FundFind
Exploring scholarly funding opportunities

Emanuil Evgeniev Tolev
Degree scheme code: G401
Degree scheme title: Computer Science (Inc Integrated Industrial and Professional Training)
Degree type: BSc
Module code: CS39440
Module title: Major Project

April 24, 2013

Declaration of originality

In signing below, I confirm that:

- This submission is my own work, except where clearly indicated.
- I understand that there are severe penalties for plagiarism and other unfair practice, which can lead to loss of marks or even the withholding of a degree.
- I have read the sections on unfair practice in the Students' Examinations Handbook and the relevant sections of the current Student Handbook of the Department of Computer Science.
- I understand and agree to abide by the University's regulations governing these issues.

Signature .....

Date .....

Consent to share this work

In signing below, I hereby agree to this dissertation being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Signature .....

Date .....

Acknowledgements

I would like to thank my supervisor, Professor Reyer Zwiggelaar for his support and unique approach to the process of producing a major project.

Thanks to Dr. Richard Jensen, Dr. Joanne Walker and Professor Simon Cox for their valuable input in the formative stages of the project.

Furthermore, thanks to all Computer Science Department staff who have taught me over the last 4 years for their patience and being such an inspiration with their varied interests and opinions.

I would like to thank Cottage Labs LLP for introducing me to the Open Knowledge movement, as well as teaching me about most of the major technologies used in this project and agreeing to host a FundFind instance.

Thanks to the Open Knowledge Foundation for striking out and encouraging prototype projects, providing ample inspiration for FundFind.

I would like to thank Research Councils UK, a partnership of the seven UK Research Councils, for all the effort they have put into the Gateway to Research API, consolidating data from seven separate institutions and making it available under an Open license.

Abstract

FundFind is a web application which enables scholars, research development officers and postgraduates to share information about funding opportunities. Data about funding opportunities and funding organisations can be submitted, browsed and searched via a mobile-friendly web UI as well as a JSON API. FundFind communicates with its data store via simple asynchronous HTTP. This enabled the development of a secondary Funding Harvest application which harvests funding data automatically and is completely separate from FundFind - it simply puts data into the same datastore.

The project is a learning exercise, aiming to explore the field of scholarly funding, identifying users and their needs, as well as potential problems with the open sharing of such data. These insights and the technical outputs are meant to be developed further (by an interested party) into a funding information discovery service after this project is completed. The main application was based on another prototype service developed by the same party.

The project did allow for a lot of insight into the funding sector and users of funding information. However, the parent project had no automated testing and difficulties with integrating multiple third-party pieces of software hampered technical development. The incomplete technical prototypes impeded the depth of the insights gained as feasible user testing was limited. The main application is, however, of similar quality to its parent prototype. The lessons learned are sufficient and valuable for the future development of a FundFind service.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Background and Objectives

1.1 The Project

This project is about learning to deal with the field of funding in modern scholarship[1] and exploring how it could be opened up, from the point of view of a software developer. It is set within the context of the Open Knowledge field and seeks to add to this field by exploring how scholarly funding can be made more transparent and easier to find for all interested parties.

The main vehicle of learning and main software output is an open-source web application named "FundFind". It's a simple crowdsourcing application which allows registered users to share information about funding opportunities under open[2] terms. Its source code is available at [22]. Access to submitted information includes a machine-friendly API (Application Programmable Interface).

One of the project features - getting funding data into the datastore - was developed as a separate technical project due to natural low coupling between the main application and this feature. It is called "Funding Harvest". Its source code repository is publicly available at [23].

1.2 Project Aims

- Learn more about the funding sector and identify user requirements within it. Results presented in "Audience and High-Level Requirements", §1.4 and more concretely in "Requirements", §2.4. The data must be sufficient to enable the creation of a project roadmap for

  a) future research work into the requirements of user groups which were not engaged during this project

  b) concrete development work which would enhance the software built during this project

  The foundations for this are laid in in "Future Work", §6.5.

- Identify how much software it is possible to build by essentially starting and seeing how far it goes in a controlled fashion. The author's own initial lack of domain knowledge about field and technical skill level mean than a fixed feature list is hardly possible. However the progression of features and the final "shape" of the FundFind software is discussed in "Requirements", §2.4.

[1] A note on "scholarship" - the word and its derivatives are used throughout this text to mean all types of academic work, due to "science" being sometimes perceived as exclusionary to the Arts and Humanities fields.

[2] The Open Definition defines "open" well [46].

1.3 Scholarship and Funding

Scholars need money to conduct research. To be precise, research projects need *additional* money (on top of the investigators' salaries) in order to cover material expenses, travel costs, hiring Research Assistants or providing studentships to PhD candidates and other expenses.

In other words, established scholars need to look for funding for aims which may be hard to define, for amounts of money ranging from thousands to millions.

Postgraduate-level scholars (candidates, postgraduates, post-doctoral researchers) are also usually looking for a way to fund the next stage of their scholarly career. (Note: undergraduate-level funding was not considered in this project.)

The quintessential problem is that the information about available funding is published in many different places. Usually each funding organisation publishes its own calls for proposals and announces its own studentships on its own website(s) and mailing list(s).

Keeping up with these sources of information becomes quite difficult - so much so, that it takes a significant portion of research development officers' time. Worse still, this is a significant barrier for those looking into a career in academia - the fragmented information can make it difficult to build a mental picture of what the funding in one's chosen field "looks like". This leads to a feeling of opaqueness instead of transparency and being overwhelmed - instead of having a clear idea of what it takes to succeed in a chosen field. Sometimes it even leads to accusations of cronyism in distributing funding (e.g. [43]).

Furthermore, if this process is that difficult for person working in a specific country, then it can easily be estimated that globalisation of scholarship and scholars' mobility is hindered as well.

These problems seem to stem from the fact that the distribution of scholarly funding seems to have evolved over time from a series of changes to more general money distribution in society. This is actually a well-known problem that the software development field has had to deal with - if multiple stakeholders with distinctive interests try to influence the development of a software system, changing its requirements over time, the results can be disastrous.

There is no easy way to track money - where it gets allocated to, what fields (or even topics) within scholarship get funded more than others. Up until November 2012, there was no publicly accessible *centralised* detailed account of historic funding information in the United Kingdom. The Gateway to Research project being implemented by Research Councils UK finally gives a partial (RCUK-specific) overview of historical data.

However, there is still no mechanism of getting *federated, present, current* data about scholarly funding.

Something which allows searching the (vastly) different, *currently available* streams of funding within academia was thus identified as potentially useful by various people interested in the subject.

1.3.1 The Value of Openness in Scholarship and Elsewhere

The Free Software, Open Source and more recently, Open Knowledge (which Open Access is a part of) movements have all demonstrated that the value of information can be greatly multiplied simply by having more people access it, reuse it and even be creative with it (e.g. visualise or summarise). The Open Knowledge Foundation argues this [49]. Seminal works like "The Cathedral and the Bazaar" [30] have also argued this.

Governments seem to have grasped the benefits as well. The UK Government has mandated that all research funded with public money (which is a lot in the UK - almost everything through the seven Research Councils) should be published as Open Access by 2014 [35]. It has also funded

initiatives such as Open UK governmental data in the form of data.gov.uk [70] and the recent Finch report [1] [76].

The "Open" part of "Open Access" means more than just throwing the data out there in any form and format. The Open Definition [46] has a lot to say about all aspects of Openness. However, principles #1 "Access" and #4 "Absence of Technological Restriction" are relevant, so it can be seen why the current practice of each funder publishing their own HTML pages is not good enough:

- "The work shall be available as a whole [...] "As a whole" prevents the limitation of access by indirect means, for example by only allowing access to a few items of a database at a time."
- "The work shall be available [...] in a convenient and modifiable form. [...] The work must be provided in such a form that there are no technological obstacles [...]. This can be achieved by the provision of the work in an open data format."

In practice, when applied to scholarly output such as publications, these have been observed by the author to mean "PDF-s are not going to cut it", "You must be able to data mine it".

So why not have this for funding data? Currently, separate HTML pages, weekly e-mails and (sometimes) RSS feeds are all that the Research Councils UK do to publish their funding information. RCUK, in particular, are a publicly funded organisation, and it could be argued that this data belongs to the public. In any case, convincing both public and private funders of the benefits of Open funding data is needed.

Private organisations are not far behind and may in fact be leading the way - one of the largest funders of science in the UK and around the world, the Wellcome Trust, has had a progressive and strict Open Access policy since 2006 [15].

1.4 Audience and High-Level Requirements

The potential audience of this project was discovered to be quite varied. It seems multiple stakeholders stand to benefit from more transparent scholarly funding dissemination.

The requirements or potential uses that these users might want to put FundFind to are closely tied to the benefits that come with opening up funding data.

1.4.1 Professionals Looking for Funding Group

1.4.1.1 Accomplished Scholars

- **Who**: Academics - lecturers, professors
- **Aims**: Wish to fund future projects
- **Benefits from opening up funding data**: Easier access and search across funding institutions through tools which aggregate funding data; Transparency and accountability of their funding bodies.
- **Possible requirements from FundFind**: Aforementioned cross-funder search; Sharing opportunities with colleagues.

1.4.1.2 Junior Academics

- **Who**: Postgraduate candidates, postgraduate students, academics in post-doctoral posts (e.g. Research Associates)
- **Aims**: Wish to find suitable positions to move on with their academic career, e.g. a postgraduate with a PhD will be looking for a post-doctoral position
- **Benefits from opening up funding data**: (More) easily find and compare funding sources
- **Possible requirements from FundFind**: Find funding. To quote a postgraduate candidate: "I want to do a taught MSc in Astrophysics but I can't afford to self fund. Where can I go and who will pay me?"

  Share funding information with peers (e.g. if a candidate for a Computer Vision PhD finds a good PhD studentship call for applications in Bioinformatics, they may well want to forward it to a colleague).

1.4.1.3 Research Development Officers

- **Who**: Professionals who find more funding opportunities for scholars
- **Aims**: Wish to record the opportunities they find somewhere and share them with scholars. Also may want to access a list of all opportunities *they* have found over time, so that they can report to their line managers more easily.
- **Benefits from opening up funding data**: Not as many as scholars, unless they are happy to share the opportunities they have found with research officers from other institutions (which they do not always want). However, a system which allows sharing of recorded opportunities with a select subset of users (such as all within the officer's institution) would be welcome as this can allow them to record a funding opportunity once, share it with "their" scholars and include it in reports.
- **Possible requirements from FundFind**: Find funding. Share funding with subset of users. See information submitted per user (themselves).

1.4.2 Funders Group

Those looking to advertise funding opportunities - Funding Stream and Programme managers at various institutions.

The needs of this group are estimated since funders were not engaged as users at this point in FundFind's development due to time constraints.

1.4.2.1 Research Councils in the UK

- **Who**: Main way of allocating *public* scholarly funding in the United Kingdom.
- **Aims**: Wish to attract the best scholars in their own field, but also look to establish new cross-disciplinary projects.
- **Benefits from opening up funding data**: Already committed to opening up historical funding data (e.g. Gateway to Research project [57]), would like to advertise current funding data more widely. The benefits are widening impact and increasing the inter- and intra-field

interconnectedness of scholarship (scholars can learn about interesting funded projects and people that they didn't know about).

- **Possible requirements from FundFind**: Mostly just search and upload of information. In particular: search for submitted items related to their institution. Bulk upload of opportunities. Control over the "funder" profile related to them.

1.4.2.2 Jisc

- **Who**: Used to be known as "Joint Information Systems Committee", but have decided to rebrand [36]. The old name of this public organisation reflects their purpose quite well - helping the adoption of digital technology in research and teaching in the UK.
- **Aims**: Wish to attract talent to work on scholarly infrastructure in the UK, i.e. give their aims and projects more exposure. Scholars rarely choose scholarship infrastructure as their main field.
- **Benefits from opening up funding data**: Have always wanted wider reach, as well as vigorously supported Open Data. Have a lot of funding opportunities, both minor and larger ones. Would like to advertise them better.
- **Possible requirements from FundFind**: The same as RCUK. In addition, however, they might wish to actually use FundFind's data in other projects - in other words, they might be an interested API consumer.

1.4.2.3 Charities, Trusts and Others

- **Who**: Various charitable Foundations, Institutes, Endowments and so on which fund scholarly activities (e.g. the Wellcome Trust [87], Shuttleworth Foundation [65], Knight Foundation [38]).
- **Aims**: Simple - want the best scholars competing for their money (at least the portion dedicated to conventional research).
- **Benefits from opening up funding data**: Need to reach more scholars to build up an ever-improving pool of applicants (in terms of quality). Also, as private institutions, would like to demonstrate greater transparency with their funding dissemination and allocation as part of their efforts to prove they are doing it effectively.
- **Possible requirements from FundFind**: Same as RCUK.

1.4.2.4 Commercial Organisations

- **Who**: All companies which perform R&D activities. Famous UK examples include Rolls-Royce [61] and IBM [32].
- **Aims**: The ones with large enough R&D budgets to get involved with scholars usually invest in whole laboratories and centres. When they do share effort and money with research institutions, they do have to search for the "best scholars" to collaborate with. The smaller ones may have a problem however - they may only need external help with R&D intermittently and thus may not have a "constant" audience of scholars following their every move since they would not be allocating funding all the time.

- **Benefits from opening up funding data**: Greater transparency of funding allocation, potentially greater audience if a federated scholarship funding source like a popular FundFind instance does come into existence at some point.
- **Possible requirements from FundFind**: Same as RCUK.

1.4.3 Analyst Group

Those looking to analyse scholarly funding - how it is allocated and/or spent.

A hackday in the middle of March which focused on the Gateway to Research project provided insight into the usefulness of certain initially planned features and inspired new ideas. The main audience of the hackday was this group. The GtR is an attempt to get the seven UK Research councils to publish historical funding data - who got funded, grant codes, who worked with whom, any related publications [58], etcetera via a web interface and a RESTful API. GtR is quite similar to FundFind, except that it relates to *historical*, not current funding information, and of course contains a lot more information than FundFind is currently scoped to hold.

- **Who**: Software developers, journalists and others interested in Open Knowledge and visualising data for the purposes of transparency or advancing the digital economy [49] The author is included in this group. Other interested parties are analysts, bloggers and the general public.
- **Aims**: Want transparency and accountability. May want to visualise how scholarship money is being spent or otherwise help society engage with scholarly spending.
- **Benefits from opening up funding data**: Open Data essential for re-use for their aims. Usually do not have first hand information about what happens behind closed doors when funding is distributed and used by universities. Think that this is strange, since often both funder and fundee are public sector organisations / employees.
- **Possible requirements from FundFind**:

There is another group which belongs here - politicians. They may wish to demonstrate accountability and good decision making. Despite the fact that they are not usually known to readily embrace new technological solutions, they do do it when the advantages (at least from a Public Relations perspective, if no other) become clear. For example, the UK government more or less suddenly embraced Open Access for all publicly funded research [35].

1.4.4 Focusing on Certain Audience Groups

Satisfying the requirements of all user groups is more suitable for a larger, better-resourced project than this one.

This project will focus on the first group of users" meaning "Professionals Looking for Funding Group", §1.4.1. The main reasons are still "time constraints and ease of access to such users". However, the "Analyst Group", §1.4.3 was also taken into account. This was partly because the success of a technological project just has to include other developers, especially when it is an open-source project. On the other hand, the author is part of this group and day-to-day interactions create unavoidable bias, especially when they include learning (how to use technologies relevant to FundFind) and feedback on the very project being developed.

As stated previously in the Progress Report, FundFind does have an open machine-ready to communicate with other software - an Application Programmable Interface (API). In terms of concrete people within this group, Cottage Labs LLP [8] still find value in the project and will probably help to take it further after the current development as a piece of course work ends.

In addition to the The Open Knowledge Foundation [49], to whom the project will be presented after it is finished and feedback will be asked for, the Gateway to Research technical team will also be contacted about possibly building a stronger relationship with FundFind. They might be interested in its use of the data provided by their API to make suggestions of interesting projects (to look at) to scholars. Furthermore, they handle *historical* funding data, whereas FundFind handles *current* funding data. Convincing the Research Councils UK to provide the same uniform interface to their *current* funding data as they are to their *historical* data with the GtR project would be a great boon to the Open Knowledge movement.

1.5 Existing Works

There is no single piece or combination of pieces of software which enables the identified audience to do what this project would allow them to do upon completion, to the best of the author's knowledge.

Related Open Knowledge projects are still as relevant as at the beginning of the project, in particular the Open Knowledge Foundation ones at [48] and [47]. "Design Details and Rationale", §3.1.1 notes individual projects which informed FundFind's technical design in some way.

The FundFind codebase was initially based entirely on the IDFind project [26], which is available under the MIT license [27]. Cottage Labs LLP and the author started it during the DevXS 2011 conference [66]. IDFind was a good place to start, since the project's context (Open Knowledge, interested parties) is similar, so a similar set up and technologies could be used to accelerate the initial development.

1.6 Licensing

FundFind is licensed under the MIT license, like its parent IDFind. This is a permissive license which should facilitate the widest possible use and reuse of code. Moreover, there is little reason to place particular restrictions since the target Open Knowledge / Funding fields are narrow enough and there isn't really any open-source software which sits at the intersection of the two.

1.6.1 Contributions

Contributions through FundFind's user interface (crowdsourcing) are licensed under the Creative Commons Attribution (CC-BY) [16] license, version 3.0 (latest at time of writing). This was chosen after considering a multitude of other licenses. In the end, the new Budapest Open Access Initiative recommendations and the Open Definition [46] recommending CC-BY informed the choice.

This license only applies to publicly visible (shared with the world) submissions. For now, it is not possible to share funding information only with certain users or groups - therefore, the license applies to all crowdsourced contributions. Proper access control was too costly to implement in light of more important / core features which would enable the system to at least act as a prototype for completely open sharing, before moving on to the more complex role of a virtual shelf of funding opportunities.

The contributions license does not apply to data which has been harvested from data sources (like funders' websites or RSS feeds) automatically. The owners of the data would have to give written consent for Funding Harvest to harvest their data and publish it as CC-BY (or, better and more likely - publish it as CC-BY themselves). Funding Harvest is only a tool which, in combination with FundFind, is meant to showcase the benefits of having funding data open, thus convincing data sources to publish data more openly. It is not a tool to infringe on the copyrights of others, although it can be used as such. This is true of any software which does page scraping or even just consumes and records RSS feeds.

Most records in FundFind's datastore denote the license they can be used under. If a record does not have a `license` field (user accounts do not, for example), then the default (restrictive) copyright rules apply to that data.

Chapter 2

Development Process

2.1 Introduction

The project used an evolutionary prototyping approach. There were two phases:

1. requirements gathering; building the basic application structure, in particular submission of information; investigating appropriate technologies
2. building the features from the initial feature list feature-by-feature; some were dropped due to difficulties with them or higher priority features - final list arrived at through this iterative process

An Agile, iterative approach to the development of the software was initially chosen. It turned out that the approach did not fit the project / author / users context. In particular:

1. The initial approach was actually not Agile to begin with (a problem of understanding what Agile actually entails).
2. Agile requires a certain level of understanding of the technology (developer's side) and problem the software is solving ("customers"' or users' side).

The length of the requirements gathering phase and the complexity of developing the most important features was initially underestimated. Analysis of the intermediate outputs and measurement against the initial aims suggested that a different scope was necessary. The new goals required a different approach, but that alone would not provide an answer as to what FundFind should have aimed for while remaining realistic.

