An Application Framework Supporting Big Data for the British Library via Windows Azure Search Service

Nektaria Stavrou
Department of Computer Science
University College London
ucabnst@ucl.ac.uk

Stelios Georgiou
Department of Computer Science
University College London
ucabsge@ucl.ac.uk

Abstract

Nowadays, the rapid expansion and advancement of technologies in the field of Information and Communication Technology (ICT) has led to an increase in the number of data sources queried across the Internet. One noticeable drawback, however, is the heterogeneity among the data collections which is not facilitated by a standard programming interface. Additionally, infrastructures that provide data integration among heterogeneous collections do not provide a well unified structure of the output. To solve these issues the British Library Big Data Experiment was conducted, incorporating a set of Microsoft Azure Cloud technologies to query different collections of heterogeneous data. The most important component of the project is Azure Search Service API, which enables the creation of search services. In particular, the project used data from the digitised collection of the British Library. Our contributions are as follows: (1) Developing an open framework that supports data integration among heterogeneous collections and (2) Defining an infrastructure that allows the user to dynamically create his own queries and (3) and produce results in a unified structure.

Categories and Subject Descriptors
D.2.1 [**Requirements/Specifications**]: Elicitation methods, Methodologies.
H.2.4 [**Systems]**: Query processing.

General Terms Measurement, Documentation, Performance, Design, Standardization, Languages, Verification.

Keywords Microsoft Azure Search Service, cloud, metadata, British Library.

1. Introduction

The rapid development of technologies in the field of Information and Communications Technology (ICT) has resulted in the creation of large data sets carrying high levels of complexity and a wealth of information. In addition, the emerging capabilities offered by a typical Software System have led to a subsequent increase in user requirements and needs. All the above combined indicate that computing will eventually become one of the fundamental utilities (after water, electricity, gas and telephony) that will satisfy people's daily needs [1]. To deliver this vision, numerous computing paradigms and software architectures have been proposed. In recent years, the predominant proposed paradigm is that of Cloud Computing.

As defined by Armbrust et al. Cloud Computing refers to "both the applications delivered as services over the Internet, and the hardware and system software in the datacentres that provide those services" [2]. These services are named Software as a Service (SaaS), whereas the hardware and software within the datacentres form the concept of a Cloud [2]. The aforementioned on-demand delivery of resources and applications via the Internet operates on a pay-as-you-go basis [3].

A diverse variety of organisations – depending upon their size, scale, sector and strategic aims – implement Cloud Computing due to the numerous advantages that it may provide. This includes lower capital expenditure, ease of maintenance, enhanced flexibility and mobility of Cloud services, increased continuity of business, and improved security [4]. These key advantages have incited the interest of many organisations to shift towards Cloud Computing. In particular, not only profit-driven organisations but also research-based institutes such as libraries, museums and hospitals, have made the decision to move their contents into the Cloud.

A notable example of a research-based organisation that utilised Cloud Computing is the digitisation strategy followed by the British Library, the national library of the United Kingdom. Globally renowned for its rich and comprehensive collection, the British Library provides researchers and the public with extensive information that has accumulated for over 250 years and that is beyond 150 million items [5].

1.1 Problem Formulation

For the past two decades [6] the British Library has undertaken activities, as part of their digital library programme strategy for 2020, to convert diverse information such as text, images and manuscripts into a digitised format. The aim is to establish a digital infrastructure that would enable current and future generations of researchers or Cloud users to enquiry and enhance their search capabilities.

Aside the apparent benefits of the digitisation plan and particularly Cloud Computing, there are several challenges that we need to be aware of in order to effectively utilise such infrastructures.

Firstly, the aggregation of data coming from different sources raises the issue of heterogeneity. That is, the integration of different types of data (e.g. integers, strings) is not facilitated by a standard programming interface. Secondly, as a consequence of

SIGPLAN'05 June 12–15, 2005, Location, State, Country.

the diversity of data available in a collection, there is a resulting difficulty in querying the data. More specifically, the structure of a query is currently predefined according to a data schema therefore restricting an application's adaptability to different types of data.

With the digitisation strategy of the British Library, the project aims to tackle these challenges and contribute to the Software Engineering field. This aim will be realised by providing increased accessibility, visibility and discoverability to the digitised collections of the British Library, and through the analysis of large quantities of digital collections using open APIs.

2. Related Work

Previews studies have used alternative technologies, besides Cloud technology, to accomplish aims and goals that are similar to the British Library Big Data Experiment. This section will introduce key related work in order to provide the foundation and motivation of the British Library Big Data Experiment.

Malmsten [7] in 2008 focused on incorporating a library catalogue, the Swedish Union Catalogue (LIBRIS), into the Semantic Web [8] and Linked Data [9]. The LIBRIS includes material from 175 libraries using a single Integrated Library System (ILS). More specifically, Malmsten's aim was to create a web interface that would integrate the definition of Semantic Web and Linked Data; that is, to reveal links among library material and make them available to public enquiry.

Malmsten's implementation exploits the Resource Description Framework (RDF) used for modelling and describing data in the web. An RDF server wrapper was initially created to access ILS through HTTP requests, and to deliver library records into RDF, rather than the binary format normally used to describe bibliographic records. The wrapper is connected to the ILS through SQL and can deliver records only by using their unique identifier. This is in contrast to the Azure Search Service API used in the British Library Big Data Experiment, which can query any field of a data schema. Finally, the data was transformed into XML format using XSLT [10].

Furthermore, Malmsten's solution connects library resources using URIs created for each record and makes them accessible through HTTP URI. HTTP content negotiation, utilises HTTP Accept Header in order to deliver the appropriate format to the user. For example, if the accept header is text/html then the server will deliver an html page to the user. Subsequently, this newly created data was loaded into a triple store to enable searching using SPARQL, a framework for both advanced queries and data analysis. Finally, the records in the LIBRIS database are assigned transient keys that are a normalised concatenation of an author and the title. This means that if an author's name is changed the link will remain the same.

Other work related to the current British Library Big Data Experiment is the study by Baldonado et al. [11] at Stanford University. Baldonado et al.'s overall aim was to develop an architecture that enables interoperability among disparate autonomous digital library services. In particular, when users search for specific information among heterogeneous collections the developed infrastructure pulls the most relevant collection according to the user's input, and subsequently displays a unified set of heterogeneous results in an integrated overview.

To accomplish the above, Baldonado et al. [11] had to identify the most relevant collections given the users' input by using an automated resource-discovery service. This service (GIOSS [12, 13]) would rely upon metadata of the collections within the digital library architecture. The purpose of GIOSS was to take a user's query and return a rank list of the collections, based on the number of hits the query is found. Prior to resource-discovery, in order to capture user input the user would have to express their search as a query through a Boolean frond-end language (DLITE) [14, 15, 16]. An automated query translator service would subsequently reform the query from the frond-end into an equivalent for each collection. Lastly, the infrastructure provides the user with a set of unified results.

Moreover, the most important component of the Baldonado et al. project was the InfoBus Metadata Facility. The objective of the InfoBus was to collect metadata about the collections within the system. This facility consists of several distributed objects communicating with each other using remote method calls. If a service is located externally, the communication is achieved using service proxies. In essence, the facility exports metadata for each remote and local service, and eventually stores it in a local database (metadata repository) that facilitates the indexing of the collections.

Relevant to our experiment was also the Chabot project at UC Berkley [17], which retrieved digitized images from a larger collection taken from the Department of Water Resources (DWR) in California. One of the goals was to build a system that will support vast types of complex data and a functionality similar to an advanced relational Database Management System (DBMS). A key component of this functionality includes a query language which uses effective indexing to optimise queries and return back results quickly. A vital and challenging goal of the project was to retrieve images not only by examining the related information among them, but also by using image analysis techniques to retrieve additional information to return the correct results in a search query.

For its implementation, the Chabot project utilised POSTGRES [18,19], an object-relational database management system (ORDBMS) of storing and managing images with their corresponding textual data. In contrast to a typical DBMS, POSTGRES provides the database developer with the ability to handle more complex data types as well us to create new custom data types. These are achieved by defining classes. In addition, other object oriented characteristics were provided such as class inheritance.

A functionality of POSTGRES that was applied in the project is the storage of pre-computed information related to each image, which was displayed as a histogram containing the image's colours. A second functionality was defining functions that can be executed during the runtime to analyse the colour histograms. Moreover, a mechanism called "Meets Criteria" was developed to identify whether an image of the database meets a specific colour criterion. This functionality obtains two inputs: an image's colour histogram and the colour criterion that an image should meet. For example, the colour criterion might include the terms "Mostly black" or "Some Red". "MeetsCriteria" is called for every existing histogram in the database by the POSTGRES query executor. In the case that a histogram meets the criterion, then the image that is associated with that histogram appears in the results.

2.1 Alternative Technologies

This sub-section examines an alternative technology for the current problem besides Azure Search Service, the Microsoft Azure Storage. Despite the remarkable benefits of Azure Storage, as it will be discussed below, the authors believe that the proposed solution is more suitable towards the realisation of the project's goals.

Azure Storage consists of three services: Blob storage, Table storage and Queue storage. Azure Blob storage stores any type of data such as text or binary, while Table storage stores only structured datasets. Azure Queue storage is a service for storing large

numbers of messages that can be accessed from anywhere in the world via authenticated calls using HTTP or HTTPS. A single queue message can be up to 64 KB in size, and a queue can contain millions of messages up to the total capacity limit of a storage account. A storage account can contain up to 200 TB of blob, queue, and table data [20].

Access to Azure Storage is available from any type of device including a desktop computer, mobile, tablet, an on-premises server or even from the Cloud [20]. Azure Storage supports users who might be using various operating systems such as Windows or Linux and also supports multiple programming languages including .NET, Java and C++ [20]. Azure Storage is therefore flexible in that applications can be designed to address a large number of users and can be scaled accordingly in order to handle different amounts of data or a number of requests [20].

A solution to the British Library Big Data Experiment is the implementation of Azure Table Storage. This alternative was thoroughly investigated during the initial phases of the project (Experiment 1). The Azure Table storage service is a NoSQL key-value datastore that stores and manages huge amounts of non-relational digital structured information, and it is schema free [21][22]. In order to access Azure Storage, a Cloud storage account is required that scales accordingly to the region that the account was registered to [23]. Each storage account has a limited capacity restricting the number of tables that an account can contain [21].

Azure Tables contain a set of entities which represent the data. Each entity is associated by default with several system properties including the PartitionKey, RowKey and Timestamp properties. The names of these system properties cannot be modified by the user. The PartitionKey and RowKey uniquely identify each entity in the Azure Table storage, whereas the table entities are organised based on the PrimaryKey. The values of the PartitionKey and RowKey, that are strings having a size up to 1 KB, can be managed by the developer, however, the Timestamp property has a DateTime value that is only managed by the server and represents the time that each record was last updated [24].

Tables that are associated with a specific storage account can be accessed from code by using the URL http://<storage account>.table.core.windows.net/<table>. The specific URL contains both the storage account and the table name. Open Data Protocol (ODB), that is used to create and consume data APIs, is used to address the required Azure Tables [21][25]. An alternative way to access the Azure Table storage is by using the Microsoft.WindowsAzure.Storage.dll dependency.

The Windows Azure Storage can be accessed by including the following namespace declarations: Microsoft.WindowsAzure.Storage; Microsoft.WindowsAzure.Storage.Auth and Microsoft.WindowsAzure.Storage.Table. For the retrieval of the entities in a table partition, a CloudTableClient is declared and subsequently used in order to access a specific table. To search throughout the table, a query has to be executed using as an identifier the PartitionKey of an entry. However, for the retrieval of a range of entities the combination of PartitionKey and RowKey is required.

2.2 Motivation

The aforementioned research projects and technologies, can be considered as alternative solutions to the current problem being investigated by the current British Library Big Data Experiment. That is, to provide an application framework that enables the user to query heterogeneous data.

Malmsten's approach to developing a mechanism, incorporating library catalogue data into the semantic web using transient keys, has the potential to misguide the user because though the link remains the same the data enclosed within the link (record) may have changed (e.g. author name). In contrast, Azure search service can query any field of a data schema without any limitations.

Furthermore, Baldonado et al.'s [11] infrastructure delivers users with information from diverse heterogeneous resources. Their proposed framework consists of several sub-services – both internal and external – that make the implemented solution unnecessarily complex. On the contrary, Azure Search Service API provides the capability to query heterogeneous datasets with only one prerequisite, an unstructured file (JSON) that can be parsed to determine the appropriate data schema. Although Baldonado et al. manage to deliver results in an integrated overview, their infrastructure has a potential limitation. That is, the exact same attributes are used to describe all returned results even though they might come from different collections. This might misguide or confuse users because their infrastructure is not fully adaptable to the metadata the search results are derived from.

Also, Chabot's retrieval system is a challenging combination of ORDBMS and content analysis techniques both used successfully to improve the ability to retrieve images efficiently. Evidently, the combination of the two techniques provides optimal results. Nevertheless, the Chabot's retrieval system does not consider any ranking mechanism of the returned results that would firstly provide the user with the ability to obtain optimal matching results immediately, and secondly minimize the return set if needed. The Azure Search Service does however provide this feature.

Finally, the Azure Table Storage does not support the dynamic creation of queries. This is due to the fact that the structure of the table is predefined. Moreover, in order to search through the fields of an entry the combination of PartitionKey and RowKey is required, whereas with Azure Search Service fields this can be queried directly.

Given these potential limitations, the project aims to exploit the advantages of Azure Search Service under an Application Framework in order to: (1) find a solution that supports data integration among heterogeneous collections, and (2) develop an infrastructure that eventually allows the user to dynamically create his own queries (3) and produce results in a unified structure.

3. Requirements Engineering and Analysis

This section will thoroughly investigate all the Requirement Engineering aspects of the project.

3.1 Purpose

The purpose of the British Library (BL) Web Portal is to provide an improved method for accessing and analysing digitised material through an Application Framework, enabling the next generations of researchers and public enquiries to advance their search capabilities. The portal primarily provides a means to access and query the digitised collection in the Microsoft Azure Cloud. In particular, users can construct their own search service through the use of Azure Search Service API, and by exploiting the benefits of the Cloud infrastructure, they are also able to download and analyse material. These enhanced user capabilities are delivered via an easy to use, learn and recall research oriented front-end.

3.2 Scope

In Software Engineering literature, a System Context Diagram (SCD) is a high-level diagram written in natural language that

bounds the problem world by declaring its interactions with the environmental entities it interacts with [26]. The SCD demonstrates that there is a clear separation between the system and real world phenomena. Figure 1 illustrates the SCD of the British Library Web Portal.

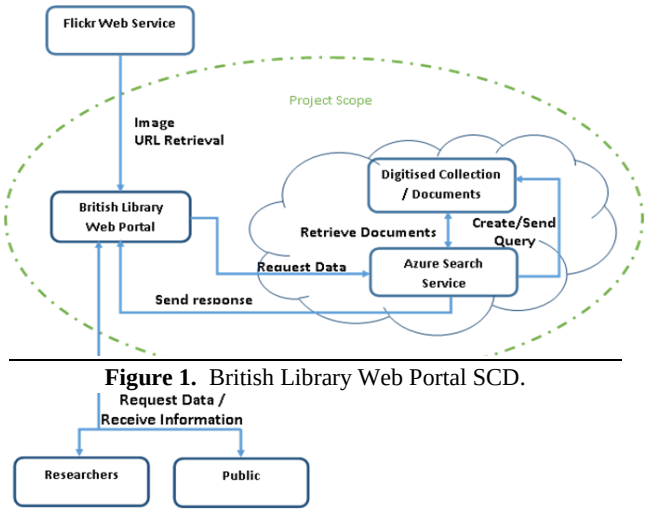


Figure 1. British Library Web Portal SCD.

