SENG402 PROJECT REPORT

Open-sourcing CS education: Computer Science Field Guide 2.0

October 16, 2015

Marcus Stenfert Kroese
98446809
University of Canterbury
Department of Computer Science and Software Engineering

Abstract

This report details efforts made towards reworking the publishing infrastructure of the Computer Science Field Guide (CSFG). It deconstructs the old system, examines implementation options for a new CSFG publishing system. The system was implemented during this project, and the new (GitHub-based) system was evaluated. It moves to discuss the internationalisation and localisation challenges faced by an open source textbook like the CSFG. Finally, it presents an application designed to aid contributions to the guide.

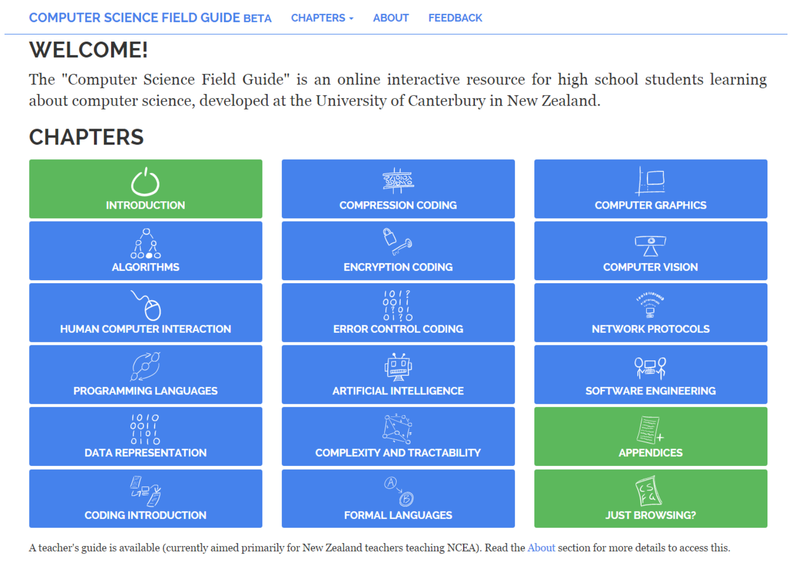
Contents

List of Figures

1 Introduction

1.1 What is the CSFG?

The Computer Science Field Guide is an online interactive ‘textbook’ for use in high school computer science education. It sees heavy use in New Zealand high schools, and has international recognition. The CSFG is helping support a broader view of computer science education being adopted in several countries [1], by providing students and teachers with an interactive learning platform.



WELCOME!

The "Computer Science Field Guide" is an online interactive resource for high school students learning about computer science, developed at the University of Canterbury in New Zealand.

CHAPTERS

A teacher's guide is available (currently aimed primarily for New Zealand teachers teaching NCEA). Read the About section for more details to access this.

Figure 1: The CSFG homepage

The content of the CSFG was designed in parallel with parts of the NZ high school curriculum from 2012. Schools needed support for the emerging computer science standards, and the CSFG was designed to provide that support. Its content covers a range of common CS topics, such as algorithms, formal languages, and network protocols, as shown on the homepage in Figure 1.

1.2 Problem statement

The infrastructure on which the CSFG was built could not support its future needs. The system lacked the scalability to grow as an open source resource. Another issue was that the core CSFG team are limited in the time and money they can commit to the project. However, the growth and improvement of the book should not be limited by the team’s capacity to contribute.

The CSFG also needed to be more adaptable. Because the CSFG is relatively new, others are often adapting it for international use. The infrastructure had other problems as well. While the web page output of the system was stable, PDF output had been defective for some time. There were also several aspects of the system which made editing difficult.

The growth of the CSFG faces more a general issue too; open source textbooks are a fairly new phenomenon. Understanding of how to create and maintain high quality open source textbooks is still developing, so there will be no 'correct' solution to the problems faced.

1.3 Initial objectives

To develop a solution to the problem, we model and analyse the existing system. This includes unique non-functional aspects of the CSFG, such as humour. It also involved identifying the constraints and requirements imposed by the old system. Finally, implementation options for a new system were evaluated, and an overall system design was decided upon.