Concentrating on and discussing the original goal of the FundFind project with interested parties proved fruitful. The overarching aims of "learning about scholarly funding" and "prototyping a solution for sharing this information" resulted in a leaner, more focused set of requirements to work towards. Since work on the code had started and a basic web application was running, it was most appropriate to enhance this and peruse the chosen datastore's unique characteristics to add new features while retaining simplicity. This approach in the second phase of the project was essentially faster evolutionary prototyping (compared to the first phase), mostly due to building better understanding of the technologies involved and insufficient time to deliver all features on the list.

2.1.1 Project Management Tools

A project management piece of software suited for Agile approaches was chosen to manage this project and was used successfully throughout: PivotalTracker. The project's progress and a list of features (in priority order) has been publicly available at [68] throughout the project's current lifespan (October 2012 - April 2013).

A private source code version control repository was set up with GitHub [33]. It is only private due to several concerns:

1. The licensing status of the IDFind project which FundFind was forked off, was unclear. In fact, with no license statement anywhere in the repository, the default copyright rules apply and FundFind could not have been legally based on IDFind without explicit written permission from the authors. However, since IDFind's only contributors were Cottage Labs LLP and the author of FundFind, this did not delay FundFind's development. Nevertheless, IDFind's licensing status was not cleared up until April 2013.

2. There were concerns about the quality of FundFind's code and the repository. GitHub is home to enough orphan repositories which are all almost empty or without proper licensing, either of which makes the project useless in terms of its Open Source aspect.

3. It wasn't quite clear whether the code could be released during the official timeline of the final-year Major Project module at the author's institution. One of the main ideas in open-sourcing code is that others can contribute to it, but that would have significantly complicated matters in terms of reviewing the final version of the code. Cottage Labs LLP would like to contribute, but agreed that they are busy enough at the moment to look at the codebase after the 26th April 2013 (by which time a copy of FundFind's codebase would have been handed in).

Furthermore, a private repository holding all of the other outputs of the author's Major Project (Outline Specification, Progress Report, presentation files, this document and so on) was set up purely for back-up purposes.

2.2 Evolution of the Development Process

Initially, Agile practices such as Pair Programming [64], full Scrum [63] and Feature-Driven Development were deemed unsuitable since they are aimed at groups of developers. Behaviour-Driven Development (with Cucumber [4]) was considered, but requires more commitment from the users - they need to learn an English-like domain-specific language which describes how the system is going to behave which didn't sit well with assumption #2 above and turned out to be the right decision as that assumption proved correct.

Thus, initially, with no methodology fitting the precise needs of the project and the Agile Manifesto [5] in mind, a methodology of an Agile *character* was planned, using the Extreme Programming lifecycle as a base [18]. The idea was to use ideas such as Release Planning (plan a set of features), Iteration (develop in small increments) and Stories (described above).

It turned out Agile needed a consistent, if not constant, supply of time however. There was little development time availability in certain weeks, but far more in others. The size of different features differed as well, with a lot of the work going into the initial features and lower-priority features actually being completed relatively easily. Thus, "evolutionary prototyping" is a much more suitable term to describe the actual approach to development over time.

The basic steps which were defined in the initial Agile methodology were:

“

1. Meet with a variety of potential users from the chosen user group (“Focusing on Certain Audience Groups”, §1.4.4), trying to pick them so that each one has a different professional perspective on scholarship.

   Emanuil Tolev in FundFind Progress Report [25]

”

This worked well, with candidate postgraduates, postgraduates, research assistants (post-doctoral), lecturers and developers all agreeing to give a bit of their time. The meetings were not as regular as initially hoped (1 per month with each user) due to users' commitments, author's commitments and a particular characteristic of this project.

Looking at requirements presented in “Audience and High-Level Requirements”, §1.4, it is easy to see that the basic searching and sharing of opportunities were two *really* important functions. Sharing is used to different ends by research officers, lecturers, postgraduates and funders, while searching is useful for different reasons to pretty much everybody. However, getting timely feedback from all target users was not considered to be possible - and that is required for iterative development. It was actually considered (and turned out to be - “The Basics”, §4.3.1) difficult enough to produce a prototype which made progress in satisfying the initial requirements of *multiple* groups. Thus, meetings essentially happened at the start at the project (October, November, December) and at the end, in March and April, since most of the time in-between was spent getting those two important features right. Advanced functionality was planned in March and developed in April and is described below in “Final Requirements”, §2.4.4.

“

2. Define *and prioritise* the functionality of the project with each user in the form of the “stories” mentioned above. Record the results using the chosen project management tools (“Project Management Tools”, §2.1.1).

   Emanuil Tolev in FundFind Progress Report [25]

”

This turned out to be a really good idea as it was easy to re-prioritise features during meetings. However, different users had different priorities - some did not see much use for certain features while others preferred them (the differences were starker between user groups, as can be expected). Reconciling those differences into one final feature list was extremely difficult - in essence, it was not possible to please everybody.

“

3. Release planning - prepare an initial timeline of which project features are to be released when and denote “release points”. This work can be repeated as required - the order and the features themselves might change on the basis of user feedback over time (this is how Agile projects try to deliver more value by responding to changing requirements). The total size of the work and the size of each feature will most probably remain the same.

   Emanuil Tolev in FundFind Progress Report [25]

”

The total size of the work for each release and the size of each feature did not remain the same. A great amount of time was spent getting the underlying code and the first three features to work, whereas the final month of the project saw the completion of everything else. This is quite a typical situation in software development and the reasons for making the assumption above are unclear. It should simply not be made in software development.

> 4. Iterate - every week is a self-contained unit of work consisting of coding, documentation and write-up work. Before starting to code each week, break up that week's story into tasks so goals can be tracked more effectively and there is something to be accomplished each day.
>
> *Emanuil Tolev in FundFind Progress Report [25]*

The time allocated to the project each week varied (besides the variable size of the work itself noted above), so the timeboxing did not work out as well. On the other hand, breaking up more complex features into tasks was probably responsible for having as many as 10 items in the final feature list.

> 5. Repeat items 1 and 2 - meaning regular meetings with previously chosen users - one per user per month, demonstrate the latest features and record the feedback.
>
> *Emanuil Tolev in FundFind Progress Report [25]*

The problem here was again a different amount of time committed to the project each week. Usually it is possible to achieve consistency in feature delivery despite this, since some features will just take less time. This has been observed by the author in other projects. However, the observations held for the later stages of projects which had a developed basic framework (and had multiple, although few, developers) - this just did not hold for a prototype which had to be brought up from the ground single-handedly.

2.3 Modifications

In the Progress Report features were planned and *scheduled* in a timeline before development took place, but that is *not* the Agile way at all.

2.3.1 What is Agile?

Agile is about getting work done in an iterative fashion so that there is always working software for the users to use. Estimates in Agile are about effort, complexity and risk. Estimates are also *relative* - when new features are added to the project scope, they are estimated in comparison to previously added ones. If numeric points are used for the estimation, the project ends up with several hundred / thousand points worth of features.

Since those are prioritised by the customer/users, work can then start. What Agile teams seem to do is to essentially see how quickly they get through work - how many points they go through per week, which is called the "velocity" of the team.

Only then is it possible to estimate how long all of the initially requested features will take to develop - by taking into account their "size" (relative to each other) and the team's abilities. Taking the total project size and dividing it by the weekly velocity gives a rough idea of how many weeks it would take to complete all requested features. Let this be `time_required`.

If `time_available < time_required`, then the scope of the project simply has to be limited, i.e. features have to be dropped, and this only becomes clear once velocity is established (so at least a few weeks after development has started). In other words, however many features there *turns out* to be time for in the development process are *what gets done*, which is why there is so much emphasis on having the users involved - prioritising the functionality themselves.

2.3.2 Is it Agile?

FundFind's Progress Report [25] stated that an Agile-like methodology had been chosen - the main concept being user stories which allow capturing user requirements very quickly in an informal way.

Having a list of desirable features is great, having a schedule of which one will be delivered when - not so much. The point of Agile is taking the specifics of the development team and users' knowledge (and specificity of their desires) into account. A fully pre-defined plan - even one informed by meeting users, such as in this case - disregards all of this and does not conform to Agile's flexibility requirements. (Responding to change better than more conventional approaches is usually one of the first touted benefits of Agile approaches.)

Figure 2.1 presents that initial plan.

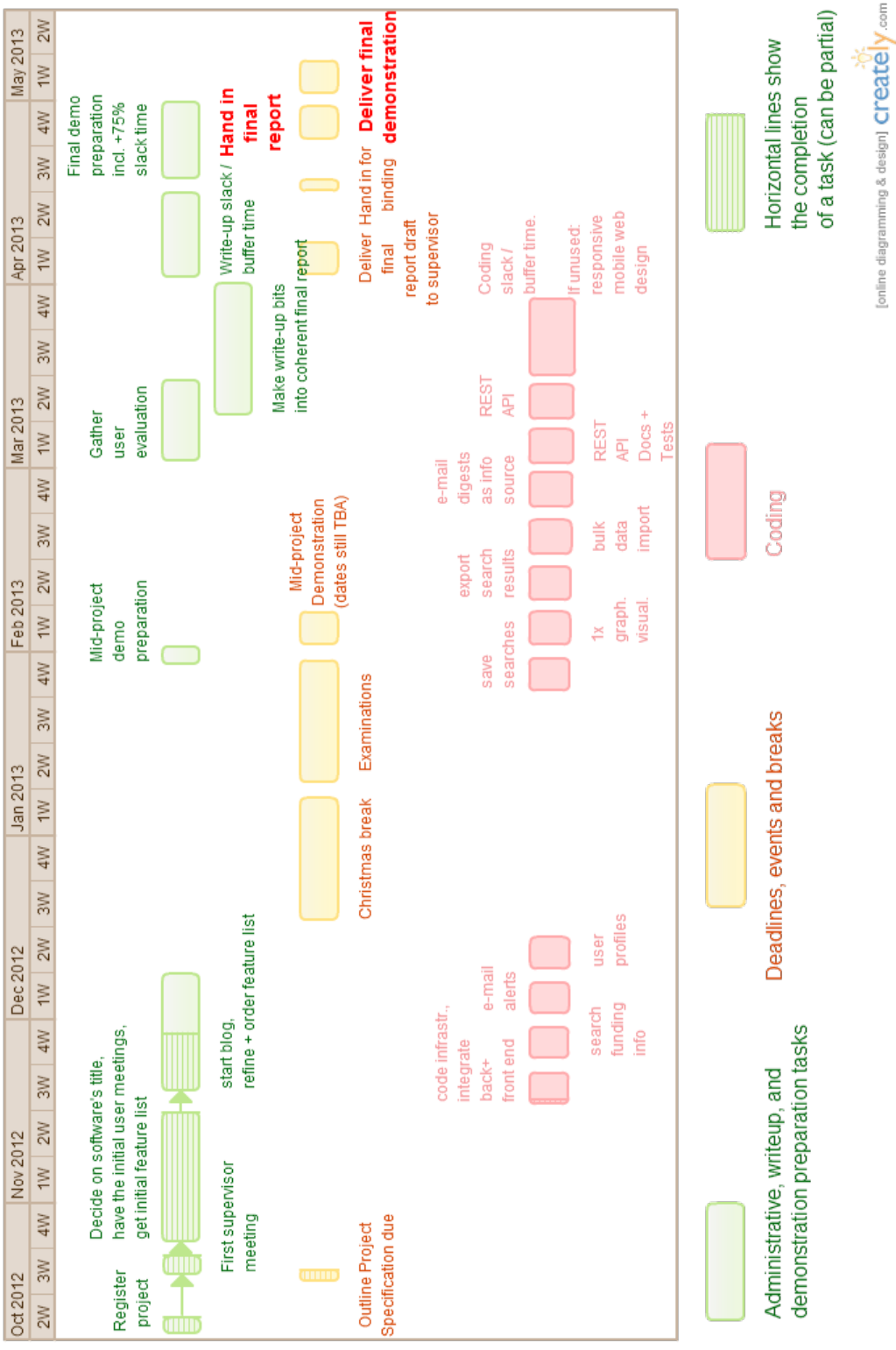

Figure 2.1: FundFind initial timeline

2.3.2.1 Underestimation

The main reasons for choosing Agile were

1. Agile's perceived advantage in pinning down changing requirements.
2. Less formal communication with the users, who are all busy professionals. It was assumed they would not have sufficient time for writing functional specifications, even in collaboration with the author (as can sometimes be assumed in industry).
3. Linked to the user communication reason was a concept which many Agile methodologies used - user stories. These are short sentences (or rarely, paragraphs) which describe a single piece of functionality - a single "thing" that the user wants to do with the system.

However, it turned out that both the complexity of the scholarly funding field and the author's ability to concretise fuzzy requirements, as well as their technical skill had been underestimated (elaborated upon in Chapter 6). The fact that requirements had to be gathered and concretised is not unusual in Agile, but playing both developer and user was a problem.

Neither did users have enough time to be properly involved throughout the project, nor did the author have enough domain knowledge to understand the requirements and develop quickly enough *in order to get the users involved* in the way in which Agile understands involvement.

This conclusion and the process of arriving at it, ultimately, are the reason why the initial Agile-like development process was deemed unsuitable and why the project's progress is split so starkly into two phases.

The initial reasons for devising an Agile-like process (above) proved to (mostly) be steps in the right direction, however:

1. The flexible requirements reason did not work out so well since it turned out the requirements did not change or need to change. Issues with funding data have been present for a while and a system which tries to prototype fixes through Open Data for some of them needs (a) developer(s) who know these issues inside out. of course, the requirements gathering can still be done in an Agile way, but needs to be mostly completed before development starts. Defining the problem, in this case, had to come before solving it.

   It could be argued that this was only a problem since the developer was also the "initiator" of the project (i.e. had to play user a lot more than usual). If a particular funder, Higher Education Institution or an organisation like the Open Knowledge Foundation wanted a project like this done, they would already know of certain problem characteristics - such as *particular* drawbacks of having so much funding data "closed". Then, more targeted exploration could be done and technical development would also have more precise targets to meet.

   This is, in part, what caused the development process to look a lot more like **evolutionary prototyping** with a lot of requirements gathering between the prototype versions rather than an Agile iterative approach.

2. The assumption that users won't have time for anything but informal problem specification turned out to be correct. Thus, the "stories" concept proved to play an important role when gathering requirements from the target audience in short meetings, on the go or at hackdays.
3. The project's stories have been publicly accessible at the PivotalTracker management tool continuously throughout the exploration phase of the project which was used to record the

aforementioned informal requirements. The "Icebox" still contains the list of features as prioritised by the various users who have been consulted during the exploration phase.

2.4 Requirements

In this case, the project's requirements (in a suitable share-able form) are actually part of the expected project output, as one of the main aims is to learn about the scholarly funding field and identify needs.

2.4.1 Requirements Gathering

The process was essentially arranged 0.5h and 1h meetings with interested users and talking about

1. what features they might find useful in a system such as FundFind and how the current list (as of the meeting) should be prioritised
2. about the scholarly funding field in general (in an attempt to get everybody's point of view and be able to learn what their needs might be from the differences in the way people approached the field)
3. about their role in this field (if any) in a further attempt to gather user needs information that might not have been otherwise articulated

The results of each meeting fed into the next one due to item #1. The final result is the list of requirements below.

It was quite difficult to get users to articulate what they need - FundFind does not match their current workflow, so there was a disconnect between the way they thought and the questions asked (e.g. "saving searches" means nothing to somebody who does not use a particular interface or system to search - it is akin to asking them if they save a copy of Google's results page for later use). This is the reason items #2 and #3 made it into the interview methods above after the first meeting.

Both of these methods required significant interpretation on the interviewer's side to produce useful results. While they certainly worked for generating new feature ideas, there wasn't much guarantee that those ideas would match the users' actual needs. Interpreting without much domain knowledge was one of the most difficult parts of the whole project.

2.4.2 Initial Requirements

The initial requirements were informed by the general needs of the various audience groups ("Audience and High-Level Requirements", §1.4).

They are listed below alongside changes which were made (kept/dropped/changed/changed priority). Further discussion on the lessons learned and future work enabled by the dropped features is presented in "Future Work", §6.5. Features which went through major changes are discussed below in "New Requirements", §2.4.3.

In **order of importance**:

2.4.2.1 Search For Scholarly Funding Opportunities

- **Why**: It's the implementation of one of the main project goals - making funding information discoverable. Every single audience group needs it in one way or another.
- **Priority rationale**: All users decided it was a top-priority feature, very much the main point of a system like FundFind.
- **Decision**: **Kept as-is**. No information suggested that this had actually become any less important than when the project started, and although the project goals changed when it became clear how much expertise was needed to actually do everything well, this was still an important software feature.

2.4.2.2 Submit Information About A Scholarly Funding Opportunity

- **Why**: In a similar vein to searching for such opportunities, multiple audience groups need it, albeit for different reasons (a funder might want to make their opportunity more discoverable, while a PhD candidate can just share a good opportunity with their peers).
- **Priority rationale**: Core to the crowdsourcing functionality of the project. Actually incentivising users to do this is a different matter entirely. That is part of the point of FundFind though - through providing the capability to share, it gives a reference point to related conversations and feedback, (hopefully) allowing developers to understand why the various users do not want to share funding information.

  of course, this requires the search functionality to be present to provide great value to the project and is thus still second.
- **Decision**: **Kept as-is**, for the same reasons as the feature above.

2.4.2.3 Submit Information About Funders

- **Why**: At the start of the project there were concerns that the various funding opportunities are vastly different in their purpose, format and thus, the data available about them. In order to alleviate this expected problem, this idea was presented to users and was found to be reasonable.

  In essence, if the funding opportunity data isn't good enough, a catalogue of funders would still be quite valuable. It was presumed to include some information to let the user search for an appropriate funder - like tagging the funders, or describing their interests.

  The idea actually came from a different Open Knowledge-related project called ACat (**A**cademic **Cat**alogue, currently not published on the web) - an attempt to mine all information on all academic publishers, their journals *and articles*. Even though ACat was just a hackday idea and hasn't evolved much since June 2012, it clearly showed the importance and value of having a list as simple as "all UK academic publishers".
- **Priority rationale**: Similar reasons to the feature above for this one to be so high up in the list in terms of crowdsourcing functionality. Its importance is very much tied to submitting information on funding opportunities - the more difficult it is to reconcile data from multiple sources (within the current scope), the more important this becomes.
- **Decision**: **Kept as-is**, for the same reasons as the feature above.

2.4.2.4 Create Profiles

- **Why**: Submissions of data could not be anonymous *to the system* - the submitter has to be authenticated and logged in some way, otherwise the submission forms will likely become the target of spam bots. It also helps with data quality, although admittedly on a very basic level - essentially, if a user is bothered to register, they are less likely to submit random junk into the system.

  This feature could include more than a username and a password however - it could include declaring research interest keywords and affiliations (the latter having been recorded as a separate feature at user meetings - see below).

- **Priority rationale**: Really desirable to enhance the basic functionality described above, but not as vital as actually having said functionality work in the first place.

  Other features which did not fit in the initial timeline at all but were earmarked for inclusion into a "Future work" section also require this (for example, research officers' reports "E-mail Alerts", §6.5.0.2).

