a better integration with other web services both within and outside of the Wikimedia Foundation. While the need of internationalisation has been referenced throughout this report, I found that this would difficult to implement as I neither speak multiple languages nor have access to a translator. However, storing the lists of items/properties/collections/colours and the translations of the GUI on a special Wikipedia page would allow these to be edited by the wider community. Besides allowing for users to contribute translations, this broader participation would have other benefits such as the ability for colours to be changed globally as conventions change, helping to ensure my stakeholders' need for stylistic consistency. As previously noted, automatic uploading would have advantages in terms of maintainability as it would ensure that maps are updated even if editors are unaware that they have changed. (My final prototype uses an '*' to denote maps that have changed but editors may not notice this or not bother to re-upload.) To address my previous concerns as to overwriting other users' manual changes, I could either use a template to mark maps which are automatically changed or implement a check of who the last edit was made by and hence force the user of my project to manually upload the map if it was last edited by another user. I feel that either approach is concordant with the values of the Wikimedia Foundation and I don't anticipate that either would cause performance or security issues that could compromise the physical fitness of my project. The use of Wikipedia Labs to further improve maintainability remains an option but based on my previous analysis I conclude that the consequences in terms of security and stability preclude future implementation of this. Further integration with other websites beyond the Wikimedia Foundation, such as the broader Web 3.0 semantic web movement, should be a long-term goal of my project but my previous analysis of the pre-existing solutions proves that this would be difficult.

Bibliography

[1] https://www.wikipedia.org

[2] https://en.wikipedia.org/wiki/Wikipedia:Purpose

[3] https://en.wikipedia.org/wiki/Auckland

[4] https://de.wikipedia.org/wiki/Auckland

[5] http://www.jmir.org/2015/5/e110/

[6] https://en.wikipedia.org/wiki/Help:Wiki_markup

[7] https://www.mediawiki.org/wiki/Lua_scripting

[8] https://www.wikidata.org/wiki/Wikidata:Glossary

[9] https://en.wikipedia.org/wiki/Wikipedia:Purpose

[10] https://en.wikipedia.org/wiki/Wikipedia:CAREFUL

[11] https://en.wikipedia.org/wiki/Wikipedia:Consistency

[12] http://www.bbc.com/news/world-africa-14069082

[13] https://commons.wikimedia.org/wiki/Category:Maps_of_the_world

[14] http://www.ruthstalkerfirth.com/pdf/accessibility.pdf

[15] http://www.color-blindness.com/coblis-color-blindness-simulator/

[16] https://meta.wikimedia.org/wiki/Research:Wikipedia_Readership_Survey_2011/Results#f._21.25_of_US_Wikipedia_readers_have_read_Wikipedia_on_a_tablet

[17] https://en.wikipedia.org/wiki/Wikipedia:VisualEditor/Why

[18] https://www.mediawiki.org/wiki/Developers

[19] https://en.wikipedia.org/wiki/Array_programming

[20] http://www.cs.utah.edu/~germain/PPS/Topics/Matlab/vectorized_or_array_operations.html

[21] https://www.mediawiki.org/wiki/Manual:Code

[22] https://test.wikipedia.org/wiki/Main_Page

[23] https://www.mediawiki.org/wiki/Manual:Extensions

[24] http://www.w3schools.com/svg/svg_inhtml.asp

[25] https://en.wikipedia.org/wiki/HTML5

[26] https://wikimediafoundation.org/wiki/Press_releases/One_Laptop_Per_Child_Includes_Wikipedia_on_$100_Laptops

[27] https://www.mediawiki.org/wiki/Extension:GraphViz

[28] https://www.mediawiki.org/wiki/Extension:Graph

[29] https://www.mediawiki.org/wiki/Manual:Pywikibot

[30] https://docs.python.org/2/reference/datamodel.html

[31] http://stackoverflow.com/questions/30362600/how-to-install-requests-module-in-python-3-4-instead-of-2-7

[32] http://osxdaily.com/2012/10/02/stress-test-mac-cpu/

[33] https://www.wikidata.org/wiki/Wikidata:Project_chat/Archive/2013/12

[34] https://meta.wikimedia.org/wiki/Wikidata/Notes/Caching_investigation

[35] http://stackoverflow.com/questions/2846653/python-multithreading-for-dummies

[36] http://effbot.org/zone/thread-synchronization.htm

[37] http://www.tutorialspoint.com/python/python_database_access.htm

[38] http://effbot.org/tkinterbook/tkinter-index.htm

[39] https://github.com/WojciechMula/canvas2svg

[40] http://stackoverflow.com/questions/8797965/python-refresh-html-document

[41] http://stackoverflow.com/questions/2787679/how-to-reload-page-every-5-second

[42] http://www.sas.com/en_us/insights/big-data/what-is-big-data.html

Technology Scholarship 2015
Outstanding

Commentary

The candidate clearly communicated a wide range of technical practice undertaken at Level 8 of the curriculum. Evidence within the report was in-depth and covered the three strands, for example:

- The students report has extensive explanations and discussions that anchor the practice from the beginning – this is essential for all scholarship candidates
- These discussions reveal the in depth and a critical initial exploration of the context and the issues faced by wiki map designers.
- Awareness of the social environment was demonstrated in terms of strong discussion about the needs of user, clients and the organisation and future trends within the proposed market.
- A high level of critical thinking and reflection was shown around the needs of the target market.
- The chosen issue was demanding and required a great deal of technical strategy, research and investigation and ingenuity when problem solving the challenges faced in such an authentic issue.
- Analysis of potential outcome directions communicates the reflective nature of the student's technological practice and gives clarity to the technological approach undertaken.
- Technological modelling was ongoing throughout the development of the outcome and the student had a comprehensive understanding of the competing and demanding contestable factors involved in creating the outcome.
- This demonstrated optimisation and polish of the student's technological practice along with the continual testing and modification.
- Feedback from others and critical reflections on the products use within its community was regularly used to inform practice.
- The report demonstrates elegant synthesis and integration by bringing together all of the information required to produce an outcome of very high quality.
- The brief was not met in its entirety but conceptually the outcome met the needs of the community of users so was fit for purpose in the broadest possible sense.