2 PRIOR WORK

Several areas of prior research were investigated. The first of these was computer science education material, especially papers relating to the CSFG. Discussion of open source education was particularly relevant.

Bell et al. [1] give an overview of the CSFG and its benefits and motivation, which are discussed above. They also outline some of the features of the CSFG, such as 'interactives', which allow students to engage with learning material in a conversational, exploratory way.

Miller and Ranum [2] found that students appreciated their course being organized all in one place in the form of an interactive textbook. They also expressed excitement at the possibilities of open source collaboration with people around the world. However, Miller and Ranum discussed that their textbook system could become a platform for instructors to add and customise their own courses, or adding material to a collective pool for others to use. At present, this is not the intention behind the CSFG content, which is more similar to that of a single course with many authors.

Other previous work gave insight into how online publishing is handled for systems similar to the CSFG.

In their subsequent paper, Miller and Ranum outline some of the features of their online textbook system [3]. The paper refers to a presentation in which they demonstrate the use of Runestone Interactive, and the way in which it provides tools for tech-savvy teachers to create their own courses or contribute to others. The presented benefits include embedded code visualisation and execution blocks, interactive assessment tools, and comment boxes to allow moderated discussion of problems. This is very conceptually different to how the CSFG is used, and highlights how diverse and complex interactive text books can be.

Ross and Grinder discuss the benefits of combining 'active learning' with online textbooks similar in principle to the CSFG [4]. They refer to these as with the term 'hypertextbooks'. Several recommendations made by Shaffer et al. [5] can be applied to a new CSFG system. Hypertextbook applets (analogous to CSFG interactives) should be available as standalone content, for use outside the book. The benefit of this is that applets can be used as self-contained tools for learning. This advice is relevant for CSFG interactives, especially those that require translation; each interactive should have self-contained translations and not be tied to the system that contains it.

3 Planning the new publishing system

3.1 Stakeholder analysis

In order to create accurate project requirements and help capture stakeholder knowledge, software-engineering-style requirements elicitation was carried out. The first step we took was to conduct a brief stakeholder analysis. This helped identify which elicitation techniques would be the most effective to use.

Three categories of stakeholder were considered; people who would help implement the new system, people who would use the new system, and people affected by the use of the new system. The stakeholder groups presented below are not exhaustive, but they represent the large majority of stakeholder.

The main person who would implement the system was the lead CSFG developer. Because of this, he was of critical importance to the project. He would also be the main user of the new system. Because of his extensive work developing the old CSFG system, he had the most technical domain knowledge of any stakeholder, as well as huge amounts of tacit knowledge. The rest of the CSFG development team were much less critical, because they would only be developing the system after its initial setup.

The users of the new publishing system were much more varied. This group consisted primarily of the CSFG development team, as well as the rest of the CSFG team. This included Professor Tim Bell, the CSFG project lead, and primary contact for this project. Prof. Bell acted as an advocate for other users of the system by clarifying their possible perspectives on using the system. For example, teachers with a low technical skill level might still want to be able to contribute as users, and should be considered. Because of this, Bell was a secondary critical stakeholder in the system. He was also able to provide some domain knowledge about the challenges of open source textbooks. Other members of the CSFG team were considered to be major stakeholders, since they would often interacted with the CSFG. Other contributors were considered minor stakeholders.

People that would be affected by the new system included the users of the CSFG content, such as students and teachers. This could also include international educators wanting to source content from the CSFG, or people looking to create educational material similar to the CSFG. The German equivalent of the CSFG is an example of something such people might want to create [1].

Elicitation processes

Because of the project size, and the low number of critical stakeholders (two), a mix of structured and unstructured interviews were conducted with each of those stakeholder. This helped clarify the scope of the project, and provided greater understanding of the requirements of the new system, some of which had been the tacit knowledge of one stakeholder or the other.

Because understanding the old system was critical to planning a new system, investigations into the old workflows were carried out. The lead CSFG developer also gave a guided walkthrough of the old system. These activities contributed to an overall analysis of the old CSFG publishing system.

3.2 Analysis of the old system

Existing system structure and data flow

The pre-existing publishing system was made up of several components. The main technologies used were Google Docs (for writing content), GoogleCL (a command line tool for extracting Google Docs text) [2], and Sphinx (a documentation generator) [3]. The publishing process was executed by a University of Canterbury server at scheduled intervals using these tools.