- **Decision**: **Kept**. **Not much changed** except "could include more than username and password" became "includes more..." as no particular difficulties arose during the design or implementation of these additional fields.

2.4.2.5 Visualise Data

- **Why**: It was thought that the "Analyst Group", §1.4.3 would have an interest in this as it might spark other ideas about visualising funding data. Furthermore, scholars themselves might have had use for the result of the analysis - such as knowing how much their field is being funded at the moment.

- **Priority rationale**: Showcased the benefits of opening up funding data, which is one of FundFind's goals.

- **Decision**: **Dropped**. This turned out to be a little ill-defined. It embodied the quintessential software development problem of trying to predict what users want without actually knowing or being able to venture a good guess. The initial reasons as to why a visualisation might be useful are still true, of course, but it turned out developers who had done other Higher Education work understood the benefits of visualising such data quite well. Similarly, scholars could tell how well their field was funded just by looking at the data - the problem being that creating a visualisation that hid data appropriately and thus decreased cognitive load or increased insight was actually quite difficult. Representing the data in some much simpler visual way was not as hard, but also far from useful for people used to interpreting textual and numeric data. These insights were gained mainly during the Gateway to Research hackday ("Analyst Group", §1.4.3).

2.4.2.6 E-mail Alerts

- **Why**: Requirements gathering showed users using e-mail alerts / digests to receive the newest calls from funders. Thus, FundFind should probably offer a similar service as it federates such information.

- **Priority rationale**: Simply get users to use FundFind. With it, they would be getting the same information they could elsewhere, FundFind needs to present the same (or more) data in a better way, that's the value proposition. Thus, FundFind needs the data they could be getting elsewhere.
- **Decision**: **Dropped**. Turned out data - getting data - was more important than the gimmicks of managing data. This is where FundFind's role as a prototype and a limited vehicle of learning came into play.

2.4.2.7 Saving Searches

- **Why**: In a similar fashion to e-mail alerts, this was thought to be important as it allowed for better management of data and reduced repeated search effort.
- **Priority rationale**: Part of value proposition #3 - additional features. Other features such as e-mail alerts actually could be implemented more easily after this one is done - however, users did not think it was such a high priority. It also turned out that searches can already be shared just by copy/pasting the URL of the search page ("Search for Scholarly Funding Opportunities", §4.3.1.1.
- **Decision**: **Dropped**. Similarly to e-mail alerts, it turned out data is the first thing to get into the application and leave enhancements for later.

2.4.2.8 Harvesting Funding Data From E-mail Digests

- **Why**: This is one of the features which actually deal with importing data, fulfilling part of FundFind's core mission of federating funding data.
- **Priority rationale**: The importance of this was downplayed by the author - users did not have much to say on it, since they did not particularly care for the details of how FundFind was going to get its data, as long as it had some and they could use it.
- **Decision**: **Changed**. This is a good idea - however, all the information turned out to be available in machine readable formats (for the funders which were considered). It was also technically quite challenging.

2.4.2.9 Tagging Research as "Potentially of Interest to" Users and Groups

- **Why**: Part of showcasing the benefits of opening up and sharing funding data - enables scholars and research development officers to target the sharing of opportunities. May also enable funder representatives to target their funding calls better, although care has to be taken since the two sides of the "advertisement" may well have different goals.
- **Priority rationale**: This is a nice feature, but builds on profiles, search and submissions working, so just has to be done after them.
- **Decision**: **Kept as-is**.

2.4.2.10 Specify Affiliation and Interests

- **Why**: This is actually the "other end" of tagging research - specifying where the user works (institution, research group, country) and what field the user works in (interests) enables the usage of the tags associated with research data.
- **Priority rationale**: Similarly to tagging research, this feature builds on everything before it, clearly part of value proposition #3.
- **Decision**: **Kept as-is**.

All initial features described below were lower priority or in some way optional in relation to the main aims of the project.

2.4.2.11 Responsive Mobile Web Design

- **Why**: FundFind is all about sharing data, thus it made sense to let people share in as many contexts as possible. The Design Rationale "Design Details and Rationale", §3.1.1 elaborates further on this feature, which is actually a UI design characteristic. It was also of personal technical interest to the author - responsive web user interfaces are everywhere in the Open Knowledge field. Some work better, some worse, yet one thing is clear - learning how to develop them is important. Preliminary research also showed that implementation might not actually be that difficult ("User Interface And Responsive Mobile Web Design", §4.3.2.3), especially with a more lax testing strategy ("Responsive Mobile Web Design", §5.3).
- **Priority rationale**: As technically nice as this is, it was certainly not prioritised highly by users. Therefore, it was given a lower priority than the main functions of the application.
- **Decision**: **Kept as-is**.

2.4.2.12 Bulk Funding Opportunity Data Import

- **Why**: Importing funding information records one-by-one can get quite boring, time-consuming and tiring - all the characteristics of a task that is ready to be automated. This was mainly conceived as a feature for the Analyst audience group ("Analyst Group", §1.4.3), specifically more advanced users who might want to use the software itself and load it up with data they are interested in - not the data all other users have shared on the public FundFind instance. This also includes potential contributors to the code. Another potential target was funder representatives, who may have access to funding opportunity information from their organisation in a structured format and would rather just upload it than copy/paste the records one-by-one.
- **Priority rationale**: This feature has quite a narrow focus - a (very) narrow subset of the audience of the project.
- **Decision**: **Dropped** - it was just really of very limited use. Even backing up the data in the public FundFind instance does not require this feature, as the datastore can be backed up with a simple file copy command ("Datastore", §3.3).

2.4.2.13 Harvesting Historical Funding Information

- **Why**: Historic funding information can be quite valuable since it can grant insight into what was previously funded. If this includes the most recent 1-10 years of data, it could be used to see how fields develop. While hopefully no scholar would choose their field based solely on its history of funding, they could see how to pitch their work most effectively - often, it is difficult to strictly categorise research.

  Something which came up during meetings was the fact that helping gain such insights from historical data might be just as, and sometimes more, helpful to *obtaining* funding than just the sheer ability to *find* it more easily.

- **Priority rationale**: FundFind had to focus on current funding opportunities.

- **Decision**: **Dropped**.

2.4.2.14 RESTful API

- **Why**: Allow integration with other Open Knowledge projects, also vital to being able to consider FundFind an Open Knowledge project. If data is being federated from multiple sources, then it better be exposed in some way. The Design Rationale "Design Details and Rationale", §3.1.1 elaborates further on this feature.

- **Priority rationale**: Targets only the Analyst audience group ("Analyst Group", §1.4.3).

- **Decision**: **Kept as-is**.

2.4.2.15 Exporting The Results of Searches

- **Why**: Simple convenience - being able to save a list of relevant funding opportunities for later perusal.

- **Priority rationale**: Everything else has some function beyond pure convenience and this would not save that much time or effort in any case. Similar to saving searches ("Saving Searches", §2.4.2.7), searches can already be shared via the search page URL ("Search for Scholarly Funding Opportunities", §4.3.1.1) which further reduced the priority of this feature, since the same search should yield quite similar results (based on the assumption that a search worth saving is a pretty specific search). This was also consistently rated as low priority in user meetings.

- **Decision**: **Dropped**. It is hard enough to find *one* highly relevant funding opportunity - finding multiple ones is not something that will happen easily without getting quite a lot of data into FundFind.

2.4.3 New Requirements

A number of new features were inserted into the initial list during development - mostly stemming from new knowledge about related projects and data, or what was technically feasible.

2.4.3.1 Harvest Funding Data From Machine-Readable Sources

- **Why**: It is far easier to process RSS feeds such as the EPSRC Open Calls RSS Feed [29] than it was to process the EPSRC Funding Call E-mail Alerts.
- **Priority**: Replaced "Harvesting Funding Data From E-mail Digests", §2.4.2.8.

2.4.3.2 Suggesting Relevant Historical Data

- **Why**: As discussed in "Harvesting Historical Funding Information", §2.4.2.13 above, it was discovered that this is actually a good way to achieve the end goal of funding scholarship. This project may have started as being about discovering *current* funding information, but this itself stemmed from the overarching goal getting more funds to more scholars. This only became possible quite late in the development process as Gateway to Research's data was discovered by the author in March and gave a concrete foundation on which to build this feature.
- **Priority**: Lowest, tacked onto the end of the feature list. Very experimental due to potential data issues, lack of discussion with actual scholars (only other developers) and lack of clarity around presenting the information on the user interface. It was also technically quite ill-defined and difficult to achieve.

2.4.4 Final Requirements

1. Search For Scholarly Funding Opportunities
2. Submit Information About A Scholarly Funding Opportunity
3. Submit Information About Funders
4. Create Profiles
5. Harvest Funding Data From Machine-Readable Sources
6. Tagging Research as "Potentially of Interest to" Users and Groups
7. Specify Affiliation and Interests
8. Responsive Mobile Web Design
9. RESTful API
10. Suggesting Relevant Historical Data

2.5 Design Process

The technical design approach was certainly bottom-up. It would start with a high-level idea, like the features described above, and proceed with looking up or creating an example (for harvesting / submitting data). Some sample code which did something very basic (e.g. loaded the example data) would follow. This was then transferred to the appropriate module and enhanced iteratively, using manual testing to see each version was working, until the code satisfied the goal. This iterative process was accelerated by the choice of programming language - in particular, the supporting tools, discussed below.

Other features such as searching did not involve that much technical design as certain components like facetview (for search) had already been chosen since they worked well with the datastore. The difficulty there was not in understanding how to solve a problem in principle, it was making the designated solution work by bringing different components together.

2.6 Technologies Used

As "Existing Works", §1.5 and "Design Details and Rationale", §3.1.1 point out, Python was chosen as the main programming language of the application. HTML5, CSS and Javascript were used for the user interface. Similarly to the Python choice, this was due to multiple other Open Knowledge projects using them in this manner - so it would be easy for interested developers to read the code and contribute. Previous personal experience with these technologies helped as well, as this is a sole developer project.

Basing the main application on another project that the author had participated in meant that a great deal of bootstrapping time was saved. A lot of knowledge about using the underlying Python frameworks was also reused. This was also partially true when the FundFind data harvester was written as it was again based on another Open Knowledge project, but the author had no previous experience with that code.

On the other hand, trying to create something to the standards enforced by other projects in the same context (Open Knowledge) forced quite a lot of reading that might not have been strictly necessary outside this context.

Python's interactive (REPL - Read, Evaluate, Print, Loop) interpreter was of great help to the development process. Instead of having to write a program in a file and then run it, the developer can type out the rough sequence of commands they want and see the results immediately. Seeing the results of a *previous* command can be very important if the developer does not know or has just forgotten exactly what its output will be or look like. If the interpreter is run in the project's directory, all modules and other existing project bits can be accessed in the same way as in a written script.

Python's philosophy of readability (no semicolons, indentation matters, new line means new statement) and certain programming conventions which have been built up around it meant that the parent codebase was quite easy to understand.

Previous experience combined with the usage of relevant libraries meant that the quirks in developing with HTML and CSS - mainly the way CSS applies to HTML elements - did not delay development. On the other hand, having to learn how to work with the libraries certainly took some extra time, but saved a lot more ("User Interface And Responsive Mobile Web Design", §4.3.2.3).

The RESTful JSON API that the chosen datastore (elasticsearch) provides also accelerated the process somewhat, since the all of the project's data could be retrieved by pointing a web browser at
`http://localhost:9200/fundfind/_search?q=*`.

The command-line **Vi** **IM**proved (vim) editor was used in conjunction with multiple graphical console windows (Konsole on KDE, Ubuntu GNU/Linux). This was not because of particular familiarity with vim, but rather because of a desire to gain such familiarity. It is also the only editor available on the testing server which currently hosts FundFind's public instance. Certain common text operations did actually become faster to accomplish in vim than in a conventional WYSIWYG editor over time (deleting and copying bits, lines or blocks of code being the most noticeable). On the other hand, quite a lot of time went into actually learning to use it.

Chapter 3

Design

3.1 Introduction

The technical design of an application is, in essence, the intersection of current "best practice" in the software development field (and the related programming language communities), its target domains and the user needs.

"Technically", FundFind:

- is a web application
- is written in Python
- stores its data in an asynchronous NoSQL data store with built-in search capabilities
- has a mobile-friendly web interface
- exposes all its major features via an API

These characteristics have not changed since the start of the project. The technical decisions that stand behind them proved to be mostly correct and helpful, and are elaborated upon in the next section.

The parent IDFind project had not done anything like one of the new features which were added during development - harvesting funding data from machine-readable sources. The new Funding Harvest feature is actually technically separate from the rest of FundFind's codebase - it just puts whatever information it finds in FundFind's datastore and the only link between the two codebases is the datastore. Developing it separately made sense as it can thus be readily re-used and changed.

3.1.1 Design Details and Rationale

FundFind

- is a web application

  This was decided right from the start of the project. The way current Open Knowledge Foundation and Cottage Labs projects worked was important since one of the two major target domains is Open Data / Knowledge and Cottage Labs is an interested party. Most OKFN projects are web applications - even the library ones have mostly been used in web applications. Cottage Labs have also focused on web applications. Such applications have

just proven themselves to be far more "co-operative" than traditional desktop applications - easy to access, easy to modify so that they expose their data via an API (interoperability). "API", §3.7 details further interoperability decisions - the transport format (JSON) and protocol (HTTP).

- is written in Python

  Similar factors to the interoperability point raised above played a role in deciding the language of the application. A lot of Open Knowledge projects use Python [44, 45, 50, 52] - the same goes for Cottage Labs projects [7, 9, 10, 12, 13, 54, 59]. The language is also simple and strives encourage and enable readability - "code is read much more often than it is written" [34].

- stores its data in an asynchronous NoSQL data store with built-in search capabilities

  FundFind relies on elasticsearch to store its data. Elasticsearch is an indexing server which allows for sophisticated search queries against a body of text and other data [78]. The essential advantages were simplicity, performance, usage by other Cottage Labs projects and certain options it leaves open for further development (see "Datastore", §3.3).

- has a mobile-friendly web interface

  FundFind is an application which tries to enable sharing of information - one of the highest priority features. Owing to the nature of the information it makes sense to try to make it mobile-friendly - scholars do not necessarily have to hear about funding opportunities while sitting at their desks.

  Whether they will also want to share this rather dense information while on the go is another matter entirely. Mobile-friendliness was not noted as having particularly high priority in the Progress Report, and it still does not - it was just easy to implement due to the fact that the libraries used to make the web user interface follow current best software engineering practice. More details are available in "User Interface And Responsive Mobile Web Design", §4.3.2.3. Essentially, making a good UI by using the libraries properly would have led to a mobile-friendly UI (at the flick of a filename and a few CSS class names).

- exposes all its major features via an API

  The JISC Report "Advantages of APIs" states "the API enables use and re-use; it is a tool by which we can disseminate knowledge" [37]. This is the essential advantage of API-s and the reason the concept of an API has become one of the building blocks of the Open Knowledge movement. If FundFind aims to prototype a useful tool which will eventually contribute to Open Data (so taking into account the needs of the "Analyst Group", §1.4.3), it needs to eventually have an API. It is simply better to design the project with the API in mind instead of tacking it on later. One example is the HiFi project, which had the following to say on the subject:

  > "We built the API to satisfy these requirements first, then we built our app on top of the API. This turned out to be a great idea. We got to dogfood our API for the entire development process and it made testing a lot simpler."
  >
  > *Kris Jordan in "First we built an API, then we built a CMS" [39]*

3.2 Overall Architecture

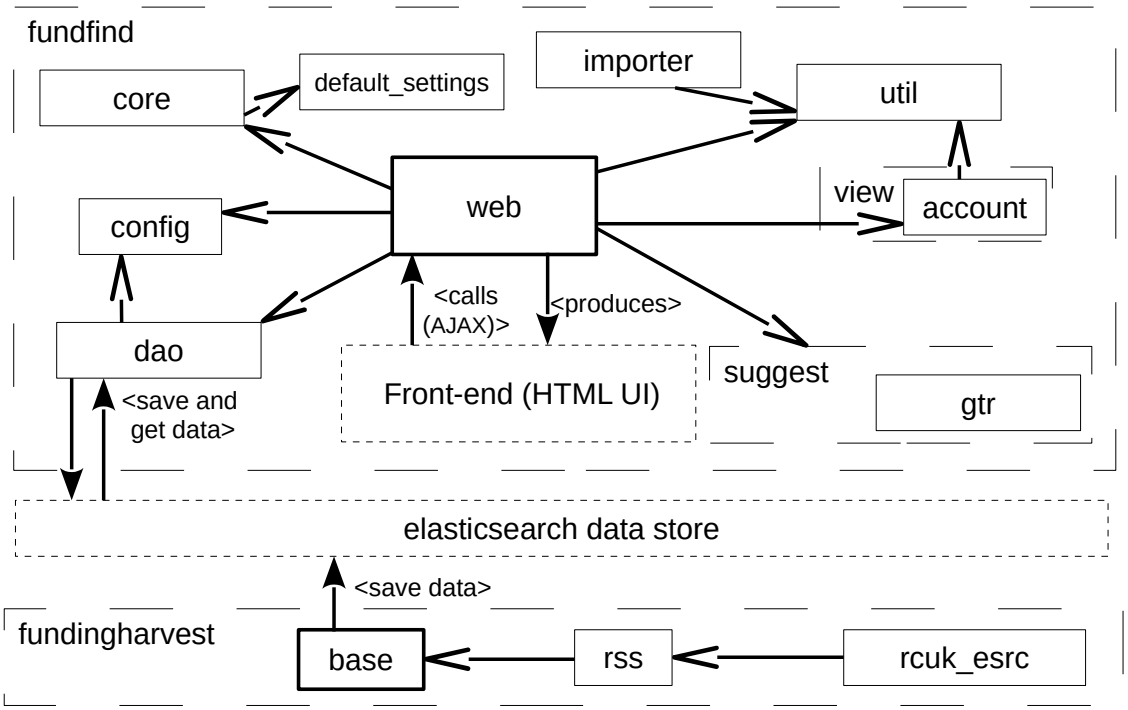

Figure 3.1: Overview of FundFind and Funding Harvest's technical architecture.

The entities drawn with continuous lines in the diagram are modules - collections of code related to solving the same problem. The contents of the modules - the details of the technical design of the main application - are discussed below in §3.4 to §3.5.

The entities drawn with a heavy continuous line are the entry points - they are the modules which actually (need to) get run by the Python interpreter.

The entities drawn with a rough dashed line are packages. Packages and modules are simply ways of organising code in Python and are discussed in more detail below.

The entities drawn with a fine dashed line are more abstract concepts (datastore and user interface) which are elaborated upon below.

The empty arrowhead arrows denote dependency - more details below.

3.2.1 Organising Code in Python

Traditionally Object-Oriented Programming talks about "classes" as, essentially, blueprints combining data structures and imperative code (methods) together. These can then be "instantiated" to produce a concrete "object".

This is of course well-supported in Python. However, Python also has an additional convention when it comes to organising code - "modules". The Python documentation says "You may also want to use a handy function that you've written in several programs without copying its definition into each program." [56]. While re-use of code is usually achieved by making everything an object in other languages like Java, Python actually allows functions and even just code statements at the top level of a source code file. Thus, instead of having to create a "Util" class with several

static methods, Python authors are encouraged to simply define the functions they need at the module level. In other words, modules are a convenient way to bundle together *related* pieces of code - whether that would just be a sequence of statements, function or class definitions. This may seem messy but has proven to be quite efficient by essentially allowing the developer to better express their mental model of how their program should be organised, instead of forcing a particular structure.