The digitised collection resides in Windows Azure Table Storage. The first step towards the solution is to mine into this data and extract as much information as possible. In order to achieve this queries will be executed using Azure Search Service. In addition, the web portal will be directly connected to the Flickr Web Service, which stores all images of the digitised collection. Following that process the users will be able to download and analyse the digitised collection using a set of analysis tools. With this achieved, researchers will be able to re-organise the new pieces of information and make them available for future use not only by themselves and also by others.

Despite the fact that requirements often change, the project scope remains the same. A refinement in the requirements is usually concerned with technical solutions that facilitate the development of the product rather than the project scope.

3.3 Requirements Engineering

The Requirements Engineering (RE) phase occurs at the beginning of every software development cycle and provides the foundation for successful deliverables. As observed by Bell and Thayer inadequate, inconsistent, incomplete or ambiguous requirements that are often encountered may have a critical impact on the final product's quality [27]. The primary measure of the quality and success of the final product is the degree to which it meets the purpose for which it was intended [28].

Given the above, requirements engineering is concerned with identifying stakeholders and their needs regarding the intended system; specifying services and constraints; and finally assigning responsibilities for the resulting requirements to agents such as humans, devices and software [29]. Without the British Library project being an exception, the initial task of the development team was to therefore extract, model, analyse and reach a consensus on the requirements among the several stakeholders of the project.

3.3.1 Stakeholders

The first step in requirements engineering was to identify all the stakeholders involved in the project. According to Rozanski and Woods a stakeholder of a system is any individual, group or entity with an interest or a concern about the architecture's realisation [30]. For example users, developers, operators and acquires are all considered stakeholders [30].

A particular challenge with multiple stakeholders is the conflicting goals each might have about the intended system. Identifying stakeholders early in the process is therefore beneficial not only for the requirements elicitation phase but also for the overall project cycle. This is because early identification can help increase the social capital among different stakeholders consequently minimising possibilities for conflict.

For the purposes of the project, the development team used the Onion model for stakeholder identification. Though there are several methods to visualise stakeholder culture and hierarchy, the Onion model was used because it facilitates the conceptualisation of stakeholders in four key layers. The project's final product, the British Library Web Portal, is illustrated at the centre of the model (Figure 2). The System layer includes all the individuals who will operate and maintain the product. The Containing System layer captures people outside the project's control but who will benefit from the new system. Finally, the Wider Environment includes the Containing System and any other stakeholders who might affect decisions about the project.

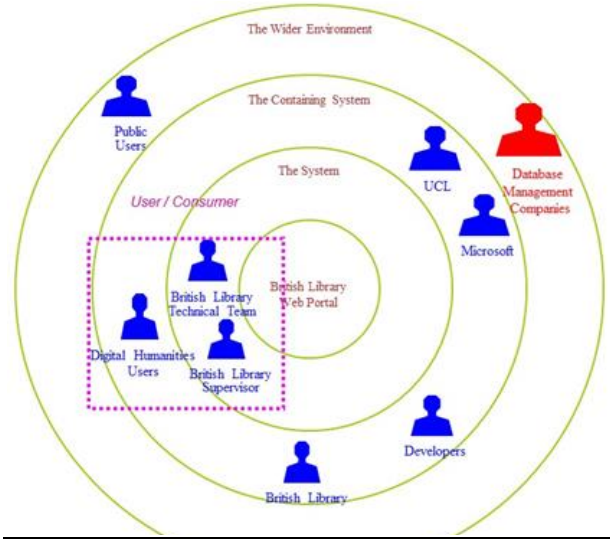


Figure 2. Onion model of the stakeholders.

Following stakeholder identification, the development team prioritised each stakeholder's importance and impact on the final deliverable by using a stakeholder investment graph (Figure 3). This aided in the prioritisation of the project's requirements.

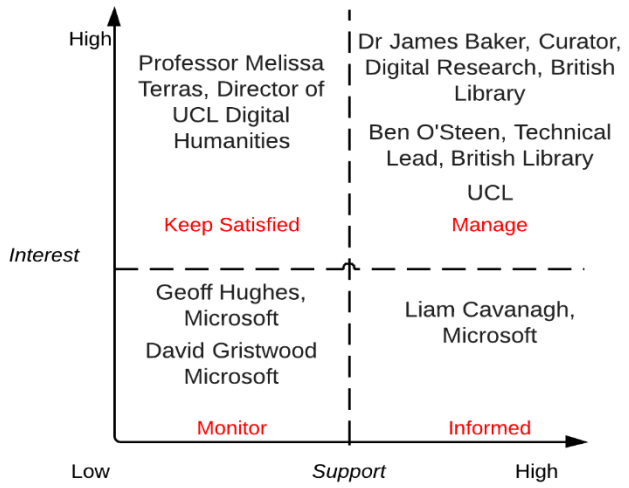


Figure 3. Investment graph of the stakeholders.

3.4 Project Methodology

A software development methodology is used to structure, plan and control the process of developing a new system. Though Sommerville acknowledges that there is no ideal software process [31] agile methodologies which are part of the Iterative and Incremental development (IID) framework, are continuously adapting in order to adequately cope with emerging market needs. As defined by Larman [32], the overall project cycle in IID methodology is divided into numerous iterations where each is considered as a self-contained project composed of a set of activities. This includes requirements elicitation and analysis, design, implementation, integration and testing. An iteration must be a fully functional component before it can be integrated and tested with the existing system which grows incrementally.

It is important to note that the British Library has primarily used the waterfall model of software development. Although it may be easy to define and is well documented at each stage, it carries several risks. For example, in order to proceed to each subsequent phase the phase must be fully completed. This is a risk prone approach to a project as it makes it highly likely that at any point during the software cycle there will be a change in the requirements.

In contrast to the waterfall model, agile methodology is more flexible and adaptive to client requirements. As previously mentioned, smaller incremental chunks are used to build a new system. When errors are observed during iterations, the development process will not continue until there is a fully functional outcome. This is considered as one of the major benefits of agile methodology because, even in the worst case scenario, a minimum viable product is secured.

Therefore, for the purposes of this project agile methodology was applied; all functionalities were split into smaller functions which were sequentially implemented, tested and reviewed by the client. Also, following the principles of agile manifesto [33] the developing team had frequent progress meetings with the client with the purpose of reviewing and evaluating the functionalities that were developed in every iteration. This continuous communication and close collaboration with the client was beneficial because it allowed the identification and verification of client needs; validation of the current functionalities; and the resolution of possible issues in product design. At the end of each iteration, the development team and the client discussed any refinements in requirements according to their priority and the available time remaining for project closure. This was facilitated by agile methodology because it allowed the developing team be highly responsive and adaptive to changes throughout the process.

3.4.1 Project Plan

Planning management was an important aspect of the project as it required a thorough understanding of the stakeholders' needs and defining the tasks to achieve these. Project scheduling was also crucial as the team needed to achieve the project objectives in time.

3.5 Requirements Elicitation

Requirements elicitation is a set of activities concerned with understanding a problem and its context; identifying and extracting the real needs of the user; and detecting any problems that might be encountered when implementing a new system. During requirements elicitation, software engineers work closely with the client and the end users of the system in order to gain as much information as possible. This is a complicated process carrying several risks, primarily because clients do not have a comprehensive view about the specifications of their new system. In particular, clients may have hidden needs that they fail to mention because they assume that others are aware of them, and needs they might not realise are possible or that are required for the new system.

In order to extract the current project's requirements it was decided that the team should conduct client and expert domain interviews, user focus groups, and lastly to compile a literature review of similar projects to gain valuable background knowledge.

3.5.1 Client and Expert Domain Interviews

Semi-structured interviews were used combining closed-ended questions at the beginning followed by open-ended questions towards the end to ensure a variety of response. This method was preferred because it is flexible and made it possible to raise hidden issues and concerns by tapping into the interviewees' tacit knowledge. The project also had a clear customer-driven approach in that client interviews were carried out multiple times throughout the software development cycle.

3.5.2 User focus groups

To gain a wider understanding of the problem, and to fill any gaps in knowledge that could not be extracted from the client and expert domain interviews, the development team invited 8 members of the Digital Humanities Community who are frequent users of similar systems. The focus groups were divided into two teams of four, where each team had two members of the development team as facilitators. The method of interviewing was identical to the one used in the client and expert domain interviews, including both close-ended and open-ended questions. In addition to the semi-structured interviews, users were asked to complete a short paper exercise which required users to brainstorm and write down

the steps they typically follow when conducting an online search for a hypothetical author (see Appendix A). The purpose of this exercise was to extract users' tacit knowledge and also to identify any conflicting requirements among potential users of the new system.

3.5.3 Live Document

Following the client and expert domain interviews and focus groups, the development team created a live document of the requirements which was distributed to all the project's stakeholders. The aim was to obtain their views and to reach a final agreement of the requirements.

3.5.4 Use Case Scenarios

To gain a better understanding of the client's functional requirements, a set of Use Case Scenarios were produced. That is, the functionality of the target system was described using scenarios in a non-technical language. This activity acted as a communication link between the clients and the development team during the requirements elicitation stage. A full list of the project's UML diagrams and use case scenarios can be found in Appendix B of the report.

3.5.5 Goal Models

Lastly, another technique to facilitate the requirements engineering process was the development of goal models. Goals play a prominent role in requirements engineering [34]. The parent goal of the project was to develop a web portal that would provide users with access to the digitised collection. In addition, the parent goal was refined into several sub-goals. The sub-goals helped us to identify multiple involving agents that take part in the realisation of the top goal. The goal models of the project can be found in Appendix C.

3.5.6 Requirements Prioritisation

The extracted requirements were prioritised using the MoSCow technique to gain a better understanding of what the client considers essential for a minimum viable product. In addition, due to the limited time constraints of the project, this technique allowed effective time management by allowing the team to focus on the most important requirements. In particular, the MoSCow technique categorises requirements in the following groups: Must (M), Should (S), Could (C) and Won't (W) have. M requirements are compulsory and satisfy the minimum requirements of the project; S requirements should be included if possible, but project success does not depend on these; C requirements are optional and not necessary to complete; and lastly, W requirements will not be incorporated in the final solution. The abstract requirements are shown in Table 1, whereas the product specific requirements can be found in the Appendix D of the paper.

3.6 Project Plan

Following the requirements elicitation and agreement stage with the project stakeholders, the next immediate step was to devise a Project Plan. Since the project methodology followed was agile, the project cycle had several iterations. During the first iterations it was decided that the Must Requirements must be implemented. That was in order to ensure that a minimum viable product will be delivered to the user. The Gantt chart of the project plan can we found in Appendix E.

Table 1. British Library Big Data Experiment Abstract Requirements.

ID	Description	Priority (MoSCow)
RQ1	The system must provide an advanced search feature set.	M
RQ2	The user must be able to track, download and package the steps followed in order to produce a search output.	M
RQ3	The system must provide an intuitive user interface.	M
RQ4	The user should be able to analyse data using a statistical analysis feature set	S
RQ5	The system should enable users to reproduce queries using URIs	S
RQ6	The system could enable the users to package and download different types of data	C

3.7 Risk Management

Throughout the project it was highly probable that some risks will be encountered. In order to minimise the likelihood of unexpected events, at the beginning of the project risk analysis and evaluation was performed. During this process mitigation strategies and contingency plans were developed. The full Risk Analysis document can be found in Appendix F of the report.

4. Approach

This section thoroughly investigates the methods and techniques used to develop the proposed solution. As it was already mentioned, the major component in the solution is the Azure Search Service API which described in detail in the following sections.

4.1 Product Description

The final deliverable of the project is an application framework that supports the query of heterogeneous data. More specifically, an intuitive frond-end was developed enabling the user to query the digitised collection of the British Library. The users are now able to construct their own queries using Boolean Operators and subsequently browse the collection. Moreover, the users are able to download and analyse the OCR text of a document. Finally the system allows the users to download the URI of their search results which acts as redirection path (trace).

4.2 Azure Search Service

Azure Search Service is a fully managed, cloud-based service that allows developers to build rich search applications using REST APIs. [35]. It provides two types of actions, Index Management and Query actions. By using these two actions the researcher is able to discover relevant information from the British Library's digital content. Index Management actions refer to administrative tasks associated with the search service or search index including creating, updating or deleting search indexes as well as collecting index information. Moreover, such tasks provide the search index with documents and can also verify their status. In addition, Query actions are queries associated with searching the content of a

search index; this includes search, document look-up, document count and suggestions.

The creation and maintenance of a search service, and its indexes, involves a particular usage flow that an administrator should follow. The initial key steps are creating the Search Service, after which two administrative keys and a single query key are obtained, and subsequently creating the Search Index. It is important to note that the search index is enclosed within the search service, therefore, in order to create it, a data schema must be defined in JSON format. Moreover, each service can have up to 5 indexes and up of 1 GB of storage for both the content of documents and the generation of the indexes.

The subsequent steps are to add documents to the newly created index and wait for them to be available for search. Inserting documents into the search index requires loading a JSON file that contains the data. Each JSON file includes up to 1000 unique documents that are distinguished by a unique key, and the data in every document are expressed as field-value pairs. For each service index, there is a limitation of up to 1.000.000 documents and documents may be loaded in batches of a 100. Additionally, inserting documents into the service may cause a delay as they are placed into a queue for processing and indexing, and queries can only be performed once documents become searchable.

Once the search service and indexes are created and the documents have become searchable, the administrator will be able to perform changes to the schema and search index, delete or update a document from an index, and even delete indexes.

4.3 Query construction

When constructing a query, it can be customised using the following request parameters. Some important request parameters are search=[string], searchMode=any|all, searchFields=[string], $skip=#, $top=#, $count=true|false, $orderby=[string], $select=[string], $fil-ter=[string].

The request parameter “search” contains the text that the search is made from and is applied only to searchable fields. The parameter searchMode=any|all is used to check if all or any of the terms that are included in the search keyword must be identified in order to consider a document as a match and be returned to the user. The $skip parameter is used to define the number of results that you want to skip whereas the $top is used for the number of results that you want to retrieve. The number of results that you are able to skip is not more than 100 000 whereas the number of results that you are able to retrieve are a maximum of 1000. The default value for the results that will be returned with each query is 50. The $count=true|false parameter defines whether or not you want to retrieve the total count of results that were returned in a search.

The $select attribute defines which fields of the schema are to be retrieved. By default all the fields are retrievable. The $filter attribute is used as a search expression and supports a subset of expressions of the Open Data Protocol (OData) syntax. Some expressions that are supported by the OData expression syntax are the logical operators; "and", "or", "not", and comparison expressions equal("eq"). not equal ("ne"), greater than("gt"), less than("lt"), greater equal ("qe") and less equal ("le").

4.4 The British Library Data Contents.

The British Library allows the public to access over a million images including maps, geological diagrams and decorative letters from books between the 17th to 19th centuries, digitised by Microsoft. These are available to the public through Flickr Commons.