Figure 2 gives an overview of the data flow during the publishing of a single “chapter". A single Google Doc contains text written in the reStructuredText markup syntax, which is parsed by Sphinx. Parsed items include comments (ignored in output), headings, and several custom extensions. The extensions would specify glossary items (to populate the Glossary

[1]http://www.inf-schule.de/
[2]https://code.google.com/p/googlecl/
[3]http://sphinx-doc.org/

page), or denote insertion points for files like images or interactives.

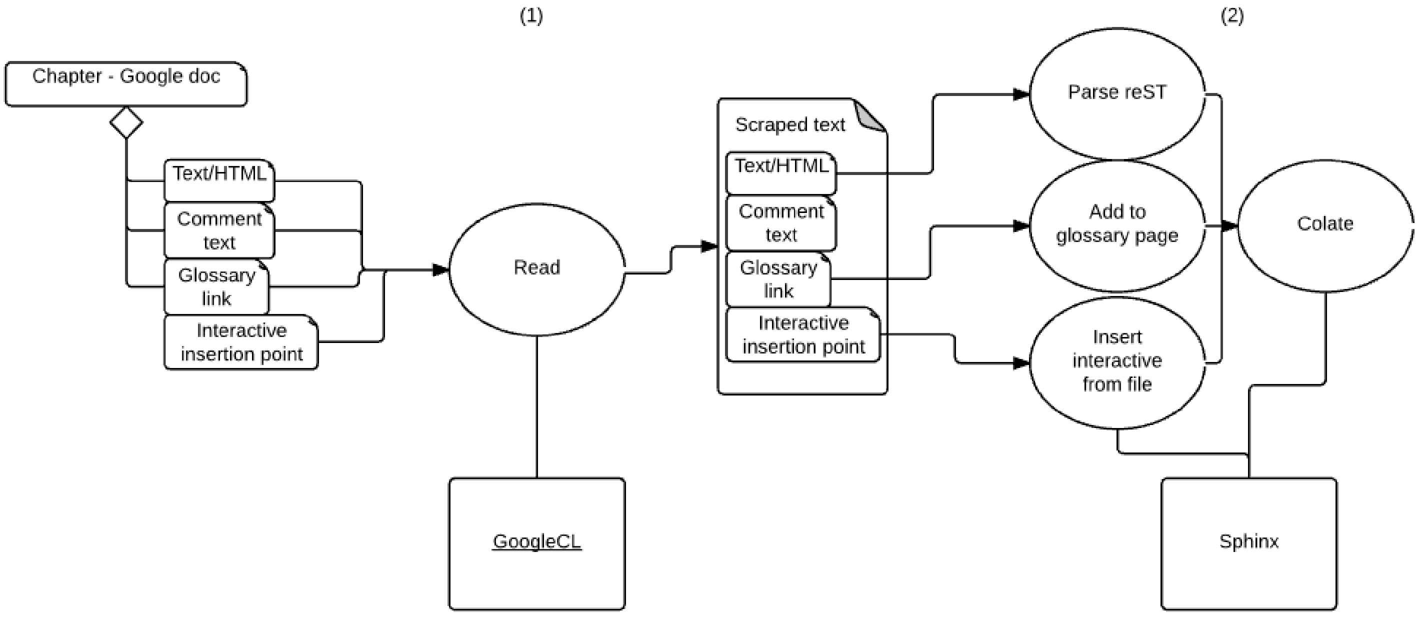


Figure 2: Previous system use cases and function overview (Based on UML data flow)

Once the reStructuredText (reST) was read using GoogleCL, it was processed by the local Sphinx installation. A configuration file provided declarations of the extensions Sphinx should handle. Sphinx carried out the actions of parsing the reST itself, including any HTML snippets in the text, and creates the final output. In the process, it linked chapters with local files. The output was generated in two formats: HTML (for web hosting), and PDF (for printing/file distribution).