An obvious characteristic of Python modules is that they are also implicit namespaces. Thus, the built-in `int()` can easily be distinguished from `random.int()`. A Python program would have to `import random` before it can use `random.int()`.

In this case, `random` happens to be a module that is part of the standard Python distribution. However, exactly the same rules apply to user-defined modules. Thus, the empty arrowhead arrows which connect the modules in Figure 3.1 essentially mean "module X imports module Y and uses something from Y".

Packages are simply a collection of modules. Since a module is a file, packages are just directories (containing a possibly empty `__init__.py` file) - another way to organise related code.

To sum up, packages contain modules. Modules could contain classes (alongside any other code and function definitions).

Furthermore, certain naming conventions exist in Python for package, module and class names, which account for the names used in fundfind and fundingharvest projects.

> "Modules should have short, all-lowercase names. Underscores can be used in the module name if it improves readability. Python packages should also have short, all-lowercase names, although the use of underscores is discouraged.
>
> [...]
>
> Almost without exception, class names use the CapWords convention."
>
> Style Guide for Python Code, Prescriptive: Naming Conventions [34]

3.2.2 Changes Over Time

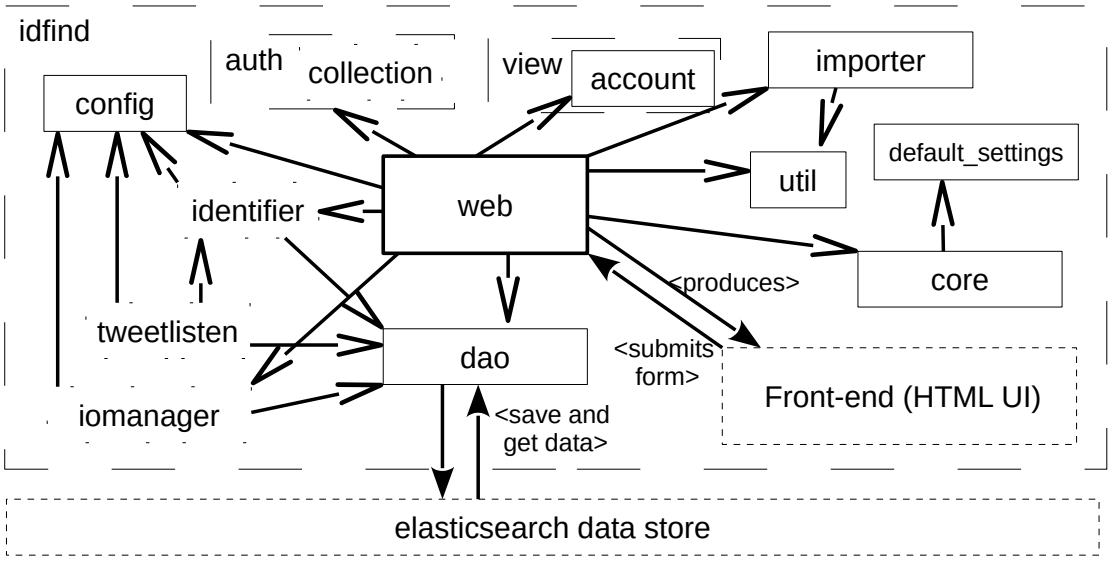

Figure 3.2: Overview of parent project IDFind's technical design. Much more complete than presented in the Progress Report.

The list of core modules which form the main web application has not changed much from the parent IDFind project (the content has, of course). The diagram of IDFind's technical design presented in the Progress Report [25] was meant to be an overview diagram and had left out several details. These are now included in Figure 3.2. The modules presented with a very thin dashed line have been removed in FundFind. The others have been left in, but several new ones have been added.

Those which were deleted merit further discussion:

- `tweetlisten` allows IDFind to respond to Twitter requests in a similar fashion as if the user was using the web user interface.

  This module was scrapped in FundFind since potential target users (including the author) just could not think of a way in which Twitter integration would be *immediately* useful. There was also an issue with the inherited tests, discussed in "Overall Approach to Testing", §5.1.

- the whole `auth` package, containing the `collection` module was not being used for anything else but to check whether a user had certain privileges (probably left over from when IDFind itself was forked from another codebase). All dependencies on that were replaced with referenced to the `current_user` object provided by the Flask-Login extension (that is all `auth.collection` was actually using anyway).

- `iomanager` again seems to have been inherited from IDFind's parent codebase. It was essentially being used as a generic way to display objects from the elasticsearch datastore, but only when there was no other appropriate way to display them. This happened when none of the routes in the `web` module responded to the incoming request. IDFind had inherited a generic search functionality with the idea of "search all objects in IDFind's datastore" but was quite limited and didn't visualise the results very well. `iomanager` was the bit that did the visualisation for this generic search. In FundFind, with facetview, and indeed direct

(at least read-only) access to the elasticsearch datastore, this module just became obsolete and the limited search functionality was stripped from the main navigation bar.

In terms of changes, the contents of all of these modules which were kept have of course changed to accommodate FundFind's aims.

Furthermore, a new package, `suggest` and one module within it, `gtr` were introduced in FundFind, with the aim of offering up suggestion for relevant past funded projects to FundFind users. Both are described in more detail in "suggest", §3.4.8 below.

3.3 Datastore

A discussion of the nature of the slightly unconventional datastore is in order since it has informed the technical design of both FundFind and Funding Harvest.

Indexing software is usually most easily understood in comparison to more traditional approaches of storing data, like relational databases (MySQL, Oracle). These store data in a highly structured way and the base structural unit is a table (theoretically equivalent to a set, which is where relational theory and such databases come from). The "indexing" part actually means "analysis" in the broadest sense. Given data - any data, images, text, Microsoft Word binary blobs pretending to be text - the software will try to find common features or look for certain characteristics in the data. For example, it will try to discern whether a particular string actually represent a datetime value.

Elasticsearch is actually a NoSQL datastore - which essentially means that it deals with very simple key-value pairs as the basic unit of organising data *instead of* tables, or sets. It also supports slightly more complex, but still very simple structures, such as lists and dictionaries (a.k.a. HashMaps). This means that the storage representation is also quite simple - in this case, all documents stored within an elasticsearch instance can be represented in the JSON data format. This simplicity enables elasticsearch to deal with semistructured data much more easily than a traditional relational database.

In terms of individual values, it should be noted that they are all optional - elasticsearch has not been configured, in this project, to enforce presence of particular data - this is up to the application pushing to the index. This means that data validation is essentially entirely up to FundFind and Funding Harvest.

A lot of funding data can be considered semistructured - one type of funding call might have a closing date, another one might not. When multiple data sources are considered it becomes even more difficult since different organisations publish different bits of data about their available funding. Since elasticsearch combines its simple storage with a powerful, fast search engine this makes such "holes" in the data a lot less important than usual - the "usefulness" of the data scales with its quality (there is no getting around the fact that less holes is better), but *the software can deal with it* without throwing null pointer exceptions and *the data that is present is still easily discoverable.*

Elasticsearch's RESTful JSON-returning API usually enables rapid prototyping, evaluation and integration with Javascript visualisation libraries such as d3 [77] and BubbleTree [51]. While these precise features are not implemented in this version of FundFind, the decision to use elasticsearch leaves this option open.

Elasticsearch allows the organisation of the data in a particular instance into "indexes" (large units very similar to databases in other datastores). Those can be further organised into "types" (the idea being that each indexed document has a type of some sort).

Each document also has to have an `id` (beyond `_type` and `_index`). This is unique within the type and index of the document [21]. For example, it is not possible to have two `funders` with an id of `rcuk`, but it is possible to have a `rcuk funder` and a `rcuk account`. Elasticsearch can assign this when it indexes the document, but the parent IDFind codebase used to generate and assign UUID4-s (random 32-digit numbers).

FundFind inherited this behaviour. Apparently, according to other IDFind authors, it is not strictly necessary in IDFind, but allows for custom `id`s to be generated by an algorithm if need be - so in this case it is a placeholder, an architectural pattern. It's done in a fairly elegant, generic way, described in "dao", §3.4.2.

Multiple running elasticsearch instances can be easily joined together to form a cluster for resilience and performance. This is not used by this project but is discussed since it is an important scalability characteristic, especially for "young" software such as elasticsearch.

Elasticsearch is extremely easy to back up - the API can be used to get a JSON data dump, but the `data` directory in the local elasticsearch installation can just be copied using the usual filesystem utilities and network storage techniques (e.g. an encrypted copy could be produced and uploaded to Dropbox via a one-line shell script). Elasticsearch will actually replicate its data automatically and instances which can see each other (in the same logical network) will find each other and can be set up to replicate each others data.

3.3.1 FundFind Datastore Details

FundFind uses the `fundfind` index and categorises the documents it stores in the types described below. Due to the nature of an indexing engine, it is easier to not validate the properties that document types have - to leave them as simple conventions, names that the code *should* use to access certain pieces of data. The code is free to sometimes save a user's username in an `id` field, and in `user_name` at other times. It should not do that, but it is not forced to - however, see "dao", §3.4.2 for the convention the code uses to prevent this from happening in practice.

- `account` holds user profiles. Its properties are:
  - `api_key` - Created by the Flask-Login extension, can be used to access non-public parts of the API (like editing a user profile). FundFind does not currently expose any non-public functionality through the API due to the additional complexity, security risk and low priority of the possible functions which need it.
  - `country, organisation, department, research_group` - these specify a person's affiliation.
  - `created` and `modified` - timestamps of when the profile was created and last modified. The data format does not matter, as long as it is something that elasticsearch understands to be a date, which is quite an exhaustive list [20]. FundFind uses the `datetime` Python module to get the current time in ISO format: `2013-04-10T11:42:09.535000`.
  - `email` - the user's email
  - `id` - In this particular case, the random digit id-s described above are overridden with the username, e.g. `emanuil`. This ensures that there will never be two users with the same username and makes it easy to tell users that the username is taken (see "Datastore Access", §4.2).
  - `interests` - a list of research interests specified at registration of the profile

  - `password` - a hash of the user's password. See "Create Profiles", §4.3.1.3 for more details.

- `funder` holds information about funding organisations.
  - `created, modified` - timestamps with the same function as the ones in `account`
  - `description` - Who is the funder, i.e. what sort of organisation are they? Details beyond the name.
  - `homepage` - URL (hopefully) pointing to the organisation's home page
  - `id` - random number as described above
  - `interested_in, policies` - A string describing this funder's area of expertise.
  - `license` - an object (dictionary) describing the license this data is available under. For items coming in from the user interface, i.e. crowdsourced, this has a `type` of "cc-by" and a `version` of "3.0", as well as a `url` of `http://creativecommons.org/licenses/by/3.0/`. Items coming in from Funding Harvest will have a `type` of "default-copyright" and the other fields will be blank. This may change in the future as attempts are made to convince publishers of funding data to publish it more openly.

    Structure inspired by the way OpenArticleGauge describes licenses [55].
  - `name` - name of the funding organisation
  - `owner` - the user who created this `funder` document.
  - `tags` - a list of strings related to the funder - perhaps their field, policies (e.g. "open access required", "green OA friendly") or anything else the users feel like tagging
  - `useful_links` - a list of URL-s where more information can be found, e.g. the policy page of a funder, reports published by them, even the wikipedia article about them

- `funding_opportunity` describes a funding opportunity.

  It shares quite a few keys (field names) with `funder` - `created`, `license`, `modified`, `id`, `owner`, `tags` and `useful_links` all have very similar functions.
  - `closing_date` - the date after which the funding opportunity goes "historical", i.e. no more responses / bids / expressions of interests are accepted by the funder for this opportunity
  - `funder` - an object (dictionary) describing the funder who is offering this funding opportunity. At present this is not a link to a `funder`-type document, it just contains `name` and `url`.
  - `funds` - amount of money offered
  - `funds_exactly_or_upto` - qualifies the figure in `funds`. What it means depends on the size of the opportunity - sometimes funders will announce opportunities for millions of pounds and expect to fund tens of project through this one stream of money. Some might be smaller, some larger - the wording used is often something like "up to X thousand / million pounds are available for research into [...]". An "exactly" value means that when projects specify how they are going to use the money, they will have to come close to that value to demonstrate they will use the funds effectively.

    Correctly interpreting this value requires funding domain experience and will probably need to be informed by the content of the `more_info` field as well.

- `issue_date` - when the opportunity was published (by the funder)
- `more_info` - more information about this opportunity - background, eligibility criteria, whatever is available
- `title` - title of the funding opportunity (given to it by the funder)
- `url` - URL pointing to the opportunity's page for more details. This is the "canonical" URL for this opportunity - the useful links can point to any web address with relevant information, but this is *the* opportunity's page.

3.4 fundfind

The Flask Python micro web framework [80] is used in this project. A web framework makes it easy to write code which responds to web requests. Instead of having to deal with the low-level networking details, including a template engine and finally - serving HTML/CSS/Javascript content, a web framework makes it easy to focus on the content. There is a tendency with certain web frameworks to enforce an extensive structure of files and decisions (such as what database to use) on the developers. The "micro" in Flask means it makes a minimal set of decisions, but features can be plugged in via its extensions, or even just ad-hoc code which makes use of the core Flask features [81].

3.4.1 core

`core` is, by convention the name of the main module in a Flask-based application. It creates the main application object and loads the configuration, including putting in the default settings.

The call to start the application as a web service listening on a certain port is done here as well. If debug mode is enabled `core` will monitor all application code and reload components on-the-fly if changes are detected.

3.4.2 dao

`dao` stands for DAO, which stands for Data Access Object – a database abstraction layer. Contains all the model classes of the application as well as functions which deal with the elasticsearch data store.

This module defines two functions and three classes, presented in Figure 3.3. The `**kwargs` notation means that that method takes a variable number (0..*) of keyword arguments (a.k.a. named arguments in other languages). In this case this is used to pass data to the constructor:
`funder = dao.Funder(name='name', tags=['tag1', 'tag2'])`
and to pass certain options to the elasticsearch server (in the query method), e.g.:
`dao.Account.query(q='emanuil', es_opt1='val1', es_opt2='val2')`

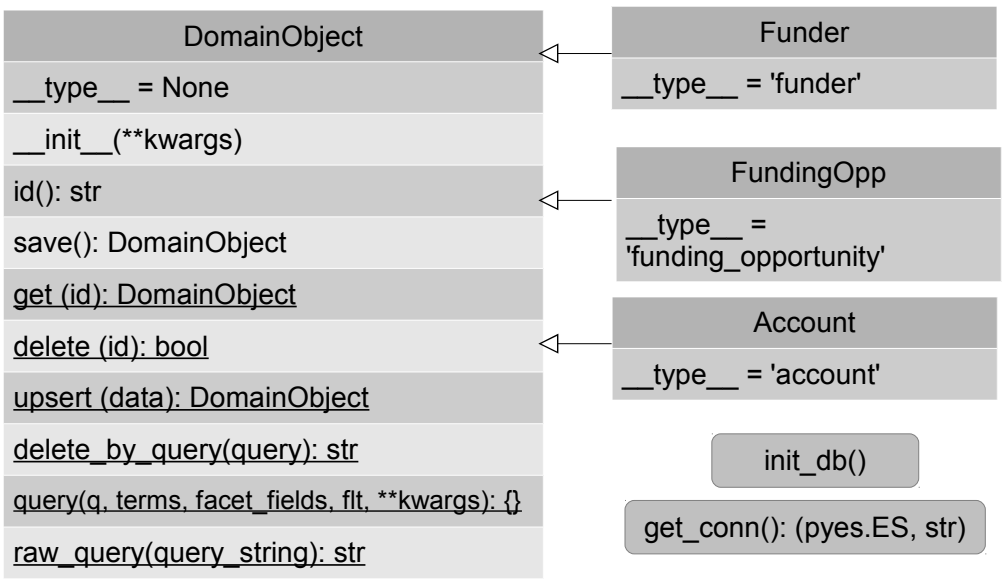

Figure 3.3: dao module overview - 3 classes and 2 functions

The `DomainObject` class defines all the necessary operations. Most of them are defined as class methods since instantiated DAO objects are rarely used in the codebase - since it's a web application, the in-memory objects are lost from one request to another, or one page to another. Instead, the datastore is used for persistence.

`DomainObject` inherits from `UserDict.IterableUserDict` in the core Python libraries, which means that it and its subclasses can be treated as a normal Python dictionary, and normal Python dictionaries can be easily converted into these classes. In practice, this means being able to do things like `funder['name'] = request.values.funder_name` as well as being able to create objects by doing:

```
incoming_data_dict = {
    'name': request.values.funder_name,
    'tags': process_the_tags_function(request.values.tags),
[...]
}

[...]

new_funder = dao.Funder(incoming_data_dict)
```

Everything in the datastore is a dictionary (or a list) so the objects themselves serve no further purpose that to define the type of the document which is being uploaded. Thus the `upsert` static method, which takes care of uploading documents to the index, takes a simple dictionary and uploads its contents to the datastore. The only difference between a user `Account`, a `Funder` and a `FundingOpp` is in the class name and associated `__type__` value, which gets passed on as the first argument to `upsert` in this line:

```
new_funder = dao.Funder(incoming_data_dict) # note class name
    # telling upsert what type the document is
```

The `upsert` method also generates and includes `id`s in incoming data, unless the data already contains an id. This gets executed for all objects which are uploaded to the index, making it easy to change the `id`-generating algorithm. It can also be overridden on a per-class basis, i.e. `Account` may wish to generate different `id`s if they are not specified.

The `get` and `delete` static methods take an object id and operate on it (either getting the contents from the datastore or deleting that document).

`delete_by_query`, `query` and `raw_query` simply provide means of interacting directly with the datastore to fetch or delete a larger group of documents.

3.4.3 web

This module is the "web server" of the application – it defines the routes and the corresponding functions which handle them (in a RESTful way), runs constantly and outputs exceptions to standard output. If more logic is needed to respond to a certain request than will fit comfortably in one short function, then the function can load another module (e.g. Importer).

If the HTTP POST and GET processing differ a lot, or other advanced features are needed, then Flask "views" can be used - these are classes which respond to HTTP requests.

The two ways to do routing in Flask which FundFind uses are illustrated below.

This is the route to the index, or home page. It does nothing but call on the Flask framework to process the given template (which is in Jinja2 format, not pure HTML as the extension suggests) and serve it to the user.

```
@app.route('/')
def home():
    return render_template('home/index.html')
```

This is a slightly more elaborate example:

```
class ShareFundoppView(MethodView):
    '''Submit information about a funding opportunity'''
    def get(self):
    # respond to HTTP GET
        if current_user.is_anonymous():
[...] # do some processing
        return render_template('share_fundopp.html',
  active_page='share_fundopp')

    def post(self):
    # respond to HTTP POST
        if current_user.is_anonymous():
            abort(401)

    [...] # do some data validation
            return render_template('share_fundopp.html')

# Register the view
```

```
app.add_url_rule('/share_fundopp',
  view_func=ShareFundoppView.as_view('share_fundopp'))
```

The last line adds the new view class to the routes, so that it will be handed all requests which come in at `/share_fundopp`.