For the purpose of the project, the British Library, provided us with the digitised collection in an unstructured file (JSON). This file contained 36 000 entries. However, this could not be used by the Azure Search Service due to the fact that it contained nested fields. Each entry in the JSON file represented an image taken from a book. More specifically, an example of an entry had the following structure:


```
{"idx": "1",
"flickr_original_jpeg": "sample url",
"printsysnum": "003308204",
"vol": "0",
"title": "sampel title",
"ocrtext": "sample url",
"scannumber": "000174",
"height": "370",
"authors": {"creator": ["SCHWERDT, Heinrich."]},
"width": "873",
"fromshelfmark": "British Library HMNTS 10411.bb.20.",
"biblioasjson":
        {"datefield": "1858",
        "shelfmarks": ["British Library 10411.bb.20."],
        "publisher": "",
        "title": ["sample title"],
        "edition": "",
        "place": "Leipzig",
        "issuance": "monographic",
        "authors": {"creator": [ Heinrich."]},
        "date": "1858",
        "identifier": "003308204", "corporate": {}},
"place": "Leipzig",
"sizebracket": "embellishments",
"electronicsysnum": "014841228",
"date": "1858",
"flickr_url": "sample url",
"azure_url": "sample url",
"pdfs": [["http://access.dl.bl.uk/lsidyv35b2caac", ""]],
"tags": []}
```


As it can be observed, the bibliojson field, is nested. In order to solve this problem all the data had to be restructured in a flat schema. Also, during this process, duplicated fields in the entries were removed.

4.5 Web Portal Data Schema

After flattening the data, the next step towards the solution was to define our data schema. The data schema for our Azure Search Service was structured in JSON format as follows:


```
{
 "fields": [
{"name":"azure_url", "type": "Edm.String", "searchable": false},
{ "name": "creators_and_contributors", "type": "Collection(Edm.String)"},
{ "name": "date", "type": "Edm.Int32"},
{ "name": "electronicsysnum", "type": "Edm.Int32"},
{ "name": "flickr_original_jpeg", "type": "Edm.String","searchable" : false},
{ "name": "flickr_url", "type": "Edm.String","searchable" : false},
{ "name": "fromshelfmark", "type": "Edm.String", "searchable" : false},
{ "name": "height", "type": "Edm.Int32"},
{ "name": "idx", "type": "Edm.Int32"},
```

```
    { "name": "ocrtext", "type": "Edm.String"},
    { "name": "place", "type": "Edm.String"},
    { "name": "printsysnum", "type": "Edm.Int32"},
    { "name": "publisher", "type": "Edm.String"},
    { "name": "scannumber", "type": "Edm.Int32"},
    { "name": "sizebracket", "type": "Edm.String", "searchable" :
false},
    { "name": "tags", "type": "Collection(Edm.String)"},
    { "name": "title", "type": "Edm.String", "suggestions": true },
    { "name": "vol", "type": "Edm.Int32"},
    { "name": "width", "type": "Edm.Int32"},
    { "name": "key", "type": "Edm.String", "searchable" : false}
     ]
}
```

As it can be observed, some of the fields in the data schema have searchable attribute set to false. The purpose of that is to save space in the index.

The fields of a JSON file are expressed based on the Entity Data Model. The data types that are supported are: Edm.String, Collection(Edm.String), Edm.Boolean, Edm.Int32, Edm.Double, Edm.DateTime, Edm.GeographyPoint, Edm.GeographyPolygon. These correspond to strings, list of strings, true/false values, numbers, double-precision numeric data, a pair of latitude/longitude values, and set of latitude and longitude values, respectively.

In the data schema, apart from the name and field type, other important attributes can be defined as well which facilitate the retrieval of appropriate items. These include searchable, filterable, sortable and retrievable attributes.

The searchable attribute sets the field to be full-text searchable which subsequently enables word-breaking in the index. The searchable attribute can only be applied to fields with Edm.string and Collection(Edm.String). In the case that a field is not required by the query, the searchable attribute can be set as false. This saves space in the index as Azure Search stores an extra tokenized version for each field with searchable attribute.

Moreover, the filterable attribute enables the developer to reference a field as $filter in the queries. In contrast with the searchable attribute, the filterable attribute for Edm.String, Collection(Edm.String) does not provide word-breaking but only exact matching instead. All fields regardless of their type are filterable by default. In addition, the sortable attribute is activated by default and can be used in all fields except for fields with Collection(Edm.String) to sort the results based on a field. Lastly, the retrievable attribute defines whether a field will be included in the results of a search.

4.6 BookModel data model

After defining the data schema of our search service, the next step was to create a Model that will matche this schema. This model, BookModel, can be considered as the most important model since it makes the connection between the data schema and the software architecture. The BookModel is defined as follows:

```
public class BookModel
   {

      public int key { get; set; }
      public Double score { get; set; }
      public string azure_url { get; set; }
      public List<string> creators_and_contributors { get; set; }
      public Int32 date { get; set; }
      public Int32 electronicsysnum { get; set; }
      public string flickr_original_jpeg { get; set; }
      public string flickr_url { get; set; }
      public string fromshelfmark { get; set; }
      public Int32 height { get; set; }
      public Int32 idx { get; set; }
      public string ocrtext { get; set; }
      public string place { get; set; }
      public Int32 printsysnum { get; set; }
      public string publisher { get; set; }
      public Int32 scannumber { get; set; }
      public string sizebracket { get; set; }
      public List<string> tags { get; set; }
      public string title { get; set; }
      public Int32 vol { get; set; }
      public Int32 width { get; set; }

      public override int GetHashCode()
      {
         return key;
      }

      public override bool Equals(Object obj)
      {
         if (obj == null || !(obj is BookModel))
            return false;
         else
            return key == ((BookModel)obj).key;
      }
   }
```

As part of the BookModel class, we also defined GetHash-Code() and Equals() functions in order to check whether or not two objects BookModel objects are equal. Two BookModels can be the same if they have the same data content, including the unique "key" property. The "key" is what uniquely identifies each Book-Model and therefore it used to compare objects at these functions.

4.7 Capturing User input

Two main Views, the Home and the AdvancedSearch page, were used. At the Home page the user can perform a simple keyword search to search across all the fields apart from the OCR, whereas at the AdvancedSearch page a user can perform more complicated queries using the boolean operators that can be applied on the fields. The corresponding controllers track the input of the user and then send the users choices to the AzureSearchService controller in order to execute the appropriate queries.

If the user performs a search from the Home page then the URI parameter stores the word "home" as well as the input keyword of the user and automatically redirects to the Index method of the AzureSearchService in order to execute the necessary queries.

When a user performs a search from the AdvancedSearchPage we can track and identify the exact input of the user based on both the textbox search term content (title, creator/contributor, publisher, publisher location, date range and OCR search) and select boxes containing the preferred boolean operator chosen by the user (and, or, not).The default boolean operator is "and".

In essence, the AdvancedSearch and Home controllers pass the words title, author, publisher, location, fromDate, toDate, ocr, and Boolean based on the choices of the user. At every case we can easily identify the choices of the user and perform accordingly. For instance if the user enters a keyword in the text box corresponding to "places", the word "boolean" with the chosen operator is passed to the URI parameter, followed by the word "location"

with the selected search keyword of the user. This example is illustrated below:

http://blpublicdomain.azurewebsites.net/AzureSearchService?queryID=boolean%C2%ACand%20%C2%AClocation%C2%ACLondon%20%C2%AC

4.8 Azure Search Service Controller

After capturing the user input, the following step is to execute the queries. This can be achieved through the AzureSearchService controller. This controller includes all the necessary functions that are needed for the creation and maintenance of a search service and its indexes.

Thus far, the procedure regarding the creation of the Search Service and its index has been discussed. In regards to the Azure Search Service Controller, the function CreateIndex() is responsible for creating the index and loading the schema. A key element of the function is that it loads the JSON file containing the schema.

Furthermore, in order to upload the data to the index, the function PostDocuments(string fileName) is executed where documents are POSTed to the newly created Index. When the Search Service and its indexes is established queries are performed based on the user's input in order to retrieve the appropriate results.

The function that is responsible for executing queries and retrieving and storing their results into lists is the function initialiseListsFromAzureSearch which is located at the AzureSearchService controller.

The first action the function initialiseListsFromAzureSearch does is to identify the user's selection. This can be achieved by taking the URI parameter queryID and identifing the user keyword input as well as the boolean selections per each property search field.

Therefore if for example the word "title" is identified in the URI parameter we will execute the following query:

```
response = ExecuteRequest("search", ref bookListTitle, "text=" + keyword + "$textMode=any&searchFields=" + searchField + "&$skip=0&$top=" + maximumNumberOfJson);
```

Given the above example, keyword represents user input; searchfield is the field a query will be executed on; and maximumNumberOfJson has the value of 1000 as the Azure Search Service has the limitation of retrieving a maximum of 1000 results in each query.

The above query adapts accordingly in instances where the user selects the fields such as CreatorsAndContributors, publisher or location. The only parameter that changes at each query is the name of the field as well as the list in which the results are stored in. The results are stored in a list<BookModel>. We have seven lists, one for each field; bookListTitle, bookListCreatorsAndContributors, bookListFromDate, bookListToDate, bookListPublisher, bookListLocation and bookListOcr.

With regards to the fields fromDate and toDate a slightly different approach is followed in the query by using OData syntax which includes the comparison operators greater equal ("ge") and less equal ("le"). An example of an executed query where the user wants to retrieve all the results from a specific year:

```
response = ExecuteRequest("search", ref bookListFromDate, "text=*&$filter=" + searchField + " " + "ge" + " " + keyword + "&$skip=0&$top=" + maximumNumberOfJson)
```

Given the above, searchField represents the value date; keyword represents the selected year value and the maximumNumberOfJson is set as 1000.

In the case that the user wants to retrieve search results of a specific time period a combination of comparison and logical operators is executed in the query:

```
response = ExecuteRequest("search", ref bookListFromDate, "text=*&$filter=(" + searchField + " " + "ge" + " " + keywordFromDate + " and " + searchField + " le " + keywordToDate + ")" + "&$skip=0&$top=" + maximumNumberOfJson)
```

4.8.1 Boolean Operators

The Azure Search Service can provide boolean operators on different fields, however it cannot simultaneously provide boolean operators and word breaking on the fields. In order to tackle this problem, we used the function booleanFunctionality which is located into the AzureSearchService Controller in order to achieve both word breaking on each field as well as boolean operators between the fields.

From our current description thus far, we have managed to retrieve for each field all the items and stored them in a list based on user input. The following step is to add combine all the Lists based on the boolean selection of the user. If the Boolean operator is "and" we identify all the common items in every list that the user has selected by performing a LINQ command. If "or" is selected union set operation is performed among the lists, whereas if "not" is selected some items are excluded from the final results.

4.9 Statistical Analysis Controller

Another important controller is the StatisticalAnalysisController which includes the functions frequentWords, frequentWordInOCR, frequentOfBiagrams and frequentOfTriagrams.

The function frequentWords (Uri filename, out List<string> MostFrequentWords, out List<int> FrequencyOfMostFrequentWords) calculates the frequency of words in a text document.
The blob URI which stores the OCR of an item is passed as a parameter to search through the entire text in order to find the most frequent words and their frequency. In addition, for the statistical analysis some words are excluded from the search results.

The funciton frequentWordInOCR calculates the frequency of a specific word within a text document. The user inputs a specific word and the system returns the number of occurrences.

Lastly, both the frequentOfBiagrams and frequentOfTriagrams functions calculates the frequency of two consecutive and three consecutive words in a text document, respectively. For example, the user inputs two words for a biagram or three words in a triagram and the frequency of that bigram or that triagram within the document is calculated and returned.

5. Design Patterns

Apart from the dynamic queries, the project aimed to develop an application framework that will support Big Data Analysis from different organisations. In this specific project, as it was already discussed, the solution is adapted to the digitised collection of the British Library. However, the design patterns used to develop the solution make the overall infrastructure an application framework that can be easily adapted and extended to different organisations.

5.1 MVC Design Pattern

The architectural design principle that is used for the British Library Big Data Experiment project is Model-View-Controller (MVC). This model provides a clear separation between its three major components; the model, the view and the controller. MVC helps developers in the sense that the individual parts of a system can be written, edited and tested easier. Models are responsible for the input logic aspect of the application whereas views are responsible for the user interface of the application, and finally controllers controls the input and the interaction of the user. The ASP.NET framework includes the ASP.NET MVC which permits developers to develop applications using the MVC design pattern [37].

5.2 RESTful Application Framework

The British Library Web portal uses the Representational State Transfer (REST) [38] architectural style that does not depend on any technology or platform and uses data formats like JSON [39]. The RESTful web services access the resources by using HTTP GET and POST methods [40]. Additionally, the Azure Search Service uses the REST API to upload, process and query the search data of the British Library project [35].

Using the model-view-controller design pattern, as well as the REST based architecture, makes our infrastructure easy to extend and adapt to different data-driven organisations. As we have distinguished the different components of the application, we can easily modify the current functionalities or add new functionalities which can be used without requiring significant changes to the whole project.

Furthermore, since the solution incorporates the use of the Entity Data Model which deals with; the challenge of ensuring code maintainability and efficiency; obtain efficient access to data; and handling storage and scalability capabilities in the most resourceful way. The Entity Data Model also addresses issues regarding data that are transferred from one form to another. This ensures that the design and development of an application will not depend on any stored form of data, which deals with the issue of having the stored data in different forms such as in relational databases and XML files [41].Therefore our project does not depend on any stored of data and can handle different forms of data.

Profit-driven organisations as well as research-based institutes such as libraries, museums and hospitals that are similar to the British Library, can easily use our solution with minor changes in order to provide search functionality throughout their data. This requires minimal changes in the code, which are mostly related to the data schema and the Model entities used in the solution.

6. Testing and Evaluation

"Program testing can be a very effective way to show the presence of bugs, but it is hopelessly inadequate for showing their absence" [42]

In today's emerging technological world it is highly likely that even the simplest software can hide high levels of complexity. The characteristic of the high complexity can be explained through a set of criteria [43]:

• **Interaction**, the diverse components that a system interacts with
• **Autonomy**, components are autonomous but subject to certain laws in order to interact
• **Emergence**, the autonomous behaviour of a component is unpredictable when interacting with numerous other component
• **Far from equilibrium**, systems usually do not fully recover after a disruptive event
• **Nonlinear**, a small set of inputs can cause extreme events
• **Self-organisation**, some systems can re-organise themselves after unexpected events
• **Coevolution**, autonomous components tend to constantly emerge.

The need to satisfy software qualities, combined with the customer's need for rapid delivery in order to meet the market requirements, makes Software Testing one of the most important aspects of a project.

The major benefit of testing in our project was to ensure that the BL Web Portal does what it is intended to do, which is to meet user requirements, and does not behave unpredictably [45]. In addition, testing enabled us to prevent and identify at an early stage faults and errors in the source code that might lead to software failures. In fact, testing made it easier, cheaper and quicker to fix errors if they were to arise.

As it was already discussed, the project applied agile methodology where at the end of each iteration features were tested and integrated to the existing application and subsequently system testing was carried out. One can argue that this testing methodology can have negative impact on the testing plan of a project since testing was only possible at the end of each iteration. However, following the Test Driven Development (TDD) approach cannot be considered as the ideal approach due to the emerging requirements of the project.