Non-functional qualities

The CSFG has several unique non-functional qualities. The teaching principles of the CSFG contribute to its uniqueness. It is written in a constructivist way, aiming to lead students through experiences that enable them to construct concepts in their own minds, rather than simply giving them information. Quirkiness and humour are also a core aspect of the CSFG, as figure 3 shows. This aims to engage students with the material [6].

Figure 3: A snapshot of a funny video from the CSFG

Challenges of embracing open source

Several challenges for the CSFG were uncovered in the elicitation. These impose additional constraints on the project.

One challenge is how to deal with 'secret' parts of the book. Textbooks often have a teachers' version that includes answers to questions. Teachers appreciate having these answers available so they can be sure they've got things right and then help stimulate discussion about the question. Openness brings the concern that a student might download the teacher version and subvert their own learning by simply reading off answers instead of thinking though the questions. [6]

Maintaining consistency of content across the CSFG is hugely important. Authors have a wide variety of styles, not all of which align with the constructivist style and quirkiness of the CSFG. Another author might see the constructivist teaching as not giving all the information, and they might "fill in the gaps". This would clearly (yet inadvertently) undermine the pedagogy behind the CSFG. This could also happen if the CSFG became too democratised, such that content needed to be agreed upon by a concerned committee.

Managing collaboration and content is another major issue. As described by [7], even comparatively simple edited books can lack coherency. Contributors can come from a variety of backgrounds; they may be teachers, computer scientists in education, or educators from overseas. Each person will have different motivations for contributing, and a different writing style. For the CSFG, the insights of these people need to be acknowledged without compromising style and pedagogy. However, discouraging contributors too much could stifle improvements to the CSFG.

Plagiarism of content is another possible pitfall. It is possible that co-authors might add plagiarised content to the CSFG without other authors knowing. Lewis [8] notes that this occurred in an entire chapter of a co-authored book.

Deficiencies

Several deficiencies were identified in the existing system. reStructuredText was hard to use for untrained users (e.g. directives can be easily mistyped, specifying headings is ambiguous). Mistakes in the markup led to broken output pages. Using Google Docs with GoogleCL made commenting inconvenient; native Docs comments could not be used, as GoogleCL treated them as normal text by including them in the in web page output. There was no system in place for managing many suggested edits or content, so suggesting requires personal contact with moderators to discuss changes. At the time, Sphinx PDF publishing could only be achieved with excessive tweaking,and ePUB publishing had been abandoned due to Python library compatibility issues.

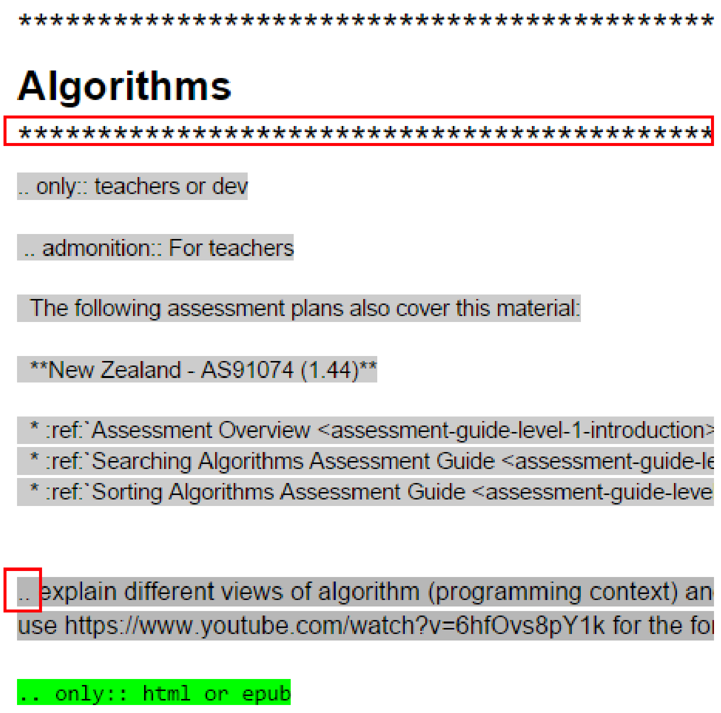
Even when using Sphinx extensions, reST has markup limitations; text cannot be both bold and italicised, and fine control over tables is not possible. Figure 4 illustrates two other issues. The first red box shows how headings are specified in reST. If there are fewer asterisks than the character length of the title, the text is not recognised as a heading. The second red box shows the two fullstops plus space character that must precede reST parser directives. Accidentally missing any of these three characters caused errors such as large chunks of author comments being thrown into the output, making them publicly visible to students and teachers.