These are all routes that `web` defines:

- GET `/` - home page
- GET `/account/<user>` - view a user profile (would have to be logged in and can only view own profile)
- GET `/content/<path:path>` - generic route to server "static" content like tutorial pages, API pages and so on (only works if the supplied filename actually matches a template, otherwise 404 Not Found)
- GET `/search` - search for funding opportunities and funders. This does not have a JSON form, i.e. it is not part of the FundFind API, it's just the HTML page which exposes FundFind's data. This is because FundFind's datastore of choice has far more extensive search capabilities for exposing the data than this project could ever hope to provide, including complex queries, fuzzy-like-this queries and scripts inside queries (dynamically ranking results based on an algorithm embedded in the query, i.e. dynamic fields).
- GET, POST `/describe_funder` - share information about a funding organisation
- GET, POST `/share_fundopp` - share information about a funding opportunity
- GET, POST `/slugify` - exposes the `util.slugify` function to the world via the API. This is mainly of use for AJAX requests although API consumers could use it if they wish to.
- GET, POST `/suggest` - ideally, suggests a mix of artefacts related to the `similar_to` parameter it is passed, like news articles, projects, funding organisations. However, FundFind only supports projects from the GtR API at the moment, so this does the same as the route below.
- GET, POST `/suggest/projects` - suggests projects related to the `similar_to` parameter it is given, currently only from the Gateway to Research API.

3.4.4 importer

A module which contains the Importer class. This handles data collection – submissions to FundFind via the `/describe_funder` and `/share_fundopp` routes. The Importer class gets instantiated and used by the `web` module. This is a pretty simple module which mostly deals with processing the data coming in from the API or the user interface information sharing forms. It has two methods - `describe_funder` and `share_fundopp` which mostly clean up incoming data. The data is retrieved from the `request` object, which is a Flask framework-provided object which represents the current HTTP request - all its parameters (in `request.values`) and other related information.

The module does define three internal helper methods, which could arguably be moved to the `util` module if there was any use for them anywhere else in the codebase:

1. `_clean_list` takes a list of items and strips whitespace off both ends of each item in the list. For example, if ['tag1', ' tag2', 'tag3 '] come in, it will return ['tag1', 'tag2', 'tag3']. This can happen quite easily since users are allowed to enter tags as a comma-separated string on the user interface, and this is then broken up by removing the commas and putting the resulting parts in a list - but it doesn't handle whitespace.

2. `_prep_link` takes a string representing a URL (e.g `'www.aber.ac.uk'` and prefixes it with `'http://'` if it doesn't already start with `'http://'` or `'https://'`.

3. `_str2isodt` converts incoming strings to Python datetime values. It uses the third-party `parsedatetime` module to try and guess the date format, since this module supports more formats than a single person can (usually) imagine.

3.4.5 config

This class is a wrapper around a simple JSON dictionary of key-value pairs which contains hard-coded configuration information.

The root directory of the repository contains `config.json`, which looks like:

```
{
    # service
    "service_name" : "FundFind",
    [...] # more settings
}
```

The `config` module makes it possible to say `from fundfind import config` in, for example, the `dao` module. The wrapper functionality essentially allows `config['service_name']` to then return `"FundFind"`, going by the example above.

3.4.5.1 default_settings

`default_settings` is a Python file which simply contains several constants that toggle various user interface frameworks' options, like:

```
# override certain defaults of the Flask-Bootstrap extension
BOOTSTRAP_USE_MINIFIED = False # small performance gain [...]
BOOTSTRAP_JQUERY_VERSION = '1.9.1' # most up-to-date [...]
```

The information is merged with the rest of the configuration when it is loaded.

The reason there are two separate modules is because this design was recommended - this is a fully-featured Python file which could do far more than simply stating values, like the `config.json` file has to. The reason the `config.json` file still exists is that it works, it is easy to understand and work with, and it simply has not yet been merged into `default_settings`. It is debatable whether the configuration should be kept in a code module or more a immediately visible location such as the root of the code repository. Time constraints preclude such debate in this project, however.

Right now it does not actually hold any useful settings since the Flask-Bootstrap extension is no longer in use, due to certain difficulties described in "User Interface And Responsive Mobile Web Design", §4.3.2.3.

3.4.6 util

Defines several functions which are generic and can be used throughout the code.

`slugify` turns normal text (including Unicode text) into a more restricted version of URL-friendly text, called a "slug". It replaces whitespace and other delimiters and punctuation with underscores. For example, "Inertial Fusion Energy in the UK and Beyond - !?" becomes "inertial_fusion_energy_in_the_uk_and_beyond".

`clean_list` strips off trailing spaces off all the items in a list. It was previously written by the author for the IDFind project and its docstring (long-form in-code description) describes its purpose best, with a (modified for this document) example:

```
Example: you have a list of tags. This is coming in from
an HTML form as a single string: e.g.
research_interests = "research interest 1, birds, ".
You do research_interests.split(",").
Now you have the following list:
["research interest 1"," birds", ""]
You want to both trim the whitespace from list[1] and remove the empty
element - list[2]. clean_list(tag_list) will do it.

What it does (a.k.a. algorithm):
1. Trim whitespace on both ends of individual strings
2. Remove empty strings
3. Only check for empty strings AFTER splitting and trimming the
individual strings (in order to remove empty list elements).
```

`prep_link` simply puts `http://` in front of a link (perhaps specified as a useful link related to a funder) if it is not already there. It can also add a trailing slash, triggered via an optional argument.

`str2isodt` can take a string and try to (quite intelligently) guess whether it is a date, time, or both together. It is a convenience wrapper around the third-party `parsedatetime` package which does the actual guessing. It is needed because `parsedatetime` does not return Python datetime objects, but its own format, and most Python code works most easily with such objects.

3.4.7 view

`view` is just a package which holds highly reusable Flask components called Views. There is only one view at the moment:

3.4.7.1 account

Handles user registration and things like logging the user in immediately after they register, so they are "in" their account after hitting "Register".

3.4.8 suggest

The `suggest` package serves as a façade - if `import suggest` is specified, it bundles together functions from the underlying modules to expose functionality like
`from suggest import suggest_projects`
or (hypothetically)

`from suggest import suggest_news`.
In other words, the caller does not have to know the implementation details - they do not have to care about the fact that `gtr` and `hypothethical_module1` suggest projects, but `hypothethical_module2`
suggests news articles. Not having to know the names also helps to easily maintain backwards compatibility - if, for some reason, the `gtr` module is renamed to `gateway_to_research`, it will not break all the code which is importing the `suggest` package.

At the moment it *only* contains the `gtr` module, which can only suggest projects, so only `from suggest import suggest_projects` will actually work.

3.4.8.1 gtr

Contains the `suggest_projects` function which hits the GtR API and passes the results to the caller of FundFind's API (which in most cases will be its own user interface). It tries to catch certain error states like answers from the GtR API which are just invalid JSON, or HTTP codes other than 200 "OK" being returned by GtR. Its only argument is `similar_to`, which is a term that the returned GtR projects should be related to. When FundFind's user interface is using its `/suggest` or `/suggest_projects` route, this request parameter will be set to the user's search query, so the suggestions will be related to whatever the user is searching for.

It also contains a few convenience functions:

- `_make_error` wraps error messages issued by `suggest_projects` in a JSON object, so that responses coming from the FundFind API are consistent (not JSON most-of-the-time versus strings when there is an error).
- `_build_svc_url` takes several of the module-level constant defined in this module and builds a URL for `suggest_projects` to use.

3.5 fundingharvest

3.5.1 dao

The `dao` module here plays almost exactly the same role as in FundFind. In fact, the *only* difference is that it does not define an `Account` object since it does not have to deal with user accounts - it only defines `Funder` and `FundingOpportunity`, which it uses to submit harvested information to the index.

3.5.2 config

Config plays exactly the same role as in FundFind.

3.5.3 base

This module contains the `BaseHarvester` class which contains no functionality, but is there in case child classes need common functionality which happens to have the same implementation. Utility functions such as `check_results()` (ensure results being returned to the code have certain keys) are one example.

This is presently less useful in its current form since there is only one class which inherits from it - `RSSHarvester`. However, the initial design for the Funding Harvest project including

a `CLISubmitter` which "harvested" information from the command line (i.e. tested the whole idea).

It also used to include an `EmailHarvesterBase` which was meant to provide common functionality for trying to parse e-mails, like connecting to an IMAP server and fetching the right e-mail (last in reverse chronological order to match a certain subject line and possibly other header conditions). `EpsrcEmailHarvester` would then connect to it. The feature was unfortunately dropped since it turned out to be too difficult to implement in the remaining time/priority considerations. However, `fundfind.email.harvester@gmail.com` has been created and is receiving weekly alerts from the EPSRC.

Therefore, the inheritance structure has been kept, keeping "Future Work", §6.5 in mind.

3.5.4 rss

This module contains the `RSSHarvester` class which aims to provide common functionality such as `load_rss_feed()` to its children.

3.5.5 rcuk_esrc

This module contains the `EsrcRssHarvester` class which consumes the RSS feed of the ESRC (Economic and Social Research Council). This council was chosen since their RSS feed [19] actually takes arguments such as search terms and time periods. RSS feeds are not supposed to be used like that - they ought to only contain the latest items since a server-determined date, usually a week or so. This is a really interesting case of somebody using their RSS publishing capabilities as an ad-hoc API, even allowing users to do what is essentially search queries, but not having to code an API or a user interface on top - just relying on the user to input the URL they want and bookmark it in their RSS reader!

However, this detail also made this feed very easy to use for a prototype project such as FundFind - Funding Harvest would have had to be subscribed to other organisations' feeds for several weeks to get a good amount of useful data into the datastore.

3.6 User Interface

The user interface is pretty simple - two information submission forms and a search page (which doubles up as a "view records" page) as well as a "my account" page (which presently only allows viewing of details, not editing).

Designing a user interface usually revolves around two questions - what is being exposed (data, functionality) and how it is being exposed so that it can be best utilised by the users.

3.6.1 Content

The content or "what" to display was driven almost solely by what values the data had in general. Examples taken into account were EPSRC [28] and Wellcome Trust [86] pages and the RSS feeds of BBSRC bbsrc-rss and ESRC esrc-rss.

The process went as follows:

1. look at available data from the listed sources
2. change the funding opportunity submission page, adding newly observed fields

3. later on, look through the data and see what the search/view page is unable to show and enhance its knowledge of the data

3.6.2 Exposing The Content

Web user interface libraries package a lot of professional UX (User eXperience) designers' and developers' knowledge into an easy to reuse package. Bootstrap [69] is one such library and it was used together with jQuery [83] and jQuery UI [67] to make FundFind's user interface, including the responsive mobile UI characteristic.

These libraries package their UX knowledge into individual components which can be mixed together in a user interface. Those components still have to be brought together by the developer in a web page, however.

The fields in the information submission forms were just ordered in order of perceived importance, mostly as presented in the data sources mentioned above. Fields related to FundFind were placed at the bottom (e.g. tags) since the best way to write such metadata is to have thought about the values in the rest of the form.

3.6.2.1 Faceted Search

Faceted search [85] was chosen as a core concept to build the user interface on due to its ability to allow easy exploration of large semistructured datasets and quick drill-down to data the user is interested in.

At first, both the HTML user interface and the API were going to use this concept. However, faceted search is actually good for semistructured data because human minds are good at processing it. That is, as long as the data is analysed on the fly and common fields and values are presented, as well as basic frequency analysis (how many times does value Y appear in field X). At the Gateway to Research hackday it became obvious quite quickly that the developers present did not care that much for the "didYouMean" key in the results as much as the raw data - and the raw data could only be retrieved through a pagination mechanism. Thus, the otherwise nice feature which helps human users a lot was not used at all as developers tried to page through the GtR API to retrieve a wholesome dataset from it - a question of different priorities. That is why the idea of including facets in the API has been dropped for the moment, as it is best to start from what API clients would actually want - the most basic being just the data - and then trying to find what further features they might want.

"Facets" can be thought of as the various "columns" of a dataset if it was flattened out into a CSV file or a SQL database table. Each record in the dataset has a some value in each column, and blanks are allowed. The analysis / aggregation of the unique values in a column is called a "facet". A (simplified) example of what the data for this project might look like can be seen in Table 3.1.

Opportunity name	Subject	Country	Funder
Shadow manipulation in automatic understanding of 3D representations of real-world video footage	Computer Science	UK	EPSRC
Say hello! A global qualitative study of greeting phrases	Psychology	USA	APA
Fast-Fourier and the Three Musketeers	Computer Science	UK	BBC

Table 3.1: Example dataset for the purpose of illustrating what facets are

If we choose to construct facets from the Subject and Country fields, they would look like the ones illustrated by Table 3.2.

Facet name	Value	Number of items
Subject	Computer Science	2
	Psychology	1
Country	UK	2
	USA	1

Table 3.2: Example facets

Faceted search is often used for semi-structured data when producing an exhaustive classification is difficult since the data available in different records varies and there are many ways to classify it. One example is library catalogues like the ExLibris Primo software [31], where the artefacts the user could be looking for vary from journal articles through manuscripts to books, span various fields and are grouped by subject - very much like funding opportunities can be.

3.7 API

The API was implemented by using Flask's capabilities to declare handling of HTTP GET and POST in an easy way, described in "web", §3.4.3 and by detecting whether the request wants JSON format. If not, the HTML user interface is served.

The routes which form the most important parts of the API and which would actually be useful are `/describe_funder`, `/share_fundopp`, `/slugify`, `/suggest` and `/suggest/projects`, described above.

3.7.1 Data Format

JSON was used as the API's main data format mainly due to interoperability reasons - it is extremely simple and as such has support in most major programming languages. Furthermore, if it assumed that other Open Knowledge-related projects are most likely to use FundFind's API, then the fact that a lot of them are written in Python (as mentioned in "Design Details and Rationale", §3.1.1) plays a role. Python has excellent support for JSON through a core library and more importantly, its most popular internal datastructures - dictionaries and lists - resemble JSON perfectly, making it easier to read and "digest" for Python programmers.

Furthermore, FundFind uses JSON itself when communicating with its elasticsearch datastore, again via an API, and there was not much reason to introduce another transport medium at this stage of a prototype project.

Chapter 4

Implementation

Instead of going through the minute details of how the architecture described in Chapter 3 is implemented, this chapter will focus on how the functionality for the final software features was achieved. Common topics are tackled first: a lower-level overview of the applications' architecture and interaction with the datastore.

4.1 Overview of the Applications

4.1.1 FundFind

```
|-- config.py
|-- core.py
|-- dao.py
|-- default_settings.py
|-- importer.py
|-- __init__.py
|-- static
|   |-- css
|   |   |-- fundfind.css
|   |-- js
|       |-- fundfind.js
|   |-- vendor
|       |-- bootstrap-2.0.0
|       |-- facetview
|       |-- jquery-ui-1.8.18.custom
|       |-- linkify
|-- suggest
|   |-- gtr.py
|   |-- __init__.py
|-- templates
|   |-- account
|   |   |-- login.html
|   |   |-- register.html
|   |   |-- view.html
|   |-- base.html
|   |-- describe_funder.html
```

```
|   |-- facetview.html
|   |-- _formhelpers.html
|   |-- home
|   |   |-- content.html
|   |   |-- index.html
|   |-- search.html
|   |-- share_fundopp.html
|-- util.py
|-- view
|   |-- account.py
|   |-- __init__.py
|-- web.py
```

The `.py` files at the root are python code files. `__init__.py` makes directories into actual Python packages (see "Organising Code in Python", §3.2.1). Modules in Python assume the name of the Python file without the `.py` extension, so `web.py` is, by definition, the `web` described in "web", §3.4.3. These files contain the code which integrates together the third-party code listed in Appendix A to achieve all of FundFind's functionality.

The `web` module uses the Flask web framework templating capabilities to load the files under the `templates` directory when the routes listed in "web", §3.4.3 are requested by a browser in HTML format (the templates are not used when building JSON responses for the machine-friendly API). `base.html` is referenced by every other template file and includes the Bootstrap, jQuery and jQuery UI web user interface libraries, as well as FundFind-specific CSS styles and Javascript functionality from `static/css/fundfind.css` and `static/js/fundfind.js`. The templates are written in the Jinja2 templating language [3], a flexible templating engine which comes bundled with Flask.

`static/vendor/` contains *all* non-Python third-party code - everything listed in Appendix A as used by FundFind which is *not* a Python package (the Appendix specifies this). This neat separation of code is actually just a convention used by Flask-based web applications. The reason Python packages are excluded is because they are supposed to be installed elsewhere in the system, so that they are usable by all Python software on a system, not just this application. In practice, this works a bit differently and packages are often bundled with their applications to ensure that only the intended version is used, but they are nonetheless not contained within the application's source directory, but in a "virtual environment" which also contains the application's source directory. There is no chance of confusing third-party Python code with the application's code since none of it is actually present in the application's directory - the application modules `import` packages that they need to use (usually at the top of the file) and then just call the appropriate functions.

Python does support a way of renaming imported functionality to more convenient names and importing only parts of third-party code (e.g. a single class or function from a whole module). In that case, it is not necessary to prefix the call to the third-party code with the third-party package name. For example:

`from json import loads` at the top of the file will allow use of
`loads(turn_this_string_into_json)`
anywhere in the code of that module. FundFind imports its own modules from the main `fundfind` package in a similar manner, so that it does not have to say
`fundfind.some_module.some_function()`
everywhere but just `some_module.some_function`. This is not used to obfuscate any third-party functionality and present it as FundFind's anywhere in the code, and `from`-style import

statements which refer to third-party code are all located at the top of their respective modules, to avoid any chance of confusion. (They may be used inline for better readability, but only for FundFind's own modules.)

4.1.2 Funding Harvest

```
|-- base.py
|-- config.py
|-- dao.py
|-- __init__.py
|-- rss.py
|-- rcuk_esrc.py
```

Since this application only has a command-line user interface (the `base` module serves as the runnable entry point), it does not include web UI libraries. It does use some third-party Python packages and built-in Python packages, described in Appendix A, but calls to these are quite obvious, as discussed above.

4.2 Datastore Access

The `dao` module merits further discussion.

When the `upsert` method generates `id`s for the incoming data, it currently simply generates UUID4 (random unique identifiers). "dao", §3.4.2 uses `Account` to illustrate the point that `id` generation could be overridden on a per-DAO-class basis. In the case of `Account` they will actually usually be specified (the users choose a username and that is used as the `id`), so a random number will *not* be generated.

The `pyes` library was used in the `dao` module to access the elasticsearch datastore. It converts the datastore's HTTP responses into Python objects, providing native shortcuts to the rather extensive elasticsearch API. It also does this for the opposite FundFind / Funding Harvest use it for their submissions to elasticsearch.

4.3 Implementing the Main Application

4.3.1 The Basics

The one crowdsourcing basic function which has to be present in a project which hopes to get any input from its audience is submission of information. One step further is allowing users to view the submitted content. These were thought to be quite easy to achieve at the beginning of the project, which turned out to be possibly the greatest miscalculation in the whole piece of work.

4.3.1.1 Search for Scholarly Funding Opportunities

The combined search/view page is an important part of the user interface, allowing the user to drill down through (potentially) a lot of semistructured data to find what they want.

The facetview project is a pure Javascript faceted search interface to elasticsearch which incorporates the faceted search concept described in "Faceted Search", §3.6.2.1. It is a jQuery plugin which has configurable initilisation. The general idea is that it will generate and put all its content inside an HTML `<div>` element identified by a certain `class` or `id` attribute (the details are up to the developer).