6.1 Software Metrics

A software metric is a mapping of the entities of a software system to numerical metric values. Quantitative measurements of the source code were an important part of the project for maintaining the quality of the product since it is easier to analyse, understand and control something that you have measured. Software metrics were produced on a weekly basis in order to evaluate, assess, improve and predict possible bugs. For the purpose of the project the following metrics were used; maintainability index, cyclomatic complexity, depth of inheritance, class coupling and lines of code. The code metrics of the solution were produced using the build-in tool available in Visual Studio.

6.1.1 Maintainability Index

Maintainability index is a metric value ranging from 0 to 100, indicating how easy it is to maintain the source code. During the various iterations of the project, the aim was to keep the maintainability index at high levels since the higher the index, the easier it was to maintain the code. The maintainability index is classified into 3 levels; green (good maintainability, 20 – 100), yellow (moderated maintainability, 10 – 19) and red (low maintainability, 0-9). The index of our solution is 82, which indicates high maintainability. [46]

6.1.2 Cyclomatic Complexity

Cyclomatic complexity is a source code measurement indicating the structural complexity of the code. This metric can be produced using the Control Flow Graph of the code and it measures the number of linearly-independent paths of a module. High Cyclomatic complexity means that the source code has complex control flow and requires more testing to achieve adequate coverage. In addition, complex source code is less maintainable [46]. The cyclomatic complexity of our code is 802, which is accepta-

ble and reasonable according to the project's Lines of Code (LOC).

6.1.3 Depth of Inheritance

Depth of Inheritance (DIP) is an object-oriented metric and is defined as the maximum length from a class definition (node) to the root of the tree [47]. DIP metric depends on three fundamental assumptions. Firstly, the deeper a class is the more methods it will inherit, a fact that makes its behaviour unpredictable. Secondly, the design complexity of a module becomes more complex when the tree is deeper meaning that a high number of classes and methods are involved. Lastly, deeper classes in the tree hierarchy have a greater potential of using inherited methods. Based on these assumptions a middle ground between, performance provided by inheritance and complexity that increases with depth, must be found. A value of between 0 – 4 can be considered as an ideal level of this metric. The depth of inheritance of our project is 3, which is a value indicating low levels of complexity. [47]

6.1.4 Class Coupling

Class coupling measures the coupling to unique classes through parameters, local variables, return types, method calls generic or template instantiations, base classes, interface implementations, fields defined on external types, and attribute decoration [47]. Good software practice indicates that types and methods should have high cohesion and low coupling. The class coupling metric for our project is 225. Though this may be seem like a slightly high value, this number is acceptable when considering the different APIS, libraries used and the Lines of Code (LOC)

6.1.5 Lines of Code

The LOC metric indicates the total number of source code in the entire solution. In relatively simple applications, increased LOC might indicate that object-oriented principles are not followed and thus the code becomes more complex. The BL Web Portal project has 3,312 LOC; this can be considered reasonable when taking into consideration that the design pattern lies on the MVC paradigm.

Maintainability Index ▲	Cyclomatic Complexity	Depth of Inheritance	Class Coupling	Lines of Code
82	802	3	225	3,312

Figure 4. Code Metrics of the Solution.

6.2 Unit Testing

Part of the testing strategies that have been followed involved unit testing. Unit testing refers to the decomposition of a program into units, where each unit is independently tested. This approach is usually followed in order to isolate and test classes or methods of a solution. It is a white-box testing approach requiring a set of test inputs in order to be executed.

Unit testing was considered as an ideal approach for the British Library Big Data Experiment since agile methodology was used. That is, at the end of each iteration the latest functionality of the system was tested separately as a stand-alone unit prior the integration with the entire system.

During the project lifecycle a test log document was produced describing all the tests that have been successfully executed. This technique is widely used when the unit testing approach is followed and it enables the early identification and tracing of errors. That is, after integrating a new component to the existing software, all the unit tests were executed again. In the case of an error, the test log carrying all the unit tests facilitated the early discovery and removal of issues in the source code. As a consequence, a more smooth integration of the several components of the system was achieved.

The purpose of the first test cases was to test whether or not the controllers return the appropriate views. That is, the appropriate web page must be rendered when the associated controller is invoked.

The next step was to test and investigate the behaviour of the software modules under different circumstances. In order to test and validate the correctness of the software, approximately 40 unit tests were produced and executed

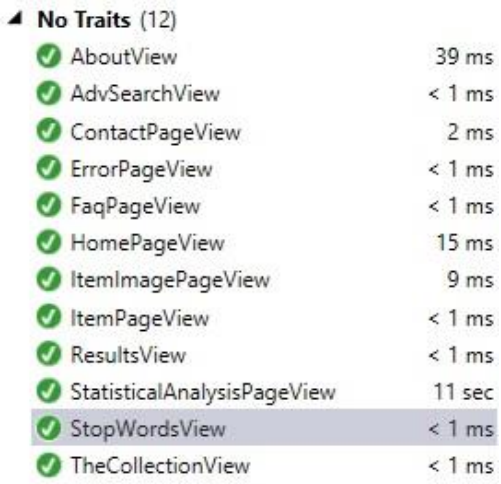


Figure 5. Unit Tests.

Unit testing was particularly instrumental in solving an issue with the source code during instances where several blobs located in the Cloud did not have any content. More specifically, when the StatisticalAnalysisController attempted to download the OCR text for a document from the corresponding blob, and the OCR text file was not included, the system generated an error. Without unit testing this type of error would have been incredibly difficult and time consuming to resolve. This indicated that try-catch statements should be added when the OCR text is required in order to redirect the user to an error page when the OCR text is not available.

6.3 Code Coverage

Code coverage is a code metric referring to the amount and degree to which the source code of a software has been tested. High code coverage often indicates high source code quality.

Even though it is not feasible to achieve 100% code coverage, the unit tests were designed in such a way to cover as much code as possible. All of the unit tests successfully passed by covering at least 70% of the code. Simple blocks of the code achieved 100% code coverage. After executing all the tests, the overall code

coverage was 84%, which is indicative that the code had been thoroughly investigated and tested.

6.4 Metadata Collector/Validator

At the beginning of the project, the British Library provided the team with a subset of data from the digitised collection. This file contained JSON-encoded entities, one per line. Each entity encapsulated useful information in key-value pairings for each image held within the British Library azure table storage. However, these JSON entities were not in flat structure and contained nested fields. An example of a non-flat JSON file is demonstrated below.

```
{"data": {
  "title": "A good story",
  "date": "1821",
  "nested_Fields": {
    "other_fields": [
      {"value": "key", "key": "key"},
      {"value": "key", " key": "key"},
      {"value": "key", "key": "key"}
    ]
  }
}}
```

Subsequently, these data were not compatible with Azure Search Service API which only supports flat structure entities. In order to tackle this problem, a small application was developed to create valid data that can be queried by Azure Search Service API. This application enables the user to access containers in the British Library azure table storage and consequently creates a new valid JSON file containing all the metadata for each blob.

Each blob URL has the following structure:
http://blmc.blob.core.windows.net/1851/0000004720_1851_covers.jpg
Each segment of the blob URL corresponds to the following general structure:

http://blmc.blob.core.windows.net/Date/printsysnum_Vol_Date_sizebracket.jpg

The application parses the URL and collects the Date, Printsysnum, vol and sizebracket of each blob. Subsequently, using the combination of printsysnum_vol as a unique identifier, it searches in the original file provided by the British Library in order to find the remaining metadata of an entity e.g. location, author that cannot be found in the blob. Using this method, a valid JSON file is created which can now be used by Azure Search Service.

6.5 Cross-Platform Compatibility

A web browser is the gateway to the BL Big Data Experiment project. Nowadays, with the advent of modern web browsers the cross-browser compatibility issue is becoming increasingly important. That is, browsers do not behave identically and render web content differently.

In order to ensure that users coming from different browsers experience the same behaviour from the portal, it was necessary to test each feature upon its completion by using different browsers. For the purpose of compatibility tests, the team decided to use the SeleniumHQ and NUnit frameworks. SeleniumHQ is software testing framework for automating web browsers whereas NUNIT is a unit testing framework. Therefore, several unit tests were developed in order investigate the compatibility of the web portal with different browsers. This approach was considered as the most appropriate based on the strict time constraints of the project. By using SeleniumHQ we managed test many browsers running on different operating systems in a timely manner. This was achieved by simply changing the Webdriver capabilities in the SeleniumHQ source code. The example below shows how a test was executed using IE 9.0 under Windows 7, followed by a table indicating the results of our tests for the most commonly used browsers.

```
capability.SetCapability("browser", "IE");
capability.SetCapability("browser_version", "9.0");
capability.SetCapability("os", "Windows");
capability.SetCapability("os_version", "7");
```

6.6 Scalability and performance

In addition to the functional requirements tests, in order to completely validate our solution it was necessary to test how robust and fail proof the web portal is. This was achieved by executing scalability and performance tests. The performance of a system refers to how quickly it responds to requests whereas scale measures how many users can be served simultaneously by the system (Throughput). [48].

The test scenarios for scalability and performance evaluation were designed to put the system under an unrealistic load of requests. When designing scalability and performance tests all the components of the infrastructure should be taken into consideration. In our case, for the Azure Search Service, several tests were executed each one using different amounts of data available on the server.

With the aim of executing scalability and performance tests, the Apache JMeter [49] loading testing tool was used. JMeter, in a graphical user interface environment, enabled us to run several tests of the web portal by sending an unrealistic and concurrent number of requests. Apart from that, JMeter facilitated the analysis of those tests by producing graphical representations of how the system responds over time with different amounts of requests (See Appendix G).

As a starting point for our testing strategy, the system was tested under reasonable load levels. That is, for the initial tests the data to be indexed incorporated 1000 documents and a small amount of query requests were executed. After the preliminary tests, further experiments were executed where each time the number of documents on the server and the number of concurrent requests was increased. For the final tests the data to be indexed incorporated 10 000 documents which is the maximum limit for the free subscription for Azure Search Service. In addition, the system was tested by sending 2000 requests and by sending 500 concurrent query requests.

After executing the aforementioned tests, by using JMeter graphical analysis it was easier to make conclusions on the scalability and performance of our system. As it can be observed in Appendix H the system performance and response time is stable regardless of the number of documents and concurrent requests. The response time of the system and throughput it is almost linear. The results of those tests exploited how scalable and responsive Azure Search Service is. The only limitation of the system is that due to the free subscription used, the index service expires after 100 query requests.

7. Discussion

This section critically assesses and analyses the software methodologies used in order to produce the final deliverable. In addition, we discuss potential pitfalls and strengths of our methodology and software solution.

7.1 Change in Technology Stack

Upon its initiation, the project's goal was to develop an infrastructure that will enable users to dynamically perform queries among heterogeneous data. This had to initially be achieved regardless of the technology used. The developing team firstly attempted to tackle this problem using Azure Table Storage. However, mid-project the requirements changed regarding the specific technology used. As a result, the team had to re-establish the technology requirements. In order to use the new technology, it was necessary to perform an in-depth literature review to understand how to cope with the demands of the new requirements.

7.2 Project Methodology

As it was already discussed in section 3.4 agile methodology was used which falls in the IID methodologies. At the end result this decision was beneficial for the team, because through the close and frequent communication with client the team managed to deliver all the agreed functional requirements.

More specifically, when the requirements changed as it was described in section 7.1, the agile approach facilitated the immediate interaction with the client in order to identify and discuss these changes. This had several advantages since the customer could be immediately informed leaving no windows of misunderstandings.

7.3 Requirements Elicitation

. Requirements elicitation was one of the most important aspects of the project initiation phase. During this stage to fully understand the requirements we used the following techniques:

- Literature Reviews
- User focus groups
- Live Documents
- UML and Use case Scenarios
- Goal Models

As it is evident from the above, the team performed more than one requirements elicitation technique. This enabled us to collect the system requirements from all the stakeholders. As a result a concrete understanding of users' needs was achieved.

However, the authors of the paper strongly believe that the most important technique was the user focus groups where the developing team had the chance to interact with potential users of the system and expert domains. This was extremely useful because we gained a better understanding regarding exactly what individuals expect from the system. Domain experts also advised us on what is currently missing and what will be a real contribution to the digital humanities.

Overall, these techniques enabled us to validate and verify the requirements and eventually produce a successful deliverable.

7.4 Azure Search Service

Azure Search Service is the major component that the project incorporates in order deliver the final solution. As a RESTful API, it provides the Cloud infrastructure that enables us to dynamically create different queries and eventually search the British library digitised collection. In addition, using this API the queries are not predefined and can be constructed depending on user input. That is, the queries are facilitated by the use of logical operators (and, or, not) and comparison expressions (eq, ne, gt , lt , ge, le) from the OData expression syntax a variety of searches.

In addition to these capabilities, the Azure Search Service provides a ranking functionality on the search results. In contrast to other search services discussed in the related work section this functionality is provided directly from the API without requiring the developers to do additional work.

However the service comes with some constraints:

• Maximum limits for shared (free) Search service

The free subscription of Azure Search Service provides some limitations to the subscribers. Azure subscribers who are on the no-cost plan cannot use this service in its full potential. This was the case for our project; the solution incorporates the maximum limit for free service which is up to 10 000 documents, thus limiting our project in terms of examining the capabilities of the service under real world (industrial circumstances).

• Response Sizes

An additional limitation to the project concerning Azure Search Service is the number of return results. The service only allows the retrieval of a maximum of 1000 documents per search page. To overcome this limitation our approach used the $skip1000 operation in order to eventually receive all the search results.

• A Query limitation

As it was already discussed in the Approach section Azure Search Service cannot simultaneously provide Boolean operators and word breaking on the fields. However in the proposed solution, the authors provide an implementation that acts as a workaround in order to solve this problem.

7.5 OCR Search

One possible limitation of our system is its performance when the OCR is included in the search. This is due to the fact that the software goes through a huge set of data. This observed limitation can be an avenue for future research and implementation.

7.6 Non – functional requirements

. In the Testing and Evaluation section, several tests were executed in order to examine the behaviour of the system under extreme conditions of workload. One pitfall of those experiments is that the current solution is based on a no-cost subscription. As a consequence, the team was not able to thoroughly investigate the performance of the Azure Search Service. Unfortunately, the executed tests were only testing the 'limits' of the no-cost version. However, from the results it can be observed that the system is not affected when a large number of requests is received.

7.7 Testing

A software system is nearly impossible to be fully tested. However, in order to achieve maximum code coverage, the team tried to generate many test cases. We also aimed to achieve as much as possible in terms of integration testing. However, one tool that was not used because the team discovered it at the latest stages of the project is SONAR. This tool produces results and analytics on the several versions of a project and helps the developers to manage quality of the code. SONAR is an excellent tool that would have helped the team in integrating the several versions produced.

7.8 Learning Curve

Upon reflection, conducting this project and implementing the British Library Web Portal l has been a challenging yet rewarding experience. It was a great opportunity for each team member to familiarise themselves to a project of industrial size. Although the project was extremely demanding, in terms of coordinating and balancing the workload fairly among team members, it provided us with the opportunity to learn how to effectively work and communicate in a team setting. Moreover for the deployment of the project, a lot of background research was conducted to provide a broad understanding about various other techniques and how they have been used in practice.