3.3 Overall requirements

The workflows, deficiencies, non-functional qualities, and challenges of the old CSFG system were used to compile requirements for the new system. The requirements are specified at a very high level because of the small scope of the project; extremely fine-grained requirements were not appropriate when the solution space still needed exploring.

R1: Open source. The CSFG must allow developers and educators to access and improve on it [1]. The ability of volunteers to contribute to the Guide should not be damaged. If the Guide remains valid under the Creative Commons Attribution-NonCommercial-ShareAlike license, this requirement will have been satisfied.

R2: Support different contributor roles. The new system should support a minimum of


```
********************************************
Algorithms
********************************************
.. only:: teachers or dev

.. admonition:: For teachers

 The following assessment plans also cover this material:

 **New Zealand - AS91074 (1.44)**

 * :ref:`Assessment Overview <assessment-guide-level-1-introduction>
 * :ref:`Searching Algorithms Assessment Guide <assessment-guide-le
 * :ref:`Sorting Algorithms Assessment Guide <assessment-guide-leve


.. explain different views of algorithm (programming context) an
use https://www.youtube.com/watch?v=6hfOvs8pY1k for the fo

.. only:: html or epub
```

Figure 4: Error-causing reST syntax in the 'Algorithms' chapter Google Doc

three different types of contributor: the CSFG development team, and contributors of CSFG educational content. The development team should be able to change the Guide as they wish, with some means of coordinating their work. The educational contributors can write content, which would then be approved by the core team. This would help preserve the coherency and teaching style of the CSFG.

R3: Suggestion support. Suggestions of small fixes to the CSFG should be possible for anyone, provided they can identify themselves as a person somehow. This would allow suggestions of small fixes or ideas for improvements.

R4: Translation tracking. Because publishing system is intended to be a prototype for other CSFG versions in different countries and contexts, parts of it will need to be translated into other languages. The new system should support tracking and/or storage of these translations with reference to the English version.

R5: Revision history. The CSFG system needs a means of viewing past changes (and ideally the reasons for those changes). For this reason, it should support some form of revision history on at least a chapter level. The past changes should be visible or publishable as an official changelog. To support a changelog, changes should be supported with

]

editors' notes on their changes.

R6: Edition support. This means publishing differently for unique curriculums. Ideally the guide should allow modular publishing; some way of customising the content that enters a published output format. Each of these 'editions' could be tracked somewhere.

R7: Role-based content viewing. The CSFG should allow for a way of specifying content visible to different viewers. For instance, teachers might require notes on a sections' relevance to their curriculum, which a student may find distracting.

R8: Variable formatting (for different content types). The authorship language of the Guide should provide simple ways for contributors to insert styled content. An example is text accessible inside a collapsible pane (such as 'curiosity' text for keen learners).

R9: Web pages, PDF, and ebook as possible outputs. Because of the interactive nature of the CSFG, a web-based version would be a minimum requirement, as this should guarantee access to all interactive content. PDFs have the benefits of being ubiquitous, portable (small file size), and printable. They can also support a range of embedded content. An additional ebook format would be even more beneficial [9].

3.4 Implementation options and choices

Based on the requirements for the new publishing system, we can being exploring implementation options for the system.

Benefits of authoring languages

In order to contribute content, CSFG authors need to write the content in either plain text, or some form of markup language. This text should then be parsed in some way, or be typed into a program that allows it to be parsed easily. We propose that a lightweight markup language would suit the CSFG best.

Lightweight markup languages have a low interaction cost for teachers and other contributors. They require relatively little effort to learn, and closely resemble plain text for ease of reading, which is a benefit over 'heavier' markup languages such as XML. Conversion between lightweight markup languages is fairy simple, because of the structural similarities between them. This will make the conversion from the existing reST syntax simple. The formatting provided by lightweight markup languages are easy to parsed into HTML or PDF, as opposed to plain text, which requires additional treatment to produce a well-formatted output.