In the case of this page, the effort was mostly configuring facetview to show all the fields the data had and surround the content of those fields with appropriate formatting. For example, the name of the field (e.g. "Homepage") would be underlined in contrast to the content of the field. Fields also (usually) need to sit on separate lines and facetview had to be configured to put them in separate `<div>`s since these are block HTML elements.

By default facetview would query absolutely everything in the index, which meant that user accounts were showing up together with funding opportunities and funders. On the other hand, restricting it to show only one document type would not work either, since both funders and funding opportunities are of interest to the users (who can then refine the results they are seeing). Thus, a particular facetview option was used - predefined filters. This filters the results to just funders and funding opportunities (and the user is free to then further filter them to just one of these types).

One neat feature of facetview, developed recently at the Gateway to Research hackday, was that of being able to share a search simply by copying the URL of the search page. When a search is executed (every time the user stops typing in the search box), the URL of the page changes to include the user's query. Sharing information for a funder and related opportunities can thus be done just by typing in the name, e.g. "wellcome trust"

On top of the configuration, there was a significant problem with actually using facetview in the fundfind codebase. The parent IDFind codebase was still using an older version (1.3) of the Bootstrap HTML user interface library. Facetview was using an incompatible 2.x version. This was known before the mid-project demonstration, but took so long to fix that the search page had to be demonstrated using an iframe (embedding another page inside it), which is a big no-no in the modern user experience field. (One explanation puts it really well: "They break the one-document-per-URL paradigm, which is essential for the proper functioning of the web (think bookmarks, deep-links, search engines, ...)." [75].)

Furthermore, facetview was expected to be far easier to configure than it was. Using a component and even seeing development being done on a component clearly do not equate to being able to use said component in software - the author had not actually tried facetview integration, although it *looked* easy.

4.3.1.2 Submitting Information about Funders and Opportunities

The information submission pages were made manually in the parent IDFind project. There is, in fact, a library which will generate forms and can be configured to validate them - WTForms [88]. Unfortunately, this was discovered a little too late - the submission pages had been written and time had to be divided between other features.

An interesting elasticsearch feature helped here. It is possible to ask elasticsearch what fields the incoming documents contained by issuing an HTTP `GET` request (e.g. with a browser) to the `_mapping` path of an index, like this:
`http://localhost:9200/fundfind/_mapping`

which promptly answers with something similar to:

```
{
"fundfind": { # index name
"funder": { # types of submitted documents
"properties": {
"created": {
"type": "date",
"format": "dateOptionalTime"
```

```
},
"description": {
"type": "string"
},
"homepage": {
"type": "string"
},
[...]
```

This lists all the properties that all documents submitted to the `fundfind` index have had and how they were analysed (e.g. date versus string), grouped by document type. Not all documents need to have all the properties of course - this is just a list of all the properties that elasticsearch has had to analyse. Which is all that is necessary for configuring facetview, since it can deal with semistructured data with missing fields in it just like the datastore can.

jQuery was used to create the "add more" buttons on the information submission pages. Adding more useful links adds more text input fields, while adding more tags extends the tags input field. Processing the extra `useful_links` in the `importer` module was an interesting challenge - at first, new elements created using jQuery were given numbers at the end of their names, like `useful_link1`, `useful_link2` and so on. The back-end code then iterated through a loop a 100 times, trying to get content from these elements (which would of course break if anybody ever submitted more than 100 useful links - it would discard everything above a 100). Then it turned out it is possible to declare the `input` element's name attribute as follows: `name="useful_links[]` (note the square brackets). Apparently, this is supposed to signal to the web server that this is an incoming list of values. So Flask's underlying webserver, Werkzeug, provided the `getlist()` function on the HTTP request object, to deal with this situation.

FundFind does `request.values.getlist('useful_links[]')` and then iterates over the Python list this returns in a conventional Python way `for link in list`, instead of using integers on a list of an unknown size. It should be noted this feature did not change much in FundFind, although it was written by the same author for the parent IDFind project.

4.3.1.3 Create Profiles

Profiles are handled almost entirely by the Flask-Login extension, except for registration of accounts. [82]

This uses the Flask views described in "web", §3.4.3, Flask blueprints and WTForms to create several actions - registering a user, logging in and logging out. Flask blueprints are a way of sharing common operations, such as rendering a particular template and processing a particular form - if another application wanted to use FundFind's user registration form, it would just need the blueprint file and would need to call `app.register_blueprint()` in their equivalent of the `web` module.

4.3.2 Advanced Features

4.3.2.1 Harvest Funding Data From Machine-Readable Sources

This was implemented using the Feedparser [40] library. After loading the feeds, a result object (just a dictionary of key-value pairs) is built up from the feed data and some mapping of keys is done, e.g. "name" gets renamed to "title".

Feedparser behaved quite well, following the Python convention of "it just works" i.e. sensible default settings out of the box and supporting the widest possible range of inputs (feed formats like RSS, Atom etc. in this case). Its documentation does contain a pretty ominous sentence:

> Values are returned as Python Unicode strings (except when they're not - see Character Encoding Detection for all the gory details).
>
> *Feedparser documentation [79]*

The implications of this have not been considered due to time constraints and Funding Harvest itself does not care about the values, but the pyes library it is using to connect to the datastore might actually mind.

4.3.2.2 Tagging

Both "Tagging Research as "Potentially of Interest to" Users And Groups" and "Specify Affiliation and Interests" fall under this category. These were actually pretty easy - all that was necessary was to let users put in a comma-separated string. The string was then parsed and cleaned up, first by chunking it up into a list using Python's `.split(",")` string method and the `util.clean_list()` method described in "importer", §3.4.4. The "add more" button on the interface was added as described above in "Submitting Information about Funders and Opportunities", §4.3.1.2.

4.3.2.3 User Interface And Responsive Mobile Web Design

Initially, this was done in a very simple way - just using Bootstrap 1.3, an old version of Twitter's Bootstrap library. The major upgrade pains described above were essentially integration difficulties between facetview and the old IDFind user interface, caused precisely by the age of the version of this library.

Thus, it was upgraded to Bootstrap 2.3 and a the old IDFind HTML templates were changed, with the hope of simply using its mobile-friendly version (so a question of loading the right files). The Flask-Bootstrap extension was used, with the goal of encapsulating best practice when it comes to structuring the HTML pages (the developer is supposed to inherit from its templates). Apparently, one is supposed to put Javascript components at the bottom of the HTML page, so the rest of the content can load - if they are in `<head>` the browser *must* use blocking, synchronous loading and wait for them to arrive before doing anything else.

However, Bootstrap 2.3 broke facetview's layout. It's usually a simple fix, but in this case the HTML of the search box, options, filters and so on is actually generated inside facetview itself, which is pure Javascript and jQuery (a Javascript library). It is also one big file, so it was quite difficult to find out where the layout generation was actually being done. It turned out to be easier to *downgrade* Bootstrap to version 2.0.1, which what facetview has been using for the past 9 months or so. Luckily, this did not break FundFind Bootstrap v.2.3-compatible layout, and all the pages finally worked (including on mobiles). However, Flask-Bootstrap did not have a release for Bootstrap 2.0.1, so it was stripped out, and Bootstrap was included manually in the `<head>` of `base.html` (so in all pages).

4.3.2.4 RESTful API

FundFind detects whether the request wants a JSON API response or the HTML user interface by looking at the request URL - if it ends with `'.json'`, it serves a JSON response (if the route

supports it - e.g. the homepage at `'/'` does not serve JSON responses and will serve HTML anyway if asked).

4.3.2.5 Suggesting Relevant Historical Data

This was going to be coded as pure Javascript since all that is really needed is an AJAX call to the Gateway to Research API, which will happily return JSON. However, one important characteristic of AJAX was overlooked - cross-domain requests are not allowed. The browser simply would not make the AJAX call to `gtr.rcuk.ac.uk` from `localhost` or even `fundfind.cottagelabs.com`, for very good and widely accepted security reasons.

Thus, a server-side `suggest` package and a `gtr` module became necessary.

The `GTRConsumer` class uses HTTP GET requests to communicate with the Gateway to Research API. Its `suggest` method just uses the API's `/search` HTTP GET route to look for projects which contain the user's query to FundFind.

It does not currently process the query in any way, just asks the GtR API to give it back everything it thinks contains a particular string. The GtR API may actually do clever processing and include "similar results" in the resultset - it does not currently do that, but is in active development and may well change.

Chapter 5

Testing

5.1 Overall Approach to Testing

Usually, tests can be inherited alongside features from parent codebases. However, IDFind had no automated tests. This was a factor in some decisions, such as dropping certain features where limited development time is mentioned. A good example is IDFind's Twitter integration mentioned in "Changes Over Time", §3.2.2. Writing any sensible test coverage for an external service *from scratch* would have taken up valuable time which was used for more important features.

Most of the tests were manual. User interface testing (with the exception of the usual develop-test-develop feedback loop). It turned out Test-Driven Development can only be applied when a certain level of knowledge of the target domain and the technology is readily available - otherwise there is simply not enough information about how the application is supposed to function in order to write the tests first. Thus, the conventional development loop was used (develop-test).

The Funding Harvest project is not tested automatically in any way. In the event that it actually finds, harvests and submits something to the index, this will be immediately visible on FundFind's user interface, without even reloading the page (as searches are executed after the user types in a query).

5.2 Automated Testing

Insufficient familiarity with the technologies involved resulted in a lot of time being spent on the exploratory phase of just getting things to work together, or copying a working example and modifying it to suit the project's needs. These are before testability can be achieved, where there is at least basic, but solid, knowledge of the technologies involved, the format of responses, return values etcetera. Furthermore, IDFind left no tests whatsoever.

Thus, it was best to proceed with writing at least some integration tests which would cover all API routes (`/describe_funder`, `/share_fundopp`, `/slugify`, `/suggest` and `/suggest/projects`). These would ensure that what is supposed to be possible for a machine to do (that is why it's an API after all) is actually possible, as well as doubling up as small sample programs on how to the API, what the available routes are, the format of the responses etcetera. Covering these routes also had the side effect of putting 90% of the code to work, thus greatly increasing the possibility of discovering defects even without unit tests. As automated tests, they are of course repeatable, in the sense that they can showcase regressions in quality just as quickly as unit tests, since they are never supposed to break.

5.2.1 Integration Tests

Integration tests were entirely separate from the codebase in the sense that they did not directly import any of the code being tested. They targeted the main API routes (listed above and described in "web", §3.4.3) and as such, mostly provided coverage for the otherwise untested `web` module, while also testing whether it would properly call upon everything else in the system, and thus also testing whether everything else would do its job to finally serve appropriate results. The integration tests also test the content negotiation of the API, e.g. whether JSON is being served when requested.

The tests are quite simple - they take some sample data such as

```
funder = {
    "description": "We are a global charitable [...]",
    "tags": [
        "healthcare",
        "medical",
        "biology",
        "support open access"
    ],
    "owner": "emanuil",
    "id": "3202f106e8894f08b49eb69d41863e88",
    "name": "Wellcome Trust",
    "created": "2013-02-08T10:38:29.273696",
    "homepage": "http://www.wellcome.ac.uk/",
    "modified": "2013-02-08T10:38:29.273719",
    "useful_links": [
        "http://www.wellcome.ac.uk/About-us/Jobs/[...]"
    ],
    "policies":
     "http://www.wellcome.ac.uk/About-us/Policy[...]",
    "interested_in": "primarily healthcare, [...]"
}
```

and HTTP POST it to the API routes. Those should respond with a JSON object with results, and this is checked by the tests. If python's `json` package fails to parse the result, then it is not valid JSON.

After that very basic test is passed, certain values in the response are checked (e.g. whether `/slugify` returns a proper slug). If the test is testing data submission, then the elasticsearch datastore will return an appropriate result containing `"ok": true` and the tests look for this.

5.2.2 User Interface Testing

User interface testing was performed manually each time the interface was changed. Selenium IDE [84] could be used to automate this testing, but testing the Javascript part (i.e. facetview) would require a quarter of the hours allocated to the software development part of this project in the first place, even if only due to lack of familiarity with the Selenium framework. facetview does not come with any tests whatsoever.

5.2.3 Stress Testing

Stress testing could be done using the multi-mechanize [6] suite. However, time constraints and the low priority of such tests preclude this for the moment, although the suite was investigated.

5.3 Responsive Mobile Web Design

The application was not tested on different devices, but relied on the Bootstrap framework to perform as expected of such a popular user interface framework, which it will probably do better than the less-than-experienced in responsive web design author. It was tested on a 2013 HTC Desire X device running the latest version of mobile Firefox. The device has a 4in. screen and runs a relatively recent version of the Android mobile operating system - 4.0, and is as such not particularly representative of the mobile device market.

Because of that, the application was also tested using a free service called "MobileTest.me" [74], which essentially does little but simulate different screen sizes (as well a few other minor tweaks). This is enough to trigger the responsive design built using Bootstrap however, and several deficiencies in the mobile user interface were caught this way, before performing a final test on the developer's personal mobile device.

Chapter 6

Evaluation

6.1 Approaching The Field Of Scholarly Funding

Somewhere in February, the 4th (out of 6) month of the project, I realised that this domain of knowledge, scholarly funding, was going to be a tougher nut to crack than I had thought.

One thing that I should have done earlier is specified that actually trying to understand this sector beyond the needs of the software was going to be a requirement of the project. Requirements gathering as a formalised goal had plenty of precedent. In January, I came to know of the G4HE project which tries to "engage with [...] Gateway to Research [...] to improve the information exchange between Higher Education Institutions and the Research Councils", which is all about funding data. The difficulties only became clear once the technical work started in earnest and it turned out it was in fact both a pretty big field, and one that was ripe with problems and conflicting interests. Now, the requirements I managed to gather and the funding sector are discussed in this write-up (Chapters 1 and 2), but they could have produced far more output, e.g. a series of blog posts covering each issue in some detail. Furthermore, the G4HE project has far more resources than I have for essentially trying to do the same thing - how funding data can be useful to HEI-s and the people inside them - and a more structured approach on my part might have allowed me to push them for some of their outputs or to contribute my efforts.

Relating information about the field and its problems to people outside the target audience and even across target user groups turned out to be much more difficult than expected.

6.1.1 Condensing The Insights Gained

Part of the point of the project was to see how the funding field worked. In terms of actually achieving understanding this went fine, I now know a lot more about it than when I started. However, sharing this information seems really difficult. It feels like my sample was way too small and that any representation I choose for it will just be laughably bad. In reality, there is clearly a place in the world for a "how does scholarly funding work for beginners, from a beginner" collection of information, but presenting this appropriately seems incredibly difficult.

All the insights gained were attained through interviews. However, it was just my opinion of the content of these meetings that was actually going to make it into the final output. This can feel quite daunting - as a developer, I prefer to publish the data and let those whose work I support actually "deal" with it. However, in this case, I *am* the researcher, this mysteriously skilled person who somehow knows what to do with this data. Others do not want to know about conversations I have had, they want knowledge - even understanding.

However, I have done similar things before - I talked about what Open Knowledge is while I was still figuring it out, so the analysis presented in "Audience and High-Level Requirements", §1.4 will be turned into a blog post. It seemed like a waste to spend so much time thinking about what could be useful in the funding field and not sharing it, even if the outcome was far from conclusive. My previous attempt to tackle a new field is published on the Cottage Labs website [24], and we do publish thoughts which concern the Higher Education field, so this is probably where it will end up as well.

6.2 Collecting Information from Disparate Sources - Technological Suitability

6.2.1 Working with the Datastore

Using an indexing server as the *main* datastore for everything in the application is actually a little unconventional - usually, such software is used to analyse or store a big chunk of semi-structured data that the application has to deal with, e.g. a web crawler might throw HTML documents in there. Accounts and information coming in from HTML forms are certainly a secondary use case for this type of software.

The nature of an indexing server is that it places much less constraints on the data than conventional datastores. That is actually precisely where its power comes from - it is still able to search through a large amount of data *without* placing datatype and other constraints on it, in effect shifting the burden of processing the data to the datastore.

However, this flexibility comes with a small price to pay - it can be difficult to guarantee the "uniqueness" of a certain set of fields. Unless `id` is being used to refer to a particular document in the index, *every* query can return multiple results - there is no concept similar to that of primary keys. This is not a big problem, but can be challenging to the mindset of a developer used to having much finer control over what the data is, and it does affect the way the code is written (assume every query result is a list of results).

After working with elasticsearch on a project as long as this, I really feel that it forces you to deal with the data instead of relying on arbitrarily constraining it (which now reminds me of "IT Department said we can't do this"). I am sure the indexer can probably be configured to support data constraints, but going with the default functionality and letting it guide development makes for low coupling between the structure of the data and the FundFind software.

To put this low coupling into context, let us assume that the Open Knowledge Foundation has decided to hold a hackday where funding data sources are found and consumed (websites: scraped, API-s: consumed, CSV-like files: loaded). All they would need is a single A4 page stating:

- URL of FundFind's current elasticsearch instance
- field name conventions for `funding_opportunity` from "FundFind Datastore Details", §3.3.1
- how to chuck data into elasticsearch using its HTTP API. HTTP is well known and most API-s use it as a transport medium, so it is very well supported by libraries in modern programming languages. Developers who are not familiar with it can be taught easily (everybody has used a web browser) and the others just need to know the endpoints to submit data to (e.g. `POST` to `http://test.cottagelabs.com:9200/fundfind/funding_opportunity/`).

In other words, there is no need for them to know how FundFind works, or even what it is, in order to be able to contribute data and scraper/parser/api client code. With a relational database, they would need to find a client/driver to connect to the database in their favourite programming language. The schema itself would be more skewed towards FundFind's specific retrieval and storage needs.

This principle of low*er* coupling is essentially what the Funding Harvest project demonstrates.

Working with elasticsearch has been one of my favourite parts of this project. It is really lightweight and plugin development can allow for really interesting applications. One potential idea which came up during the Gateway to Research hackday includes outputting arbitrary data into a graph structure, so that visualisation libraries such as graphview [41] can work on it immediately, thus seriously reducing the time needed to produce useful, insightful visualisations.

Another idea that I had while working on FundFind and another open-source project, OpenArticleGauge [54] is writing an "input" plugin for elasticsearch so that it can index images in some fashion. Images and algorithms for analysing them vary greatly depending on the aims of analysis, so I looked towards medical images - looking for images of symptoms described by incoming data (either text, or another image). I'm pretty sure it's one of the most algorithmically difficult practical problems I can currently imagine, and there are existing solutions which do medical image search. I would be curious to see if some of their limitations might not be lifted by using elasticsearch to do the search (handle applying the algorithm and collecting the results), thus really focusing on the indexing / "understanding" algorithm itself, thus leading to a better solution.

6.3 User Needs

6.3.1 Identifying Requirements

Far too much thought seems to have been given to the priority of features - at the end it's just a list of functions and they're all desirable. There was not a single feature which was later identified as a "bad" idea. Of course, they can be roughly grouped according to what users seemed to want and this is very important in guiding future development, but in hindsight, there does not seem to be much value in the detailed prioritisation presented in "Initial Requirements", §2.4.2. With rough groups such as "vital", "desirable", "low priority", the prioritisation process could be simplified significantly and there would be relatively few features that would be difficult to categorise.