8. Future Work

The primary goal of the British Library Big Data Experiment was to provide an improved method for accessing, analysing and delivering data from heterogeneous collections in a unified structure. This was achieved by exploiting the capabilities of Azure Search Service API under numerous design patterns that make the solution an extensive application framework. Due to the limited time constraints and strict project deadlines certain areas were not investigated, nevertheless, these may be considered as avenues for future research and development. The following examples may be applied to enhance the overall system's functionality:

- Experiment with more data

As it was already mentioned, for the purpose of this project the maximum number of documents queried by the system is 10,000. This is because the system operates under the free trial plan of Microsoft Azure Search Service. The standard pricing tier allows the use up to 25 million documents, up to 25 indexes and up to 600 GB of storage [35]. This provides the potential for future experiments under industrial conditions, which can examine how the Azure Search Service can cope under high levels of workload.

- Hadoop Optimisations

The current implementation does not incorporate the use of Hadoop framework since optimisation was not the primary goal of the project. However, as this is a data driven project, in future developments Hadoop jobs can be integrated with the current solution in order to optimise the analytics provided by the system. For example, MapReduce jobs can help in the faster processing of the OCR search.

- Search Refinement

When performing a search, the results page at its current state enables the user to execute new queries based on a specific field of an item. For example, if a document has the publication date of 1896 the user is allowed to click on this field and instead of the system executing a query only on the initial results, all documents from the entire collection that were published in the year 1896 are fetched. This gives the notion of a minimal search refinement. A possible future development is the implementation of a custom search refinement where users are able to narrow down their initial search results based on different fields such as publication location, publisher name, language, genre, creation date, material type, and author/contributor.

9. Conclusions and Contributions to the field

9.1 Contributions to the field

The team by exploiting cutting edge technologies through the Microsoft Azure Cloud Infrastructure, successfully delivered to the client the British Library Web Portal. After conducting our literature, we found that this project, provides some contributions to the field: (1) Developing an open framework that supports data integration among heterogeneous collections and (2) Defining an infrastructure that allows the user to dynamically create his own queries and (3) and produce results in a unified structure.

9.2 Conclusion

Despite the several challenges and the change of technology stack during the mid – session, the team managed to deliver a fully functional solution. In order to achieve that, the entire software engineering cycle was undertaken.

Even though the projected ended, we hope that we opened a whole new spectrum for future research.

Lastly, facing real life problems and challenges, one of the most rewarding experience so far. This will equipped us for our future careers will all the necessary tools in order to withstand.

We would like all the UCL, British Library, and Microsoft for their support throughout the project.

Appendix A

Focus Groups Plan

Item 1 – Introductions

13:00 Arrival of attendees

13:15 Introduce the team

Brief project

Brief focus group:

- Ask a few questions
- Tea and coffee provided, there will breaks
- There are some exercises
- Request permission for photos and all discussions from focus group

Attendees introduce themselves:

- Short backgrounds (not research)
- Guilty pleasure ice breaker

Item 2 – Focus Group Questions

13:35 Split into two groups and distribute paper etc.

Begin focus group questions

Q1. What attracted you to the project/brought you to come to this focus group?

Q2. What is the current system?

Q3. What do you like about it, and/or problems they face and the causes?

14:15 SHORT BREAK (5 minutes)

14:20 ***Exercise:*** *Card Sorting (15 minutes)*

- Show example of search steps for poetry
- Ask participants to list their steps on the cards (numbered!)

Q4. Give example of searching with fields:

- If given title, and author, with 2/3 other fields for searching:
  - What would they be? (what query search fields)
  - How will they use the fields together to optimise your results?
- When performing search, from returned results, what extra information would you like to be presented from this?
  - Voyant (text tools)
  - Statistical analyses (i.e. keyword appears in single item a certain number of times)

Q5. Have you ever used statistical analysis tools for your research?

- If yes, what they used it for.
  - Most frequently used tool.
- If no, do you see any advantage in using this?

14:55 LONG BREAK (15 minutes)

15:10 ***Exercise:*** *Flipchart brainstorm (for Q6.)*

Q6. If a recommender engine was implemented, what criteria would you like it to apply for recommendations?

- Searching for a specific author, which recommendations are best related?
  - Same author
  - Similar topics
  - Time period
  - Combination? Weighting of combination?

Q7. A feature we are thinking of implementing is the traceability of your research:

- Ability to traverse back through steps taken and take alternative path - like a tree with branches
- Such as a computer folder path: C:/Documents/Research
- E.G. a humanities context: Time period/geographical location/specific topic
- May help future research to reproduce steps, and be used in machine learning
- Would a history of your search be useful? How would you use this?

15:30 SHORT BREAK (5 minutes)

15:35 Q8. Big data discussion:

- Ask if familiar with big data, if not briefly explain:
  - Collection of large, rapidly growing data sets
  - In this context, collections of books and information on Microsoft Azure HDInsight
- First thing that comes to mind when you hear the term 'big data'?
- How would you want to see big data used in your research?

15:45 Feedback and any questions from attendees.

.

N.B. All timings are approximate.

Appendix B

Use Case Scenarios

Use Case: Advanced Search Feature
Precondition: A user of the British Library Portal accesses the Advance Search Feature page.
Postcondition: The relevant items are displayed to the user.
Description: The system enables to user to browse the British Library Resources using advanced search features.

Main Success Scenarios

Success Scenario 1: Keyword Search.

1. The user enters a keyword in the Keyword Search text field.
2. The user clicks on the search button.
3. The BL portal retrieves the relevant items.
4. The BL portal returns the relevant items.

Alternative Scenario

1a. The user does not enter a keyword in the Keyword Search text field.
1b. The system does not return any documents.

Success Scenario 2: Keyword Search on a specific field.

1. The user enters a keyword in the Keyword Search text field.
2. The user selects a specific field, where the keyword search will be applied.
3. The user clicks on the search button.
4. The BL portal retrieves the relevant items.
5. The BL portal returns the relevant items.

Success Scenario 3: Exact matching search on a phrase.

1. The user enters a phrase in the Exact Matching Search text field.
2. The user clicks on the search button.
3. The BL portal retrieves the relevant items.
4. The BL portal returns the relevant items.

Success Scenario 5a: Boolean Operations Exclusion Case 1 (NOT).

1. The user enters a keyword in the Keyword Search text field.
2. The user enters a keyword to be excluded from the search in the None of These Words text field (NOT).
3. The user clicks on the search button.
4. The BL portal retrieves the relevant items.
5. The BL portal returns the relevant items.

Alternative Scenario

1a. The user does not enter a keyword in the Keyword Search text field.

1b. The user enters a keyword to be excluded from the search in the None of These Words text field (NOT).
1c. The system does not return any documents.

Success Scenario 5b: Boolean Operations Exclusion Case 2 (NOT).

1. The user enters keyword in the Keyword Search text field.
2. The user enters keywords to be included from the search in the Any of These Words text field (OR).
3. The user enters keywords to be excluded from the search in the None of These Words text field (NOT).
4. The user clicks on the search button.
5. The BL portal retrieves the relevant items.
6. The BL portal returns the relevant items.

Alternative Scenario
1a. The user does not enter a keyword in the Keyword Search text field.
1b. The user does not enter keywords to be included from the search in the Any of These Words text field (OR).
1c. The user enters a keyword to be excluded from the search in the None of These Words text field (NOT).
1d. The system does not return any documents.

Success Scenario 5c: Boolean Operations Exclusion Case 3 (NOT).

1. The user enters keywords to be included from the search in the Any of These Words text field (OR).
2. The user enters keywords to be excluded from the search in the None of These Words text field (NOT).
3. The user clicks on the search button.
4. The BL portal retrieves the relevant items.
5. The BL portal returns the relevant items.

Alternative Scenario
1a. The user does not enter keywords to be included from the search in the Any of These Words text field (OR).
1b. The user enters a keyword to be excluded from the search in the None of These Words text field (NOT).
1c. The system does not return any documents.

Success Scenario 5d: Boolean Operations Case 1 (Either Term or Both (OR)).

1. The user enters keyword in the Keyword Search text field.
2. The user enters keywords to be included from the search in the Any of These Words text field (OR).
3. The user clicks on the search button.
4. The BL portal retrieves the relevant items.
5. The BL portal returns the relevant items.

Success Scenario 5e: Boolean Operations Case 2 (Either Term or Both (OR)).

1. The user enters keywords to be included from the search in the Any of These Words text field (OR).
2. The user clicks on the search button.
3. The BL portal retrieves the relevant items.
4. The BL portal returns the relevant items.

Use Case: Data Analysis

Precondition: The user clicks on an item retrieved from the Advances Search Feature.

Postcondition: Download an analysis report.

Description: The system performs statistical analysis on a specific item.

Main Success Scenario

Success Scenario (Frequent Words):

1. The user clicks on the Frequent Words button.
2. The BL portal retrieves the most frequent words for the specific item.
3. The BL portal returns and displays to the user a list with the frequent words.

Success Scenario (Frequency of a specific term):

1. The user enters a specific term (single term) in the Frequency of a Term text field.
2. The user clicks on the Retrieve Frequency button.
3. The BL portal retrieves the frequency of the provided term in the specific document.
4. The BL portal returns and displays to the user the frequency of the provided term in the specific document.

Alternative Scenario

3a. The provided term does not exist in the specific document

Success Scenario (Frequency of a bigram):

1. The user enters a bigram in the Bigram Frequency text field.
2. The user clicks on the Retrieve Bigram Frequency button.
3. The BL portal retrieves the frequency of the bigram in the specific document.
4. The BL portal returns and displays to the user the frequency of the bigram in the specific document.

Alternative Scenario

3a. The provided bigram does not exist in the specific document

Success Scenario (Frequency of a trigram):

1. The user enters a trigram in the Trigram Frequency text field.
2. The user clicks on the Retrieve Trigram Frequency button.
3. The BL portal retrieves the frequency of the trigram in the specific document.
4. The BL portal returns and displays to the user the frequency of the trigram in the specific document.

Alternative Scenario

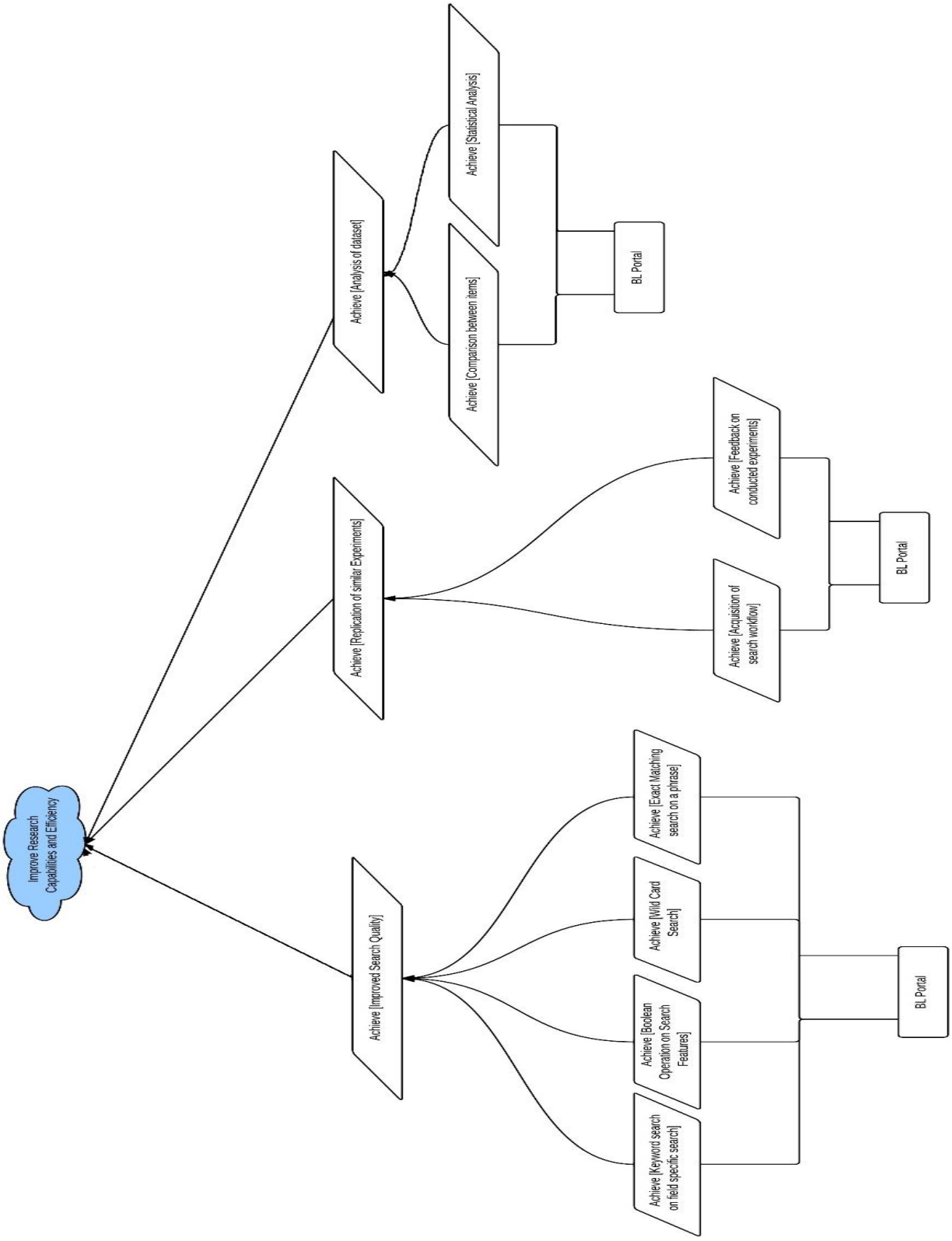
3a. The provided trigram does not exist in the specific document

Success Scenario (Adjacent words for the most frequent terms):

1. The user clicks on the Part of Speech of Frequent Words button.
2. The BL portal retrieves the most frequent words.
3. The BL portal retrieves the adjacent words for the most frequent words.
4. The BL portal returns and displays to the user a list with the adjacent words for them most frequent terms.