For less experienced authors, multiple editors are available that allow markup to be previewed in rich text. This helps authors visualise how their content will look when published, and add styling via GUI if they forget the language syntax. Overall, use of a markup language gives CSFG authors the power to style their text in an easy way, without the need for specific editing programs or applications.

Another possible alternative could involve using a 'What you see is what you get' HTML editor to let authors drastically customise the style of their content. This HTML could be directly added to the HTML of a CSFG web page, and could then be published. However, this puts more burden on the authors than is necessary, and makes it more difficult to keep the guide styled in a unified way.

Choice of authoring language

After specifying that the CSFG authoring language should be a lightweight markup, a specific language must be chosen.

The Markdown language provides many benefits. It is widely used and relatively well-known, in part due to its maturity. Markdown has slowly accrued more features as different so-called 'flavours' of Markdown have been developed. There are also many extensions for Markdown, which is in addition to the basic additions offered by different flavours. Extensions can allow Markdown parsers to support features like tables. Markdown is also well-supported in Github; CSFG content could be previewed easily there.

Potential alternatives to Markdown exist, but they are not suitable. The Asciidoc lanaguage, while popular, suffers some of the same drawbacks of the old reST system such as some ambiguous formatting.

LaTeX, a more complex markup language, has an extremely powerful and extensible macro system. These macros could be used to make CSFG-specific formatting easy to apply. However, finding a balance between LaTex's typesetting and post-parsing styling (when compiling chapters for the web) would be very inconvenient. Styling HTML is best handled with a single step (like applying CSS), rather than a two-stage process. This makes styling more predictable and easier to update. The effort required to create the LaTex macros was judged to be too high. Also, anyone wanting to maintain or alter the system would need to become proficient themselves, which costs time. LaTeX is also dissimilar to other markup languages, meaning teachers are less likely to have been exposed to it.

An established version control system (VCS) would provides many features that fulfil the project requirements. A VCS tracks every change made to the project, which helps to create logs of the revision history. High level overviews of the changes could be created by someone scanning through the most recent change snapshots (known as ‘commits’ or ‘changesets’). A standard naming scheme for commits would make this even easier. A VCS also allows the CSFG to support many more contributors. Each contributor can make their own changes on a separate working branch, and ask to merge their changes with the main guide. Intermediate authors or CSFG team members would still need to negotiate the changes, to preserve the constructivist teaching style and cohesiveness of the content.

Version control hosting options

Beyond a VCS, the CSFG would benefit from a repository (‘repo’) hosting service; a system that offers simple revision control and document management. Another benefit is improved role-based access. Repos can be owned by an ‘organisation’ group. Members of the organisation can have varying degrees of access to the repo, so it is possible to control who can commit changes. Two main options for hosted source control were considered; Bitbucket [4] and Github [5].

The two platforms were evaluated based on the needs of the CSFG using a small weighted decision matrix and accompanying justifications. Each factor was given a weighting based on its importance to the CSFG, based on the discussions with the CSFG development team. The category ‘Same’ indicated neither system has an advantage over the other for that criterion, while ‘Better’ indicates one system met that criterion better than the other.

Both platforms allow ‘commit messages’ to be attached to a set of text changes, which is a key requirement. Quickly reading the content of a commit is essential for creating teacher-friendly changelogs.

Ease of authentication, a sense of involvement, and a useful issue tracking software were given medium weightings. Github provides a range of metrics to view project participation, including views, other project traffic, and who is making contributions. In comparison, Bitbucket allows almost no viewing of who is involved in a project, or how project activity is changing.

[4]https://bitbucket.org/
[5]https://github.com/

Table 1: Comparison of version controlled hosting services

Criteria	Bitbucket	GitHub	Weighting
Easily allows 'commit' message for text changes	Same	Same	3
Sense of project community/involvement		Better	2
Ease of authentication	Better		2
Issue tracker	Better		2
Focus on public repositories		Better	1
Interface navigation	Same	Same	1
Hosts projects relevant to the CSFG		Better	1
Integration with free plugins		Better	1
Discoverability		Better	1
Community support		Better	1
Desktop client ease of use		Better	1

Ease of authentication is relevant to teachers; it is an inconvenience to create a new user account just to edit the CSFG. Bitbucket allows sign-ups using Facebook, Twitter, or Google accounts, making editing much more accessible to new contributors.