It was still sensible to categorise new features which came up during development, since the feature list was not really fixed until development had ended.

6.3.2 Meeting Needs

It turned out that data was far more important to users than the other features. This was known from the first user meetings, but I had assumed I would be able to finish the software features *and* focus on data gathering at the same time. Users didn't really get the other features without data though - a faceted search over an empty list is no good. That is why gathering data was finally prioritised after the basics were up and running.

6.3.2.1 Exploring Scholarly Funding

Within the group of people with analytical needs, there was a sub-group of developers which inspired and was part of the reason the project was conceived - Cottage Labs LLP, the Open Knowledge Foundation and the author. It was always the project's intention to learn about scholarly

funding specifically so that these people could have a better idea of how to meet the Higher Education sector's needs. This went relatively well, the feature list presented in "Initial Requirements", §2.4.2 is certainly a very good start and gives space for a lot more development in the future.

The project was not presented to the Open Knowledge Foundation discussion list due to insufficient polish on the software side. Writing that first e-mail is quite important for gathering potential collaborators or even getting any feedback at all, so I felt it should at least have a solid base. On the other hand, developing for a long time without wider feedback from one of the target audience groups is probably not a good idea, as a lot of effort could end up wasted pursuing the wrong direction. A balance has to be struck between using other people's time and using my own, in order to eventually come up with a good product.

6.4 In Retrospect

Basing the project on another one was obviously a big decision. In retrospect, I should have been more careful - it did not have any tests! I cut out so much of it that in the end it was just the fact it was a Flask application which integrated the Flask-Login extension that saved any time at all. I'm sure it's not the only one that does it though, and identifying that this is what I actually wanted as a base would have helped immensely in obtaining a better (and tested) parent codebase. On the other hand it is good I didn't start trying to learn how to develop a web application in Python/Flask from scratch, since I was familiar with projects that used the combination, but hadn't written one from scratch before - which would have probably resulted in even less features making the final cut.

The Open Knowledge and funding contexts forced quite a lot of reading (20+ hours) on top of what I already knew about the fields. Trying to figure out how the two fields intersect or should intersect is not something I accounted for in the initial time budget, but I am not sure I could have avoided it. I would try to focus more on the core features if I had the chance to do this again - no matter how lacking my understanding is, there is no end to learning more. The current understanding should have (and did) informed some sort of software development method and some feature list, just focus on delivering and evaluating. Do not cut the cycle short when new ideas come about or the flaws are exposed in old ones.

Another issue was pure technical skill. Upgrading a user interface library (Bootstrap) in order to integrate a component you want (facetview) would have taken the people I have worked with on hackdays etc. a lot less time than it took me. Clearly, observing how rapid development is done doesn't guarantee the ability to do it - there is no substitute for experience. This is particularly obvious if the original feature timeline is taken into account - how did I ever think I was going to be able to implement one feature a week? Probably possible - if I did absolutely nothing else most weeks!

This, bad time management due to inaccurate estimates, is actually another issue. I did not do these underestimations (reading time and personal skill) just in the context of the project, I applied them consistently to other university work and extracurricular work. Other assignments often took priority when they came up, since I thought I could do them in a relatively small amount of time, tick them off my to-do list and focus on FundFind. I do not recall a case when I did not underestimate the time needed at least by 33%, although it started getting better towards the end of the semester.

I was clearly over capacity for a prolonged period of time, taking on some development work for the OpenArticleGauge project alongside the university work (15 days actually). It tries to see if a particular scholarly publisher's articles are open-access by detecting the license statements on

publisher pages - all that just given a DOI (Digital Object Identifier, e.g. of an article of interest to the OAG user). It actually informed the RSS harvesting bit and helped with more Python knowledge, but no matter how interesting it was, fitting all the other university work, FundFind and OpenArticleGauge together was not possible. Perhaps possible for somebody with a little more technical skill and significantly more work management / focus abilities than me, but for me it was just wishful thinking. Saying "no" to interesting work is certainly a new concept for me - undergraduate years are usually spent trying to get any sort of useful real-world work done - but is obviously necessary for quality output.

6.5 Future Work

6.5.0.2 E-mail Alerts

This is a good feature and it's great that it was identified, especially since the author was not previously aware of the current scholars' workflow. This will make use of tag information which is currently just being collected. It is probably best implemented as yet another small application which communicates with the datastore and runs a few complex queries. It could work as follows. For each funding opportunity:

1. maintain a datetime marker field called `email_alerts_last_processed` or similar
2. let elasticsearch search for all users whose research interests match the opportunity's tags (at least one value in each list is the same)
3. send an e-mail to the users returned by the query, who have not already had an e-mail about that opportunity

"Who have not already had an e-mail" is what actually caused this to go to the Future Work section at the design stage. A list of users who have already heard about this could be maintained, but then date windows (only opportunities new in the last week) should probably be introduced, at which point the feature becomes quite complex. facetview, the search interface, can actually generate elasticsearch queries, so in the technical sense the feature could be simplified slightly by reusing those capabilities - but reusing the Javascript code would probably not be trivial, even using software which is designed to run Javascript server-side like Node.js .

6.5.0.3 Exporting the Results of Searches

The need of research development officers to do reports of information they have submitted has been mentioned in "Audience and High-Level Requirements", §1.4. This feature would fit the bill perfectly, although it would have to be combined with the ability to share information only with certain users/groups of users, since I did not get the impression that research officers would just like to collectively share all their funding data.

6.5.0.4 More Advanced Feedback Feature

The ability to submit feedback to the project. This would only be useful as a software feature if it actually used some of the context, i.e. it allowed a user to capture an error in the software more easily or made error reporting faster or more visual. The comparison is with simple email and the GitHub issue tracker, and there is no point coding something which will do no better than these two.

6.5.0.5 Using The Tagging Information

Currently, tags (including research interests) are derived from the comma-separated strings that users enter. However, they're not used in the system in any way - allowing users to browse by tags would be a matter of understanding how to configure facetview to merge together tag values from different items, sort them by frequency and allow the user to filter the search results by any tag value.

E-mail alerts would also have to make use of this information. The

6.5.0.6 Better User Profiles

Right now, there is no way to delete your profile or edit the information provided at registration. This was only left as-is due to the fact that you cannot do much with the information provided (e.g. research interests are collected, but not used, as is affiliation data). As both of these change, editing the profile data will become quite important.

6.5.0.7 RESTful API

The API could be extended to reply in other data formats, such as XML - there will be projects who are already using XML despite JSON's simplicity, so this will only further the interoperability goal.

Furthermore, proper content negotiation usually includes more support than just tacking `'.json'` at the end of the request URL - the HTTP Accept header is, apparently, the canonical way of doing this (there do not seem to be many sources describing this as fact beyond a few opinion pieces on programming blogs, but the Apache web server documentation says it is part of the HTTP 1.1 specification [2]). There are python libraries which might help, although they do look like they are in the alpha stage at the time of writing [53] [59].

Furthermore, not all of FundFind's functionality is exposed via the API - the suggestion functionality might be quite useful to developers. The authors of a hypothetical "News for Scholars" newsfeed reader application might want to suggest previously funded bids to its UK readers, and FundFind can easily be adapted to serve as the server-side back-end of such an application, complete with the suggest functionality. Of course, right now it adds little value on top of Gateway to Research's own API, but if more data sources are added, this becomes more critical to FundFind's outlook as a modern open web application which exposes its useful functionality.

6.5.0.8 Suggesting Relevant Historical Data

Clearly, more data sources besides the Gateway to Research API could be added. In fact, the ESRC's ad-hoc API in the form of their RSS feed described in "rcuk_esrc", §3.5.5 is a prime candidate for this. Relevant news articles (e.g. from The Higher Education Times) could also be added.

Beyond more and more data, one very important and yet minor change would be using the funding opportunity's title to suggest similar successful bids, instead of just the user's query. It would probably be necessary to enhance the user interface for viewing individual funding opportunities first since there just wouldn't be enough screen real estate to have multiple suggestion categories on the side of the current search results page.

Furthermore, the suggestion algorithm itself can be improved. Instead of dumping everything coming in from the data sources, it could tokenise (break it up into words) the suggestion source - user's query or funding opportunity title. Then words such as "the" and "and" could be removed, as well as other commonly used English words, giving more weight to domain-specific keywords.

Currently, Mark MacGillivray [11] is working on something similar while trying to analyse a person's scientific output, but alas, neither his PhD nor the associated software are published yet.

One of the best outcomes of this project was successfully using the Gateway to Research API, which actually returns really interesting results. Only of the resource types is used however - "project". They also have people and organisations, and the data is stored in a datastore not dissimilar to elasticsearch (Apache Lucene). Suggesting "people" relevant to a search query from *previous funding data* would yield some really interesting results, although Lucene's advanced searching capabilities or a data dump indexed into elasticsearch might have to be used to actually achieve this.

6.6 Learning

This project certainly pushed the boundaries of what I could do, both technically and in terms of thinking about the wider issues around technology.

Technically, my programming skills in Python have improved, with a much better idea of what is "pythonic", and even better - an understanding that it doesn't necessarily matter, if the software does what it is supposed to do. A balance needs to be found - a prototype cannot be written perfectly and there is little reason to waste precious time on doing so. This type of inflexibility is exactly why Agile approaches were developed, and now eschew "big up-front design" in most cases. I seem to go after technical implementation when I do not know what else to focus on. Having clean and maintainable code is certainly not only good, but a requirement in larger, more mature codebases. However, it is pointless when the code will be thrown away 10 minutes later because "I found a component in my favourite UI library which does this far better than me". Software projects certainly differ in their goals and context, and these differences need to be taken into account when writing code and picking a methodology. I judge the Python gain to be particularly valuable since a lot of Open Knowledge projects use it as previously noted, and I intend to work in that field.

On the other hand striving to write code "correctly" also means reusing widely tested and used components, and to that end, I have certainly gotten better at building HTML/CSS/unobtrusive Javascript user interfaces.

Vim, the command-line editor mentioned in "Technologies Used", §2.6 was something that I probably needed to learn, especially for working in server environments. I am glad that I stuck with it, text editing is now slightly faster than with a conventional editor, even with my basic vim skills. It was probably not such a great idea to try to apply it to every single project during the year including FundFind, however. It didn't really contribute that much of a delay to FundFind, but it did impact other work at the beginning of the year, therefore impacting FundFind - there was no need to further complicate an already difficult work situation.

The scholarly funding field has been an interesting challenge to comprehend, as noted above. This will certainly be useful - a lot of the current Open Knowledge field is dedicated to serving the needs of the academic community and being acquainted more closely with that community is most beneficial. Having looked at funding information targeted at all levels of the academic's career will also come in handy if I myself decide to become a scholar.

A related, albeit much more narrowly focused learning experience has been typesetting this document in LaTeX. Clearly this is not as useful in industry, but it would be in academia. I had typeset documents in LaTeXbefore, but learned a lot from the template provided for this write-up, as well as having successfully defined my own reference command to include the name of the target section. Dividing the focus of writing using separate chapter files and the TeXCount [73] script are

examples of further useful experience.

6.6.1 Applying Openness

Lastly, a very important lesson (towards which FundFind greatly contributed through trying to be an Open Knowledge project) was that Openness is not just a label that can be slapped over anything, especially technology, to magically make it better. When talking about "Open", the benefits that it will bring - or at least the reasons it is the "right" course - have to extremely clear. This came from trying to explain to potential users or friends what the benefits of opening u[funding data would be.

Open culture and the associated ambiance have been steadily gaining traction and seem appealing to a wide audience. A significant section of this audience is technologists and scientists, although the uptake by developers in industry seems greater than that by scholars in the hard sciences. In those circles, there are many who feel that openness must be good in some way, but major new ideas in the openness field seem to come from a small percentage of dedicated individuals. They seem to have a very well-defined idea of how an open approach can help other fields and then set about showing that to others. Examples include those who drive the Creative Commons organisation [17], the Open Knowledge Foundation's Rufus Pollock [62], the Free Software Foundation's Richard Stallman [60].

On the other hand, the openness aspect of computer science is now explored by thousands of developers world-wide. Understanding why they (seemingly) readily accept that sharing one's work openly is beneficial and the ultimately correct thing to do seems key to understanding why scholars do not necessarily share information as readily. This is mostly of relevance to the Open Access field, which focuses on research output. However, why would a scholar *submit* any information to FundFind? Why would they *share* on FundFind - their time, accumulated experience, effort, even just list of funders to follow? I have asked this question at user meetings but the answer seems elusive. Determining what it is that makes developers do it (demonstrated benefit to each *individual*, goodwill?) might help and is something which needs to be explored further.

6.7 Summary

I am generally satisfied with the progress made with regards to the goals presented in "Project Aims", §1.2. The overarching idea was to explore the scholarly funding field, and I was particularly interested in doing it in such a way that I'd then (after this phase was completed) be able to develop FundFind into a proper service, leveraging Cottage Labs LLP's professional network and presenting it to the Open Knowledge Foundation, making it a valuable part of that movement towards Open Data.

- "Learn more about the funding sector and identify user requirements within it." - This was clearly satisfied, with the breakdown by potential audience members in "Audience and High-Level Requirements", §1.4 probably being the most valuable result. With this categorisation in hand, new examples can easily be added just by talking to professionals in the Higher Education sector, e.g. asking around at hackdays where publishers, representatives of funders (e.g. Wellcome Trust, Jisc, other HEI-s), the ARMA conference for research development officers. Thus, future exploration of user needs in this sector is enabled.

- "Identify how much software it is possible to build by essentially starting and seeing how far it goes in a controlled fashion." - I would have certainly preferred to have a better-tested, more expansive system. A more structured approach to development might have

been useful, but the number of hours spent certainly was not below the expected 400 hours by much. Software in academia, especially prototypes, are often built with the sole purpose of learning about a problem, or how to handle it with technology. The software quality is not guaranteed to be very high from a pure software engineering perspective (although it tries to stick to best practice), but its development allowed the feature list presented in "Requirements", §2.4. It satisfies the goal of enabling future "concrete development work which would enhance the software built during this project" from "Project Aims", §1.2.

I no longer wonder where to begin - as with many other projects developed in academia, I cannot see the end, but I now have something to aim for.

Appendices

Appendix A

Third-Party Code and Libraries

1.1 FundFind

1.1.1 Python Packages

1.1.1.1 Python Standard Distribution Packages

`json` is used for parsing the JSON data format (e.g. the app configuration is in a JSON file, the API responds with JSON).

`sys` and `os` are Python packages for interacting with the Operating System. `sys` contains certain global pieces of information such as the running Python version number.

`logging` handles logging output such as warnings, errors or just debugging information to the command line and to files.

`uuid` handles the generation of UUID-format unique identifiers.

`UserDict`, when subclassed by an application class, gives that class most of the properties of a dictionary. All `fundfind.dao` objects have property since they inherit from `fundfind.dao.DomainObject` which in turn is a child of `UserDict`. This allows things like `account['department']`.

`httplib` is one of the myriad of Python libraries for handling HTTP connections. `requests` is another one, albeit much cleaner and easier to use and is used in the FundFind tests. `httplib` is used by certain `dao` methods for accessing the datastore which were not changed from the parent IDFind codebase and can probably be refactored out of use by making more extensive use of the `pyes` elasticsearch access library.

`datetime` handles time in Python.

`re` is the standard regular expressions Python module.

`unicodedata` lends a hand in handling Unicode characters to the `util` module when producing URL-friendly slugs of strings.

1.1.1.2 Other Python Packages

Flask (`flask` in imports) is the main back-end framework upon which the whole web application is based [80]. The `fundfind.dao` module imports `werkzeug`, which is the underlying web server component for Flask for help with generating user password hashes.

The Flask-Login extension (`flask.ext.login` in imports) was used for helping with user authentication [82]. It does not provide registration capabilities or creating profiles.

`pyes` was used for connecting to elasticsearch datastore [72].

`parsedatetime` [42] is a module which tries to guess the time and date format from a simple string.

1.1.2 Non-Python Code

Web user interface libraries - Bootstrap [69], jQuery [83], jQuery UI [67], linkify [71] (linkify is just a jQuery plugin which turns text on HTML pages that looks like a URL into an actual clickable URL).

Search page - facetview [14], developed as a jQuery plugin.

1.2 Funding Harvest

Funding Harvest uses the standard distribution Python packages `datetime`, `logging`, `json`, `sys`, `uuid`, `UserDict` and `httplib`, described above, for the same reasons (mostly `config` and `dao` modules, which are very similar to FundFind's). It also uses `copy`, which a standard module which allows deep copies of objects to be made (i.e. when copying complex objects the actual values are copied, not just pointers to other objects, no matter how complex the hierarchy).

It does use one third-party (not a Python standard) module - feedparser [40] for parsing RSS feeds.

Annotated Bibliography

[1] Alok Jha, "Open access is the future of academic publishing, says Finch report," http://www.guardian.co.uk/science/2012/jun/19/open-access-academic-publishing-finch-report, June 2012, accessed November 2012.

> A Guardian article describing the Finch report which looked into Open Access and its benefits for academic publishing and society at large.

[2] Apache Software Foundation, "Content Negotiation," http://httpd.apache.org/docs/2.2/content-negotiation.html, accessed April 2013.

> A page in the Apache Webserver documentation explaining what content negotiation is and how Apache implements is. It is being cited for the clear explanation of the subject.

[3] Armin Ronacher, "Welcome to Jinja2," http://jinja.pocoo.org/docs/, accessed April 2013.

> The welcome page of the Jinja templating engine, a templating language for Python which comes bundled with the Flask web framework.

[4] Aslak Hellesoy, "Cucumber - Making BDD fun," https://cukes.info/, accessed April 2013.

> Cucumber is a system for Behaviour-Driven Development, which was considered as one of the options for development methodologies for this project.

[5] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas, "Manifesto for Agile Software Development," http://agilemanifesto.org/, accessed November 2012.

> A succinct set of statements enumerating the core values behind the Agile software development collection of approaches to software development.

[6] Corey Goldberg, "Multi-Mechanize - Performance Test Framework," http://testutils.org/multi-mechanize/, accessed April 2013.

> The home page of multi-mechanize, a performance (stress) testing framework written in Python.

[7] Cottage Labs LLP, "artemis," https://github.com/CottageLabs/artemis, accessed April 2013.

> The GitHub versioning system page of artemis - a parts management system for a commercial company.

[8] ——, "cottage labs - we don't sell products, we trade our skills," http://cottagelabs.com/, accessed November 2012.

The homepage of a partnership of freelance software developers working together to produce open source software for the benefit of Higher Education and commercial companies around the world. Dedicated to Open Knowledge and Open Scholarship.

[9] ——, "Cottage Labs website code," https://github.com/CottageLabs/cl, accessed April 2013.

The GitHub versioning system page of cl - the source code of www.cottagelabs.com .

[10] ——, "LEAPS," https://github.com/CottageLabs/leaps, accessed April 2013.

The GitHub versioning system page of the Lothians Equal Access Programme for Schools (LEAPS) online questionnaire management software.

[11] ——, "Mark MacGillivray / Cottage Labs," http://cottagelabs.com/people/mark, accessed April 2013.

A page on the Cottage Labs website describing Mark MacGillivray, one of the partnership's founders.

[12] ——, "portality," https://github.com/CottageLabs/portality, accessed April 2013.