Appendix C

Goal Model



Appendix D

Product Specific Requirements

Product Specific Reqs

ID	Description	Priority (MoSCow)	Notes & Updates
RQ01	The user must be able to conduct a search in order to retrieve items from the digitalised collection, using the following fields: > Keywords (in both abstract (if exists) & document), > Title, > Author, > Contributors, > Publication date (with advanced filters, e.g. last year), > Publication place, >Publisher	M	
RQ02	Users should be able to perform advanced search with boolean operations (AND, OR and NOT).	M	
RQ03	The user should have the ability to cite the search with the opportunity to store their search, like a string of text, recording the search method. (Step Traceability)	M	URL that can cut and paste to get to the search results
RQ04	The user should be able to retrieve a chosen number of the most frequent words of a document.	S	
RQ05	The user should be able to retrieve the frequency of a specific word in a document.	S	
RQ06	The user should be able to retrieve the frequency of a bigram and a trigram.	S	
RQ07	The user should be able to retrieve the adjacent word for the most frequent terms of a document	S	
RQ08	The users couldbe able to package and download desired data.	C	

Appendix E

Project Plan

ID	Task Mode	Task Name	Duration	Start	Finish	Predecessors	Resource Names
1							
2		**BL Big Data Experiment**	**80 days**	**Mon 02/06/14**	**Fri 19/09/14**		
3							
4		**Project initiation**	**4 days**	**Mon 02/06/14**	**Thu 05/06/14**		
5		Team introductions	2 days	Mon 02/06/14	Tue 03/06/14		
6		Project kick-off	1 day	Wed 04/06/14	Wed 04/06/14	5	
7		Identify Stakeholders	1 day	Thu 05/06/14	Thu 05/06/14	6	
8							
9		**Reqs Gathering**	**14 days**	**Fri 06/06/14**	**Wed 25/06/14**		
10		Focus group session	1 day	Fri 06/06/14	Fri 06/06/14	7	
11		Define Reqs	3 days	Mon 09/06/14	Wed 11/06/14	10	
12		High Level Reqs	2 days	Thu 12/06/14	Fri 13/06/14	11	
13		Risk analysis	2 days	Mon 16/06/14	Tue 17/06/14	12	
14		Review Reqs	3 days	Wed 18/06/14	Fri 20/06/14	13	
15		Reqs sign-off	3 days	Mon 23/06/14	Wed 25/06/14	14	
16							
17		High Level Analysis & Design	2 days	Thu 26/06/14	Fri 27/06/14	15	
18							
19		**Implementation of Must Reqs**	**10 days**	**Mon 30/06/14**	**Fri 11/07/14**		
20		**Iteration 1**	**5 days**	**Mon 30/06/14**	**Fri 04/07/14**		
21		Implementation	2 days	Mon 30/06/14	Tue 01/07/14	17	
22		Integrate	1 day	Wed 02/07/14	Wed 02/07/14	21	
23		Testing	1 day	Thu 03/07/14	Thu 03/07/14	22	
24		Evaluate	1 day	Fri 04/07/14	Fri 04/07/14	23	
25		**Iteration 2**	**5 days**	**Mon 30/06/14**	**Fri 04/07/14**		
26		Implementation	2 days	Mon 30/06/14	Tue 01/07/14	17	
27		Integrate	1 day	Wed 02/07/14	Wed 02/07/14	26	
28		Testing	1 day	Thu 03/07/14	Thu 03/07/14	27	
29		Evaluate	1 day	Fri 04/07/14	Fri 04/07/14	28	
30		**Iteration 3**	**5 days**	**Mon 07/07/14**	**Fri 11/07/14**		
31		**Implementation**	**2 days**	**Mon 07/07/14**	**Tue 08/07/14**		
32		Reqs refinement	2 days	Mon 07/07/14	Tue 08/07/14	24	
33		Integrate	1 day	Wed 09/07/14	Wed 09/07/14	31	
34		Testing	1 day	Thu 10/07/14	Thu 10/07/14	33	
35		Evaluate	1 day	Fri 11/07/14	Fri 11/07/14	34	
36		**Iteration 4**	**5 days**	**Mon 07/07/14**	**Fri 11/07/14**		

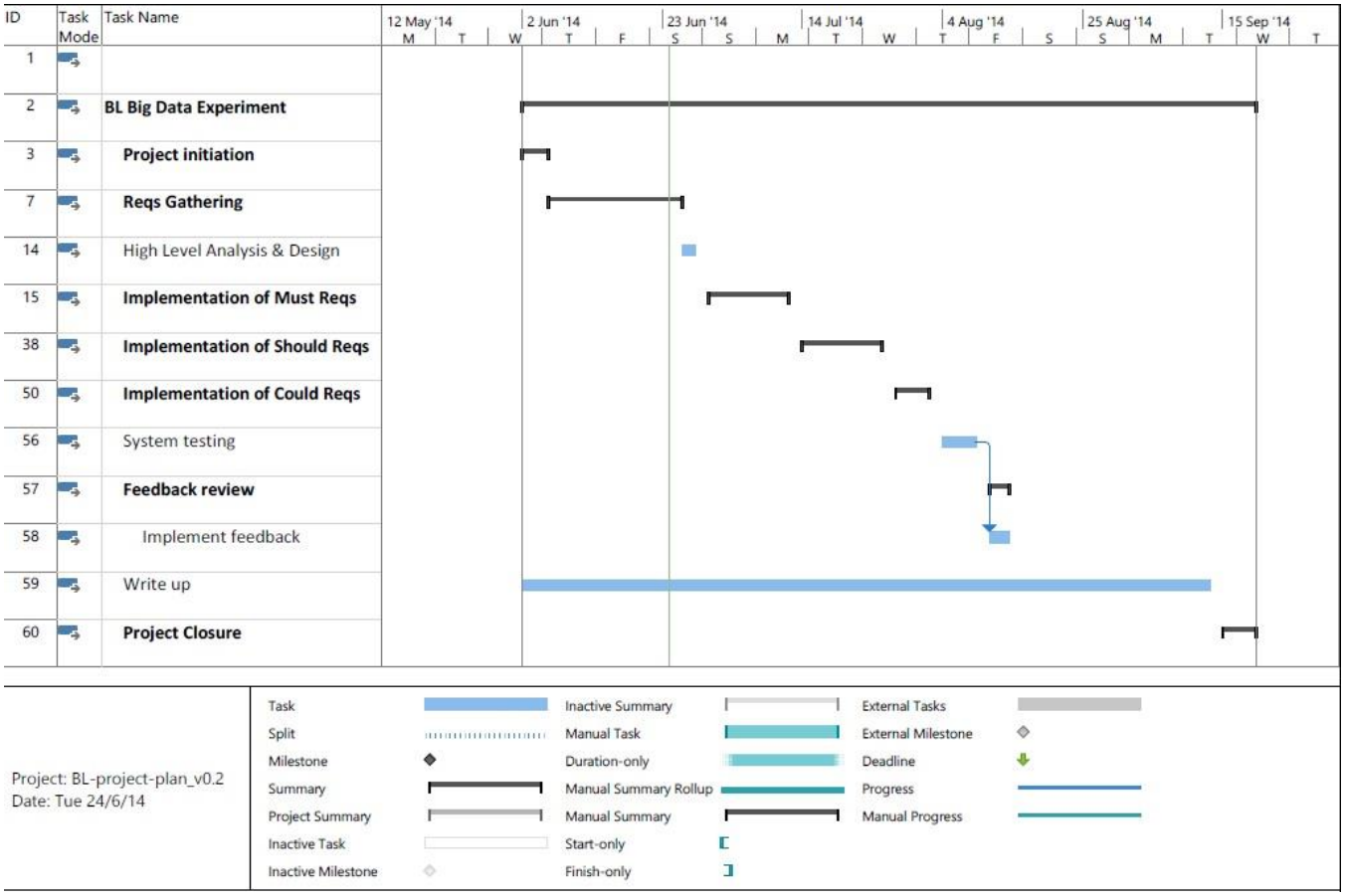
Appendix E

Project Plan (Continue)

ID	Task Mode	Task Name	Duration	Start	Finish	Predecessors	Resource Names
37		**Implementation**	**2 days**	**Mon 07/07/14**	**Tue 08/07/14**		
38		Reqs refinement	2 days	Mon 07/07/14	Tue 08/07/14	29	
39		Integrate	1 day	Wed 09/07/14	Wed 09/07/14	37	
40		Testing	1 day	Thu 10/07/14	Thu 10/07/14	39	
41		Evaluate	1 day	Fri 11/07/14	Fri 11/07/14	40	
42							
43		**Implementation of Should Reqs**	**10 days**	**Mon 14/07/14**	**Fri 25/07/14**		
44		**Iteration 1**	**5 days**	**Mon 14/07/14**	**Fri 18/07/14**		
45		Implementation	2 days	Mon 14/07/14	Tue 15/07/14	41	
46		Integrate	1 day	Wed 16/07/14	Wed 16/07/14	45	
47		Testing	1 day	Thu 17/07/14	Thu 17/07/14	46	
48		Evaluate	1 day	Fri 18/07/14	Fri 18/07/14	47	
49		**Iteration 2**	**5 days**	**Mon 21/07/14**	**Fri 25/07/14**		
50		**Implementation**	**2 days**	**Mon 21/07/14**	**Tue 22/07/14**		
51		Reqs refinement	2 days	Mon 21/07/14	Tue 22/07/14	48	
52		Integrate	1 day	Wed 23/07/14	Wed 23/07/14	50	
53		Testing	1 day	Thu 24/07/14	Thu 24/07/14	52	
54		Evaluate	1 day	Fri 25/07/14	Fri 25/07/14	53	
55							
56		**Implementation of Could Reqs**	**5 days**	**Mon 28/07/14**	**Fri 01/08/14**		
57		**Iteration 1**	**5 days**	**Mon 28/07/14**	**Fri 01/08/14**		
58		Implementation	2 days	Mon 28/07/14	Tue 29/07/14	54	
59		Integrate	1 day	Wed 30/07/14	Wed 30/07/14	58	
60		Testing	1 day	Thu 31/07/14	Thu 31/07/14	59	
61		Evaluate	1 day	Fri 01/08/14	Fri 01/08/14	60	
62							
63		System testing	5 days	Mon 04/08/14	Fri 08/08/14	61	
64							
65		**Feedback review**	**3 days**	**Mon 11/08/14**	**Wed 13/08/14**		
66		Implement feedback	3 days	Mon 11/08/14	Wed 13/08/14	63	
67							
68		Write up	75 days	Mon 02/06/14	Fri 12/09/14		
69							
70		**Project Closure**	**5 days**	**Mon 15/09/14**	**Fri 19/09/14**		
71		Organise project closure meeting	1 day	Mon 15/09/14	Mon 15/09/14	68	
72		Project team meeting	1 day	Tue 16/09/14	Tue 16/09/14	71	

Appendix E

Project Plan (Continue)



Appendix F

Risk Analysis

		Owner			Date			Risk Rating						
Ref No.	Keywords	MSc	MS	BL	Work Package	Open	Due	Closed	Likelihood	Time	Performance	Overall Rating	Description	Risk Level	Impact
R01		All							3	3	2	9	Team members designing the software are inexperienced.	Medium	Development cycles are extended.
R02		All							3	5	3	15	Weak team communication.	High	Project milestones missed and overall project delay.
R03		Stefan							3	4	5	15	Poor communication with the stakeholders.	High	Failure to capture user needs. Project failure. Stakeholder's dissatisfaction.
R04		Nektaria							3	4	4	12	Ineffective project plan - underestimate the workload of tasks.	High	Project delay which can evolve to project failure.
R05		All							4	5	5	20	Inability to complete critical milestones.	High	Project delay which can evolve to project failure.
R06		All							3	4	3	12	Incomplete milestone dependancies.	Medium	Failure to meet the agreed requirements. Missed deadlines and subsequently milestones delayed.
R07		All							5	5	5	25	Poor capture of user requirements - over ambitious, non-specific requirements	High	Failure to capture all the requirements. Project failure.
R08				James, Adam					3	4	3	12	Client changes requirements.	Medium	Rescheduling overall project workflow.
R09			All						3	3	3	9	Limitations of third party libraries, frameworks or tools.	Medium	Increased workload overhead. Project delay.
R10		Stelios							4	4	4	16	Weak implementation of testing plan.	High	Poor software quality which can lead to undesired and unexpected outputs.
R11		All							1	3	2	3	Ineffective code collaboration.	Medium	Inconsistent code. Code duplication. Integration failures. Integration leaks. Code smell.
R12		Wendy							2	3	3	6	The users (mostly non-technical) find the prototype hard to use, reuse and learn.	Low	Dissatisfication of stakeholders.
R13		All	All	All					1	3	3	3	Unexpected events.	Low	Project delay which can evolve to project failure.

Appendix F

Risk Analysis (Continue)

Mitigation	Contingency Plan	Notes & Updates
• Training. • Read resources, materials, relevant to the project scope and domain. • Train on tools and frameworks that are vital for the project development cycle.	• Read additional resources, materials, relevant to the project scope and domain. • Schedule urgent additional sessions on tools and frameworks that are vital for the project development cycle. • Get advice from domain experts and supervisor.	9/06/2014 Everyone agreed that it is best, we agreed to review.... 13/06/14 The work was carried out.....
• Effective meeting scheduling. • Exchange communication channels (e.g. e-mail, telephone etc.) and use accordingly. • Team bonding.	• Schedule team administration. • Get advice from supervisor. • Team bonding. • Team leader discusses the difficult person's problem one to one. • Clarify that all problems should be addressed to the team leader as a first point of contact.	
• Well structured preparation for meetings. • Schedule regular meetings with stakeholders. • Use visual aids.	• Escalate issue to project supervisors and project sponsor.	
• Thoroughly investigate similar project plans. • Review initial project plan with project supervisors.	• Review and amend the project plan accordingly. • Review the new project plan with project supervisors.	
• Effective project scheduling.	• Agreement on task prioritisation. • Commit more staff time. • Change in strategy: work may be divided into two groups in order to complete a key task. • Increase resources, give more time to ensure critical tasks are delivered on time.	
• Effective project scheduling. • Manage the critical path.	• Reschedule the project plan. • Redistribute the project resources.	
• Focus on capture of user requirements, at start of project. • Ensure user requirements are properly assessed. • The user requirements are fully investigated and agreed before specification. • Do not sign off on ambigous, ambitious and incorrect requirements.	• Immediately review the requirements with stakeholder. • Review the requirements with project supervisors and project sponsor.	
• Early prototype to identify reasonable expectations. • Use an iterative and incremental software development methodology.	• Escalate issue to project supervisors and project sponsor. • Reprioritise the tasks. • Reschedule the project plan.	
• Read resources and materials about tools, libraries and frameworks. • Guidance from technical experts.	• Support from technical experts.	
• Schedule a comprehensive testing plan. • Background research in testing methodologies. • Use automated testing tools. • Schedule testing plan for every iteration of the project cycle.	• Review the testing plan: schedule a comprehensive testing plan. • Read additional research materials on testing methodologies.	
• Use software configuration management tools. • Code convention.	• Use software configuration management tools. • Use high cohesion low coupling.	
• Use an iterative and incremental software development methodology. • Once a feature is fully functional, a prototype of the product is given out to the clients in order to evaluate it. • In the case that the clients are satisfied the feature is integrated into the existing system. • Otherwise, the feature is revised.	• Rebuild the system. • Improve some features of the system.	
• Use software configuration management tools. • Implement scheduled backups. • Code convention and consistency. • Assign milestones to both a primary and a secondary owner.	• Use software configuration management strategies. • In the case that a person is sick, redistribute project resources. • Communication with the team and supervisors in the case of an unexpected event. • Implement scheduled backups.	

Appendix G

JMeter Results

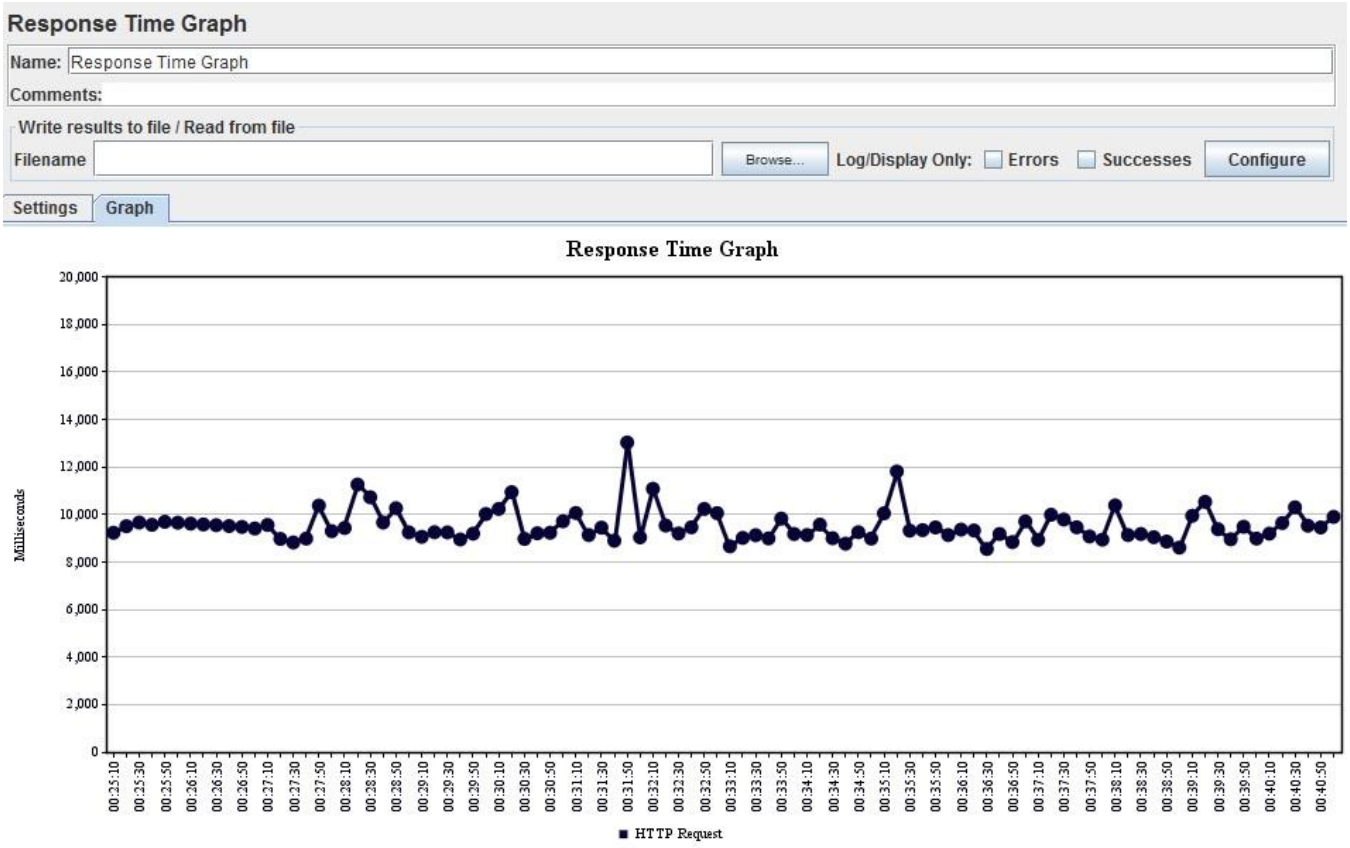


Figure G1 Response Time After 2000 Query Requests

Appendix G

JMeter Results (Continue)

Figure G2 Throughput after Sending 2000 Requests

Appendix G

JMeter Results (Continue)

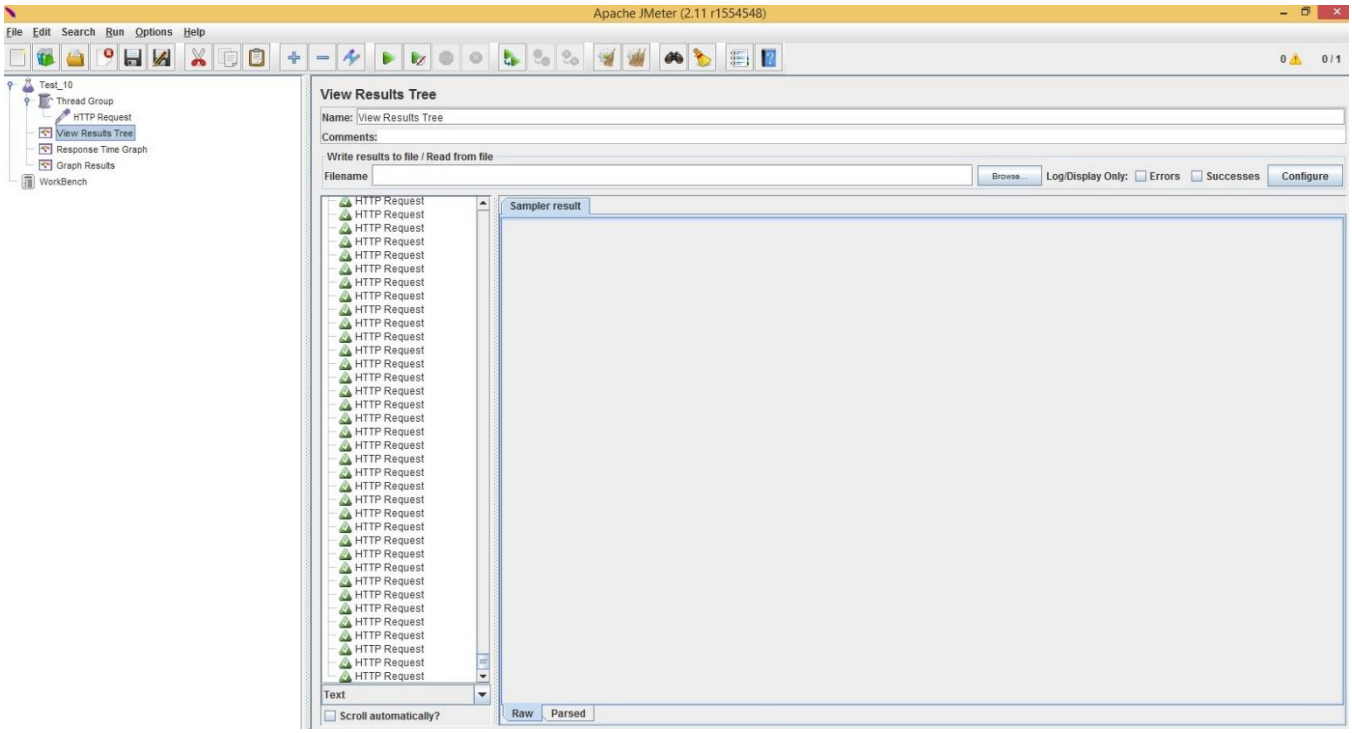
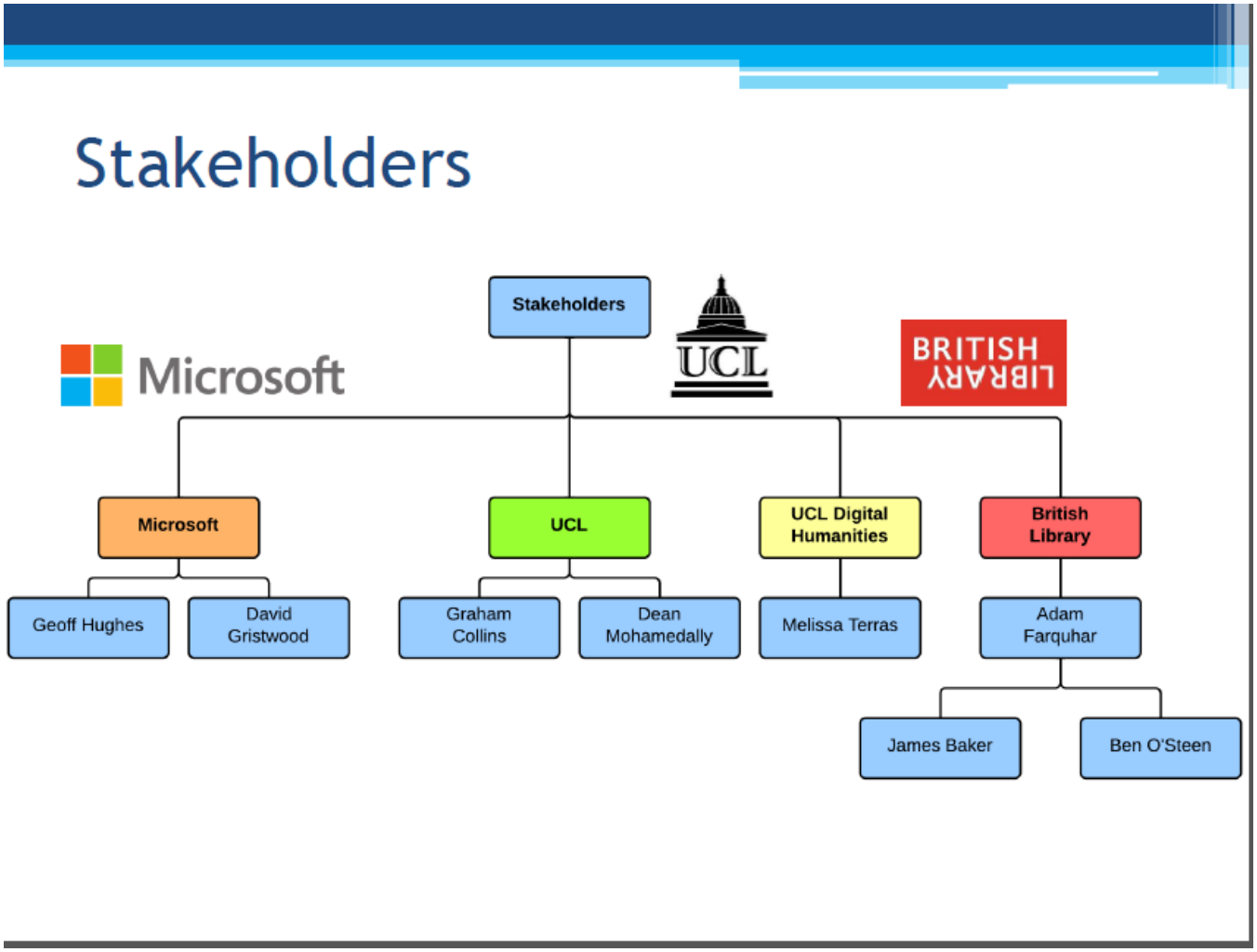


Figure G3 Simulation of 2000 HTTP Requests

Appendix H

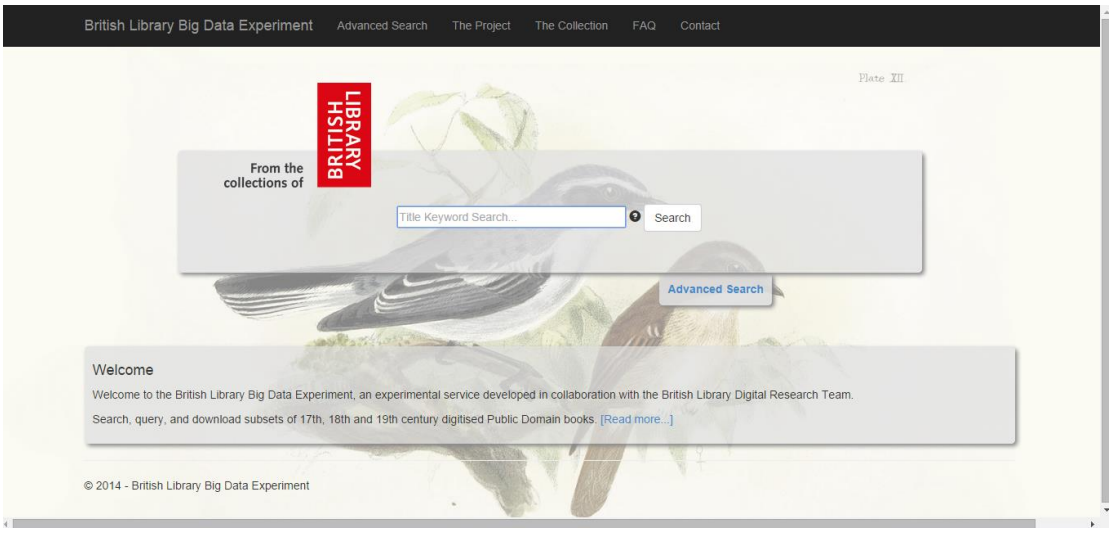
Stakeholders



Appendix I

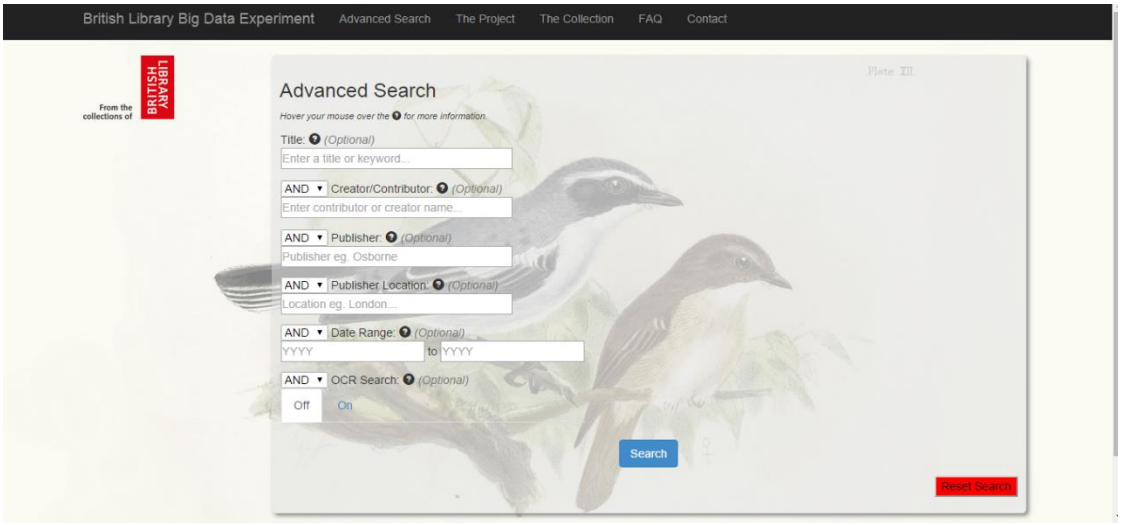
British Library Web Portal - User Manual

Home Page

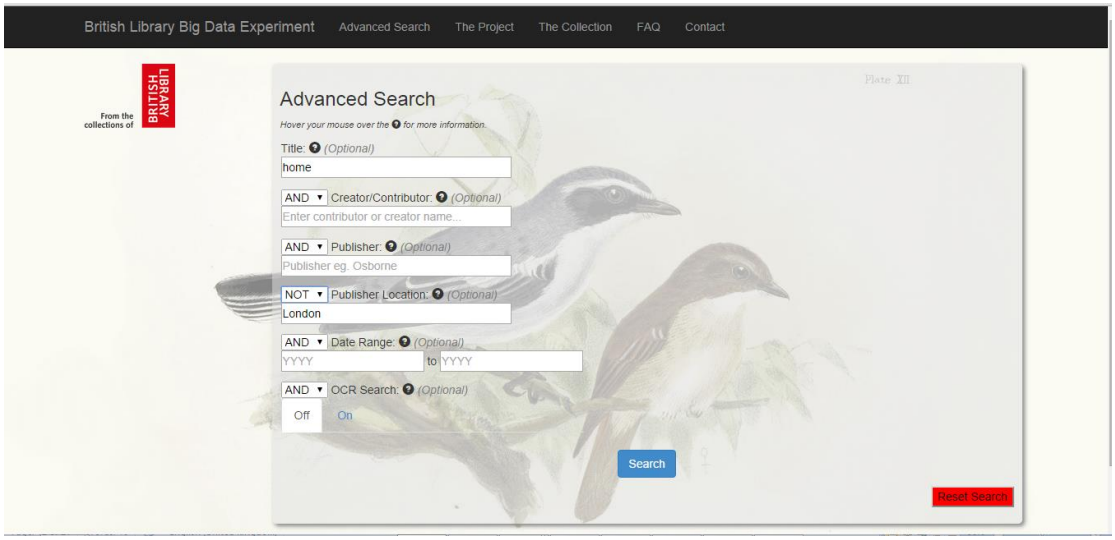


The user can access the Web Portal from the following link: http://blpublicdomain.azurewebsites.net/
At the Home page the user can perform a simple keyword search to search across all the fields apart from the OCR.
The user is able to search from a simple word or multiple keywords.
In the case that the user enters multiple keywords, results that match any of the input keywords are returned. In case a user enter a year, all the items about this specific year will be returned.