The GitHub versioning system page of portality - All the basics for starting a portal frontend. Essentially an attempt to make the back-end of IDFind, FundFind and many other Cottage Labs projects much more reusable and encapsulate best practice while doing so.

[13] ——, "XCRI demonstrator project code," https://github.com/CottageLabs/xcri, accessed April 2013.

The GitHub versioning system page of xcri - the source code of an application which consumes XCRI-CAP course feeds, for the JISC XCRI-CAP course feeds demonstrator project. (XCRI-CAP is a way of describing course data for education, including Higher Education. The idea is to have the data in a machine-frienly form besides the institution's web pages.)

[14] Cottage Labs LLP and Open Knowledge Foundation, "FacetView - Pure Javascript Frontend for SOLR and ElasticSearch," http://okfnlabs.org/facetview/, accessed November 2012.

Homepage of the Facetview project, a faceted Javascript browser (front-end) to indexing servers such as elasticsearch.

[15] Craig Brierley, "Wellcome Trust strengthens its open access policy," http://www.wellcome.ac.uk/News/Media-office/Press-releases/2012/WTVM055745.htm, June 2012, accessed November 2012.

A press release by the Wellcome Trust stating it is strengthening its Open Access policy and related punishment for breaking the terms of the policy, as well as linking to a full version of the policy.

[16] Creative Commons, "Creative Commons Attribution 3.0 Unported (CC BY 3.0)," http://creativecommons.org/licenses/by/3.0/, accessed April 2013.

Readable (i.e. not legalese) explanation of the Creative Commons Attribution license version 3.0 Unported, with links to the legal text.

[17] ——, "Creative Commons Staff," http://creativecommons.org/staff, accessed April 2013.

A diagram showing the structure and listing the members of the Creative Commons team.

[18] Don Wells, "Extreme Programming Project," http://www.extremeprogramming.org/map/project.html, accessed November 2012.

A diagram showcasing the Extreme Programming (Agile software development practice) project lifecycle. Different sections of the diagram provide links to more detailed explanations of the various parts of the project lifecycle.

[19] Economic and Social Research Council, "ESRC RSS feeds," http://www.esrc.ac.uk/rss-feeds.aspx, accessed April 2013.

A page on the ESRC's website which describes the varius RSS feeds and the ability to create custom ones (essentially creating an API of sorts).

[20] Elasticsearch Global BV, "elasticsearch guide / date format," http://www.elasticsearch.org/guide/reference/mapping/date-format/, accessed April 2013.

Discusses possible values in date formats in elasticsearch dynamic and custom mappings. Has an exhaustive list of pre-defined date formats which elasticsearch will recognise.

[21] ——, "elasticsearch guide / uid field," http://www.elasticsearch.org/guide/reference/mapping/uid-field/, accessed April 2013.

Discusses internal unique identifiers (uid) in elasticsearch and the uniqueness characteristics of id fields in documents (which is why it is cited).

[22] Emanuil Tolev, "fundfind at GitHub," https://github.com/emanuil-tolev/fundfind, accessed April 2013.

Source code repository for FundFind, the main software output of this project.

[23] ——, "fundingharvest at GitHub," https://github.com/emanuil-tolev/fundingharvest, accessed April 2013.

Source code repository for Funding Harvest, the secondary software output of this project.

[24] ——, "Open: Scholarship, Science, Knowledge, Access," http://cottagelabs.com/news/open-scholarship-science-knowledge-access, accessed April 2013.

A blog post by the author of this document, describing a beginner's point of view to the Open Knowledge / .

[25] ——, "Progress Report - FundFind," Nov. 2012.

Progress report on this project.

[26] Emanuil Tolev and Cottage Labs LLP, "ID Find - an identifier identifier," http://test.cottagelabs.com/idfind/, accessed April 2013.

The current home page and public instance of the IDFind project.

[27] ——, "IDFind - Got an ID? Not sure what it is? What you can do with it? Well, use this!" https://github.com/CottageLabs/idfind/, accessed April 2013.

The version control repository page of the IDFind project. It tries to guess what kind of identifier an unknown string is.

[28] Engineering and Physical Sciences Research Council, "A step towards an intelligent information infrastructure," http://www.epsrc.ac.uk/funding/calls/open/Pages/intelligentinformationinfrastructure.aspx, accessed April 2013.

An example of an open funding call considered when thinking of what values describe a funding opportunity for the crowdsourcing pages. (Note: it will become an Archived or Closed call at some point, but the fields and their values should still be clear).

[29] ——, "EPSRC Open Calls RSS Feed," http://www.epsrc.ac.uk/funding/calls/open/pages/rss2.aspx, accessed April 2013.

A machine-readable RSS feed of current funding opportunities at the EPSRC, one of the UK Research Councils.

[30] Eric S. Raymond, "The Cathedral and the Bazaar," http://www.catb.org/~esr/writings/cathedral-bazaar/cathedral-bazaar/, Sept. 2000, accessed November 2012.

A discussion on why / how JSON maps better to intuitive human understanding of the world than XML does and why this is important for a data representation format.

[31] Ex Libris Ltd., "ExLibris Primo - A New User Experience," http://www.exlibrisgroup.com/category/PrimoUserExperience, accessed April 2013.

Describes the expected user experience that users of this library catalogue system might have. In particular, presents faceted search (called faceted navigation) as a big advantage when searching through a lot of data.

[32] FAST, "Organisation Details - IBM UK Research Laboratories," http://www.fastuk.org/atcommunity/orgview.php?id=3073, accessed April 2013.

"The Foundation for Assistive Technology was founded in 1998 to tackle the inadequate design of assistive technology products and services." This listing of IBM as a "source of funding" in their R&D section seemed as an appropriate reference to IBM Research UK as a commercial R&D vendor in the context of the project.

[33] GitHub Inc, "GitHub - Homepage," https://github.com/, accessed November 2012.

Currently popular decentralised version control system, usually used for software source code (but suitable for any artefacts although less suitable for binary formats or large files).

[34] Guido van Rossum <guido at python.org>, Barry Warsaw <barry at python.org>, "PEP 8: Style Guide for Python Code," http://www.python.org/dev/peps/pep-0008/, accessed April 2013.

Several sections of the Python style Guide are cited. The topics of readability and naming code organisation units are of particular interest since they influence the design, implementation and naming of design elements in the software.

[35] Ian Sample, "Free access to British scientific research within two years," http://www.guardian.co.uk/science/2012/jul/15/free-access-british-scientific-research, July 2012, accessed April 2013.

An article in the electronic edition in the Guardian newspaper detailing the UK Government's policy on Open Access publications in 2014.

[36] Jisc, "Our history : JISC," http://www.jisc.ac.uk/, accessed April 2013.

History page of Jisc. JISC have rebranded as Jisc.

[37] ——, "The Advantages of API-s," http://www.jisc.ac.uk/publications/reports/2012/advantages-of-api.aspx, Dec. 2012, accessed April 2013.

A Jisc report on the advantages of Application Programming Interfaces.

[38] Knight Foundation, "Knight Foundation - Informed & Engaged Communities," http://www.knightfoundation.org/, accessed November 2012.

The homepage of a global charitable foundation which supports (financially and otherwise) ideas that aim to change the world. They are particularly interested in quality journalism, media innovation, engaging the various communities around the world / in society and in fostering the arts.

[39] Kris Jordan, "First we built an API, then we built a CMS," http://www.gethifi.com/blog/first-we-built-an-api-then-we-built-a-cms, Dec. 2010, accessed April 2013.

An article by a HiFi company developer detailing why they built their Application Programming Interface first, right from the start.

[40] Kurt McKee, Mark Pilgrim, "feedparser 5.1.3," https://pypi.python.org/pypi/feedparser/, accessed April 2013.

The Python Package Index page of feedparser, a Python third-party package which can parse multiple feed formats such as Atom and RSS.

[41] Mark MacGillivray, "Gateway to Research data visualised using graphview," http://test.cottagelabs.com/gtr/graphview/, accessed April 2013.

An example of the graphview javascript library running, trying to visualise data from the Gateway to Research project. Unfortunately, the library is still in its pre-alpha stages and has not been open-sourced yet.

[42] Mike Taylor, "parsedatetime 1.1.2," https://pypi.python.org/pypi/parsedatetime/, accessed April 2013.

The Python Package Index page of parsedatetime, a Python third-party module which tries to parse human-readable dates and times (i.e. understand dates from pure strings).

[43] Nature Publishing Group (Editorial), "Banish cronyism," http://www.nature.com/news/banish-cronyism-1.11642, accessed April 2013.

An editorial describing a recent scandal with cancer research funds in Texas and allegations of cronyism in science.

[44] Open Knowledge Foundation, "Let's link! - nomenklatura," http://nomenklatura.pudo.org/, accessed April 2013.

The home page of the nomenklatura project, a web applicatoin for making mappings of variant names to a canonical form (e.g. Prifysgol Aberystwyth University might be the canonical form of the University of Aberystwyth variant).

[45] ——, "OKFN Activity API," https://github.com/okfn/activityapi, accessed April 2013.

The GitHub versioning system page of the activityapi project, an application whose main purpose it to provide an API which aggregates online activity of the Open Knowledge Foundation (e.g. tweets, blog posts).

[46] ——, "Open Definition v.1.1," http://opendefinition.org/okd/, accessed April 2013.

The Open Definition - defines what "open" is in relation to content and data.

[47] ——, "Open Knowledge Foundation Github Organisation Page," https://github.com/okfn, accessed November 2012.

The homepage of the Open Knowledge Foundation organisation on Github, the currently popular version control system.

[48] ——, "Open Knowledge Labs creates services and tools that unlock the digital commons for everyone." http://okfnlabs.org/, accessed November 2012.

The homepage of a collection of sofware projects which have sprung up to support or resulted from the Open Knowledge Foundation's pursuit of spreading the Openness cause in society.

[49] ——, "Our Vision / We Believe in the Power of Openness," http://okfn.org/about/vision/, accessed November 2012.

Notes on the Open Knowledge Foundation's vision of how openness can be beneficial for society (meaning all of current human civilisation).

[50] ——, "PyBossa," https://github.com/PyBossa/pybossa, accessed April 2013.

The home page of the pybossa project, a web application which enables crowdsourcing tasks which require human intervention to be executed. An instance of this application powers http://crowdcrafting.org/.

[51] ——, "Radial Bubble Tree Visualization," https://github.com/okfn/bubbletree, accessed November 2012.

Homepage of BubbleTree, a higher-level Javascript graphical visualisation library.

[52] Open Knowledge Foundation Deutschland e.V., "Was passiert in Bundestag und Bundesrat? - OffenesParlament," http://offenesparlament.de/, accessed April 2013.

The home page of the Offenes Parlament project, a German parliament legislative process tracker.

[53] Public Library of Science and Cottage Labs LLP, "negotiate 0.0.1," https://pypi.python.org/pypi/negotiate, accessed April 2013.

The Python Package Index page of a Python package which aims to provide content negotiation functionality to Python web application, and claims to integrate particularly well with Flask.

[54] ——, "OpenArticleGauge," http://oag.cottagelabs.com/, accessed April 2013.

The home page of the OpenArticleGauge project, part of the HowOpenIsIt suite, developed by the noted authors.

[55] ——, "OpenArticleGauge API Documentation," http://oag.cottagelabs.com/developers/api, accessed April 2013.

The API documentation of the OpenArticleGauge project, part of the HowOpenIsIt suite, developed by the noted authors.

[56] Python Software Foundation, "Modules in Python," http://docs.python.org/2/tutorial/modules.html, accessed April 2013.

A section of the Python documentation dedicated to Python modules as a way of organising code in Python.

[57] Research Councils United Kingdom, "Gateway to Research," http://www.rcuk.ac.uk/research/Pages/gtr.aspx, accessed April 2013.

A description of the Gateway to Research project by its parent organisation, Research Councils UK.

[58] ——, "Gateway to Research," http://gtr.rcuk.ac.uk/, accessed April 2013.

The home page of the Gateway to Research project.

[59] Richard Jones, "negotiator 1.0.0," https://pypi.python.org/pypi/negotiator, accessed April 2013.

The Python Package Index page of a Python framework for making content negotiation decisions based on the HTTP accept headers.)

[60] Richard Stallman, "Richard Stallman / A Serious Bio," http://stallman.org/#serious, accessed April 2013.

A section of Richard Stallman's home page, the Founder of the Free Software Foundation. The page is from his personal website.

[61] Rolls-Royce plc, "Research - Rolls-Royce," http://www.rolls-royce.com/about/technology/research_programmes/, accessed April 2013.

Rolls-Royce is a commercial company which sports extensive R&D. This page details Research Programmes in a wide variety of (technical and scientific) sectors.

[62] Rufus Pollock, "Rufus Pollock / About," http://rufuspollock.org/about/, accessed April 2013.

A page describing Rufus Pollock, the Founder of the Open Knowledge Foundation. The page is from his personal website.

[63] Scrum Alliance Inc., "Scrum Is an Innovative Approach to Getting Work Done," http://scrumalliance.org/pages/what_is_scrum, accessed November 2012.

A succinct description of the Scrum framework of Agile software development practices.

[64] J. Shore, "The Art of Agile Development," http://www.jamesshore.com/Agile-Book/pair_programming.html, Feb. 2010, accessed November 2012.

The online edition of a famous work (book) on Agile Development. This section focuses on and provides a good description of Pair Programming, an Agile software development practice.

[65] Shuttleworth Foundation, "Shuttleworth Foundation - Supporting exceptional people to change the world," http://www.shuttleworthfoundation.org/, accessed November 2012.

The homepage of a global charitable foundation which funds individuals with its scholarships and then multiplies the money they invest in projects which aim to bring about social change for the better.

[66] Team DevXS; hello@devxs.org . University of Lincoln, Student as a Producer project., "DevXS Conference," http://devxs.org, accessed April 2013.

DevXS was a national student developer conference running from 11th to 13th November 2011 at the University of Lincoln.

[67] The jQuery Foundation, "jQuery UI," http://jqueryui.com/, accessed April 2013.

The home page of the jQuery UI Javascript/CSS library. Note: this is a separate project from jQuery itself.

[68] E. Tolev, "PivotalTracker - FundFind," https://www.pivotaltracker.com/projects/688967, Nov. 2012, accessed April 2013.

Publicly visible project management for the software component of this project. By default, it will show what the author is currently working on and what is in the backlog. Click on "Icebox" (in the top left-hand corner of the web page, next to the "CURRENT" and "BACKLOG" links) to see all recorded user stories and chores for getting the project done. Stories in the Icebox are ordered in descending order in terms of priority, i.e. the more important ones according to aggregated user opinion are at the top.

[69] Twitter Inc., "Bootstrap - Sleek, intuitive, and powerful front-end framework for faster and easier web development." http://twitter.github.com/bootstrap/, accessed November 2012.

The homepage of the Twitter Bootstrap User Interface library.

[70] UK Government - Unknown individual name, "DATA.GOV.UK - OPENING UP GOVERNMENT," http://data.gov.uk/, July 2012, accessed November 2012.

The UK Government Open Data portal containing various datasets released under Open licences.

[71] Unknown, "parsedatetime 1.1.2," https://github.com/uudashr/jquery-linkify, accessed April 2013.

The GitHub source control system page of linkify, a jQuery plugin which turns text on HTML pages that looks like a URL into an actual clickable URL. The author's real name is unknown, although their GitHub handle / nickname is uudashr.

[72] ——, "pyes," https://github.com/aparo/pyes, accessed April 2013.

The GitHub source control system page of pyes, a Python connector for the elasticsearch datastore. The author's real name is unknown, although their GitHub handle / nickname is aparo.

[73] ——, "TeXCount," http://app.uio.no/ifi/texcount/, accessed April 2013.

A page describing the TeXCount script, a word count script for LaTeX documents. Its ability to produce a wordcount for a document as complex as this is a testament to its flexibility.

[74] Vangelis Bibakis, "MobileTest.me - Test your mobile sites and web applications right from your browser," http://mobiletest.me/, accessed April 2013.

A web application which allows testing of other web applications' mobile user interfaces.

[75] Various, "Why developers hate iframes?" http://stackoverflow.com/a/1081330/1154882, accessed April 2013.

A discussion on why the iframe element is considered extremely bad practice in web development.

[76] Various (a UK Government Working Group), chaired by Dame Janet Finch, "Accessibility, sustainability, excellence: how to expand access to research publications," http://www.researchinfonet.org/wp-content/uploads/2012/06/Finch-Group-report-FINAL-VERSION.pdf, June 2012, accessed November 2012.

The Finch report - looked into Open Access and its benefits for academic publishing and society at large, commisioned by the UK Government.

[77] Various authors - an open-source project, "D3 - Data-Driven Documents," http://d3js.org/, accessed November 2012.

Homepage of D3.js, a lower-level Javascript graphical visualisation library.

[78] ——, "elasticsearch - You know, for Search," http://www.elasticsearch.org/, accessed November 2012.

The homepage of the elasticsearch indexing server.

[79] ——, "Feedparser Documentation / Parsing a feed from a string," http://pythonhosted.org/feedparser/introduction.html#parsing-a-feed-from-a-string, accessed April 2013.

Gives a code example in Python on how to parse an RSS (or other e.g. Atom) feed using the feedparser Python library. The note specifying that the return value types could vary is of particular interest.

[80] ——, "Flask - Web development, one drop at a time," http://flask.pocoo.org/, accessed November 2012.

Homepage of the Flask Python web framework project, helps build RESTful web applications in Python.

[81] ——, "Flask Documentation Foreword," http://flask.pocoo.org/docs/foreword/, accessed April 2013.

A part of the Flask documentation which is supposed to be read before one gets started with Flask. Strives to answer some questions about the purpose and goals of the project, and when it should or should not be used it.

[82] ——, "Flask-Login," https://flask-login.readthedocs.org/en/latest/, accessed April 2013.

The home page of the Flask-Login extension for Flask, which helps with common operations needed to authenticate users. Does not help with creating profiles, i.e. registering users.

[83] ——, "jQuery is a new kind of JavaScript Library." http://jquery.com/, accessed November 2012.

The homepage of the jQuery Javascript and User Interface library.

[84] ——, "SeleniumHQ - Web application testing system," http://seleniumhq.org/, accessed November 2012.

The homepage of an open-source software project which enables the automation of web browsers and other testing techniques for the purposes of testing web applications.

[85] Various authors / Wikipedia, "Faceted search," http://en.wikipedia.org/wiki/Faceted_search, accessed November 2012.

The Wikipedia page on faceted search. Would not make much sense to a reader unfamiliar with the topic, but should do with the example in the report body.

[86] Wellcome Trust, "Biomedical resource and technology development grants," http://www.wellcome.ac.uk/Funding/Biomedical-science/Funding-schemes/Strategic-awards-and-initiatives/WTDV031727.htm, accessed April 2013.

An example of a funding opportunity by the Wellcome Trust considered when thinking of what values describe a funding opportunity for the crowdsourcing pages.

[87] ——, "Wellcome Trust - We are a global charitable foundation dedicated to achieving extraordinary improvements in health by supporting the brightest minds." http://www.wellcome.ac.uk/, accessed November 2012.

The homepage of a global charitable foundation which funds lots of scholarly activities in a variety of areas.

[88] WTForms Team, "WTForms Documentation," http://wtforms.simplecodes.com/docs/1.0.3/, accessed April 2013.

The main page of the WTForms project which tries to automate certain common tasks when generating HTML submission forms for the web.