Advanced Search Page



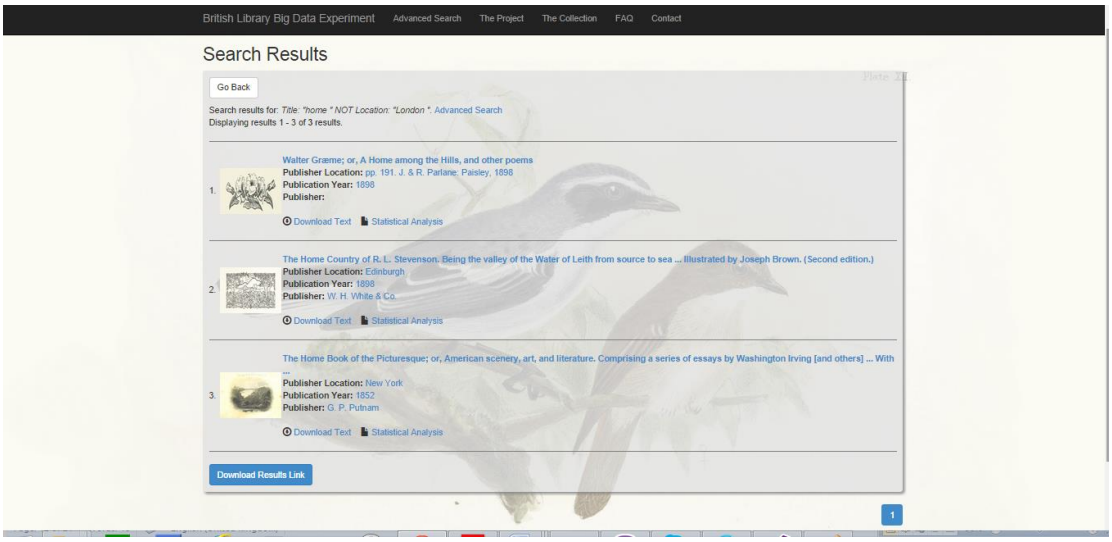
Access to the Advanced Search Page can be performed either directly using the URL http://blpublicdomain.azurewebsites.net/AdvSearch/Index or by clicking to the link "Advanced Search " at the Home Page. At the AdvancedSearch page a user can perform more complicated queries using the Boolean operators that can be applied on the fields.



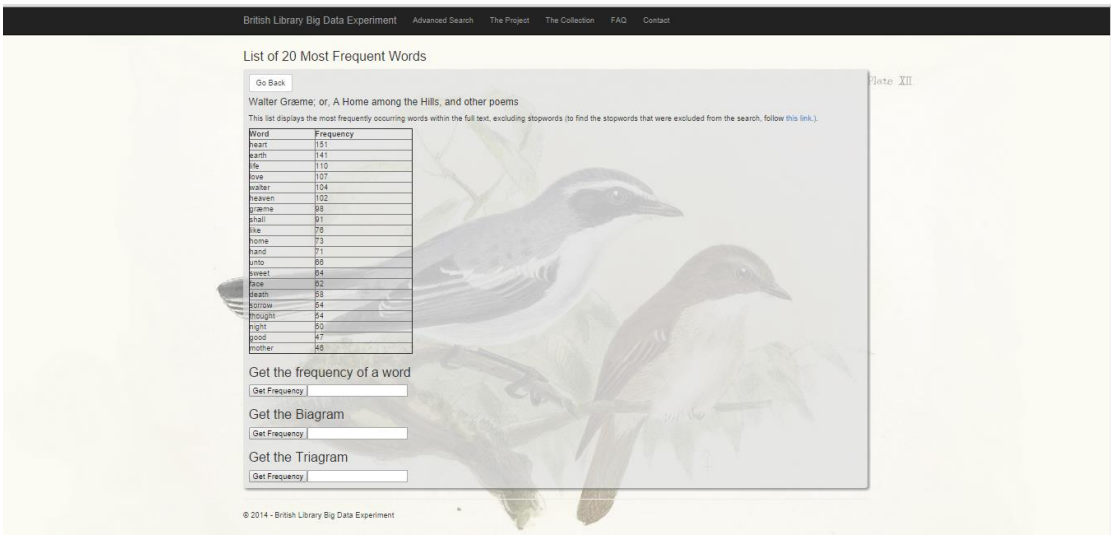
Example

The user inputs the keyword "home" at the field title, selects the boolean operator "not" and then enters the keyword London at the field Publisher Location.

Then retrieves the appropriate results that include the keyword "home" at the title and does not have as publisher location the keyword "London."



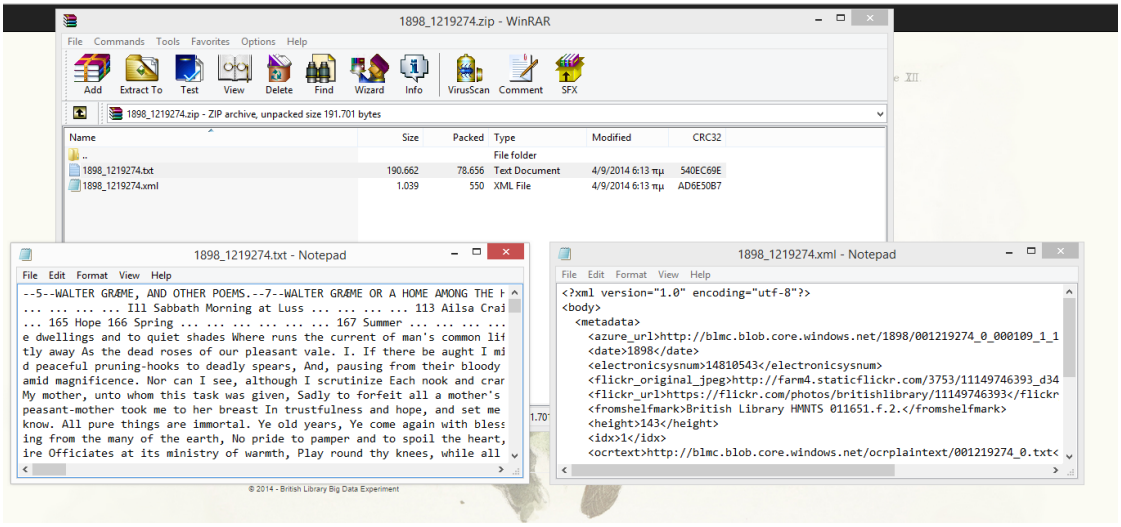
At the results page, when the user selects the Download Results Link, an internet shortcut is downloaded and the user is able to reproduce the queries.



In the case that user clicks at the link for the statistical analysis of an item the system retrieves the 20 most frequent words of the document and also the user has the ability to retrieve the frequency of a word, a bigram and a triagram.

In the case that the user clicks at the link “Download text” for an item at the Results Page, the OCR text and the corresponding xml are downloaded.
In order to open the zip file please install one of the following application

- WinRaR
- WinZip



Appendix J

Technology Stack

In order to deploy the system for future development the following are required:

- Microsoft Visual Studio 2013 Professional or Ultimate
- Microsoft Azure SDK
- Microsoft Azure Search Service

Frond – end

- ASP.NET Razor Markup
- HTML5
- Java Script
- JQuery

Testing

- Unit: NUnit
- JMeter
- SeleniumHQ

Source code Management

- TFS

References

[1] Buyya, Rajkumar, et al. "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility." Future Generation computer systems 25.6 (2009): 599-616.

[2] Armbrust, Michael, et al. "A view of cloud computing." Communications of the ACM 53.4 (2010): 50-58.

[3] Amazon. “What is Cloud Computing?” Web. 20 Aug 2014 http://aws.amazon.com/what-is-cloud-computing/.

[4] Alex Boardman. "Top 5 benefits of cloud computing "Microsoft Business, Web. 20 Aug 2014. http://www.microsoft.com/en-gb/business/news/top-5-benefits-of-cloud-computing-801588789.aspx/.

[5] British Library. “Quick Facts & Figures”. Web. 25 July 2014. https://pressandpolicy.bl.uk/content/default.aspx?NewsAreaID=20

[6] British Library. “Digitisation Strategy 2008-2011”. Web. 29 July 2014. http://www.bl.uk/aboutus/stratpolprog/digi/digitisation/digistrategy/.

[7] Malmsten, Martin. "Making a library catalogue part of the semantic web." Universitätsverlag Göttingen (2008): 146.

[8] Berners-Lee, Tim, James Hendler, and Ora Lassila. (2001). The Semantic Web. Scientific American, 284, 34-43.

[9] Berners-Lee, Tim. (2006). Linked data. Retrieved April 12, 2008, from http://www.w3.org/DesignIssues/LinkedData.html.

[10] Clark, James. (1999). XSL Transformations (XSLT). Retrieved April 13, 2008 from http://www.w3.org/TR/xslt.

[11] Buyya, Rajkumar, et al. "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility." Future Generation computer systems 25.6 (2009): 599-616.

[12] Luis Gravano, Hector Garcia-Molina, and Anthony Tomasic. The efectiveness of GIOSS for the text-database discovery problem. In Proceedings of the 1994 ACM SIGMOD Conference, May 1994.

[13] Luis Gravano and Hector Garcia-Molia. Generalizing GIOSS to vectr-space database and broker hierarchies. in Proceedings of VLDB '95, pages 78-89, September 1995.

[14] Steve B. Cousins, Scott W. Hassan, Andreas Paepcke, and Terry Winograd. A distributed interface for the digital library. Technical Report SIDL-WP-1996-0037, Stanford University, 1996.

[15] Steve B. Cousins. A task-oriented interface to a digital library.In CHI 96 Conference Companion, pages 103-104, 1996.

[16] Michelle Q Wang Baldonado and Steve B. Cousins. Addressing heterogeneity in the networked information environment. "Review of Information Networking", to appear.

[17] Ogle, Virginia E., and Michael Stonebraker. "Chabot: Retrieval from a relational database of images." Computer 28.9 (1995): 40-48.

[18] The POSTGRES Group, “The POSTGRES Reference Manual”, Computer Science Division, University of California, Berkeley, January 1993.

[19] Michael Stonebraker, et al., “The Implementation of POSTGRES,” IEEE Transactions on Knowledge and Data Engineering, March 1990.

[20] Microsoft, "Introduction to Microsoft Azure Storage", Web 18 Aug 2014. http://azure.microsoft.com/en-us/documentation/articles/storage-introduction/.

[21] Microsoft, "How to use table storage from .NET", Web 19 Aug 2014. http://azure.microsoft.com/en-us/documentation/articles/storage-dotnet-how-to-use-tables/.

[22] Microsoft, "Azure Table Storage and Windows Azure SQL Database - Compared and Contrasted", Web 19 Aug 2014. http://msdn.microsoft.com/en-us/library/jj553018.aspx.

[23] Microsoft, "Azure Storage Scalability and Performance Targets", Web 19 Aug 2014. http://msdn.microsoft.com/en-us/library/dn249410.aspx.

[24] Microsoft, "Understanding the Table Service Data Model", Web 19 Aug 2014. http://msdn.microsoft.com/en-us/library/azure/dd179338.aspx.

[25] Open Data Protocol, "Open Data Protocol Advanced Tutorial", Wen 20 Aug 2014. http://www.odata.org/getting-started/advanced-tutorial/.

[26] Axel Van Lamsweerde, Requirements Engineering from System Goals to UML Models to Software Specifications, 1st ed., Wiley, 2009.

[27] Zave, Pamela. "Classification of research efforts in requirements engineering."ACM Computing Surveys (CSUR) 29.4 (1997): 315-321.

[28] Nuseibeh, Bashar, and Steve Easterbrook. "Requirements engineering: a roadmap." Proceedings of the Conference on the Future of Software Engineering. ACM, 2000.

[29] Van Lamsweerde, Axel. "Requirements engineering in the year 00: A research perspective." Proceedings of the 22nd international conference on Software engineering. ACM, 2000.

[30] Rozanski, Nick, and Eóin Woods. Software systems architecture: working with stakeholders using viewpoints and perspectives. Addison-Wesley, 2011.

[31] Ian Sommerville, Software Engineering, 9th ed. , Addison-Wesley, 2011.

[32] Larman, Craig. Agile and iterative development: a manager's guide. Addison-Wesley Professional, 2004.

[33] Beck, Kent, et al. "Manifesto for agile software development." (2001).

[34] Letier, Emmanuel. Reasoning about agents in goal-oriented requirements engineering. Diss. PhD thesis, Université catholique de Louvain, 2001.

[35] Microsoft, "Azure Search Overviee (Preview)", Web 19 Aug 2014. http://msdn.microsoft.com/library/azure/dn798933.aspx

[36] MSDN, "Object.GetHashCode Method", Web 19 Aug 2014. http://msdn.microsoft.com/en-us/library/system.object.gethashcode(v=vs.110).aspx.

[37] MSDN, "ASP.NET MVC 4 Content Map", Web 25 Aug 2014. http://msdn.microsoft.com/en-us/library/gg416514(v=vs.108).aspx.

[38] MSDN, "Using the REST API Service (Messenger Connect)", Web 25 Aug 2014. http://msdn.microsoft.com/en-us/library/ff748607.aspx.

[39] MSDN, "A Guide to Designing and Building RESTful Web Services with WCF 3.5", Web 26 Aug 2014. http://msdn.microsoft.com/en-us/library/dd203052.aspx.

[40] MSDN, "Using the REST Services with .NET", Web 26 Aug 2014. http://msdn.microsoft.com/en-us/library/jj819168.aspx.

[41] MSDN, "Entity Data Model", Web 27 Aug 2014. http://msdn.microsoft.com/en-us/library/vstudio/ee382825(v=vs.100).aspx.

[42] Dijkstra, Edsger W. "The humble programmer." Communications of the ACM 15.10 (1972): 859-866.

[43] Pelrine, Joseph. "On understanding software agility–a social complexity point of view." E: CO 13 (2011): 26-37.

[45] Myers, Glenford J., Corey Sandler, and Tom Badgett. The art of software testing. John Wiley & Sons, 2011.

[46] MSDN, "Code Metric Values", Web 18 Aug 2014, http://msdn.microsoft.com/en-us/library/bb385914.aspx.

[47] Chidamber, A Metrics Suite for Object Oriented Design (IEEE Transactions on Software Engineering, Vol. 20, No. 6). Web Aug 30 2014, http://www.pitt.edu/~ckemerer/CK%20research%20papers/MetricForOOD_ChidamberKemerer94.pdf.

[48] MADN, "Building for Scale", Web 20 Aug 2014, http://msdn.microsoft.com/en-us/library/ff650487.aspx.

[49] Apache JMeter, Web 21 Aug 2014, http://jmeter.apache.org/