An issue tracker provides a space for people to give suggestions and feedback on the CSFG. Bitbucket has a marginally better issue tracker than Github as the UI is faster to use.

Github is promoted as more publicly-focused (by its pricing scheme, marketing scheme, and by those who use it). Bitbucket, however, focuses more on private use. As an open source project, this aspect of Github suits the CSFG well.

Both services pose similar interface navigation challenges for new users; neither is more usable than the other. The user interfaces rely on the user having knowledge of the VCS domain, which editors such as teachers are unlikely to have.

The presence of other projects relevant to the CSFG is also a consideration. The Bootstrap and Foundation UI frameworks, several Markdown parsers, and many Python packages are all hosted on Github.

Github also has more free plugins and tools developed for it, while Bitbucket's main plugins are proprietary software products by Atlassian, the company who created it.

Hosting the CSFG project somewhere it can be discovered is also useful to raise awareness of it. Github allows anyone to browse all publicly hosted repositories, and has a 'Showcases' webpage which allows the browsing of projects based on a persons interests. Finding a project in Bitbucket usually requires knowing exactly what to search for. For this reason, projects in

Github are more easily discoverable than those in Bitbucket; Github projects are more likely to attract new contributors and viewers.

Community support for Github is much greater than for Bitbucket. The Stack Overflow page for Github is much more active, meaning Github questions that are posted are answered more quickly. The open source attitude of Github also contributes to a more active and open community.

The desktop clients for each service could be used to help new editors contribute content. While Bitbucket's 'SourceTree' app is extremely useful for coding projects, its layout is overcrowded. Non-programmers would find it very difficult to use. The Github desktop app has a less intrusive user interface, and would be less intimidating for novice users. It would still require an instructional guide, because of the VCS-related jargon words present in the application.

Overall, Github was chosen as the most suitable hosting service, with a weighted score of 8 versus Bitbucket's score of 4.

3.5 Drawing from existing project infrastructures

Investigation into similar systems was carried out to discover their implementation principles.

The Runestone Interactive project and the OpenDSA project have similar aims to the CSFG [2], [10]. The systems both provide open source interactive education to young students. Ideally, these existing implementations could be adapted to fit the CSFG. Unfortunately, both projects lack the features required of the new framework, while still including some of the drawbacks.

Both projects showcase ways open source 'textbooks' can be implemented. Even though both projects were built with very similar philosophies, and the publishing workflows for authors are also aimed at ease of use, the drawbacks of re-purposing either framework outweigh the positives. One drawback is that the systems have different conceptual models of their content than the CSFG. The CSFG content structure (in chapters, with interactives and other material linked in) does not fit with the more intricate approaches of OpenDSA or Runestone, which are based around whole courses with dynamic questions and content.

Technical constraints also make these systems poor choices. Both Runestone Interactive and OpenDSA rely heavily on Sphinx and the reStructuredText markup language. As discussed earlier, reStructuredText must be avoided in the new system. Because of this heavy Sphinx usage, repurposing either system is not feasible. OpenDSA also focuses heavily on step-through algorithm visualisations as the main ‘interactive’ component for students, but the CSFG aims contain interactives that provide better student engagement [1].

In future, it might be useful to re-examine the systems to better understand their architecture. This is mainly because of Runestone’s focus on dynamic web content, which the CSFG doesn’t need in the foreseeable future, but could conceivably use for something (like monitoring student engagement with content).

3.6 Contributor workflows and use cases for the new system

The old CSFG system use cases should preserved after the systems are switched. To confirm that the new system features would support this, a mapping was created between the new system and old use cases. Figure 5 shows how the system use cases would change after shifting the system to GitHub.

It is equally important to plan how the new system will be used by primary contributors such as the CSFG development team. 6 gives an overview of the workflow for contributors (using formal Business Process Model and Notation) in the new system. The barrier to entry of this method is familiarity with Git, and ownership of a GitHub account. How these limitations could be overcome will be discussed in more detail.

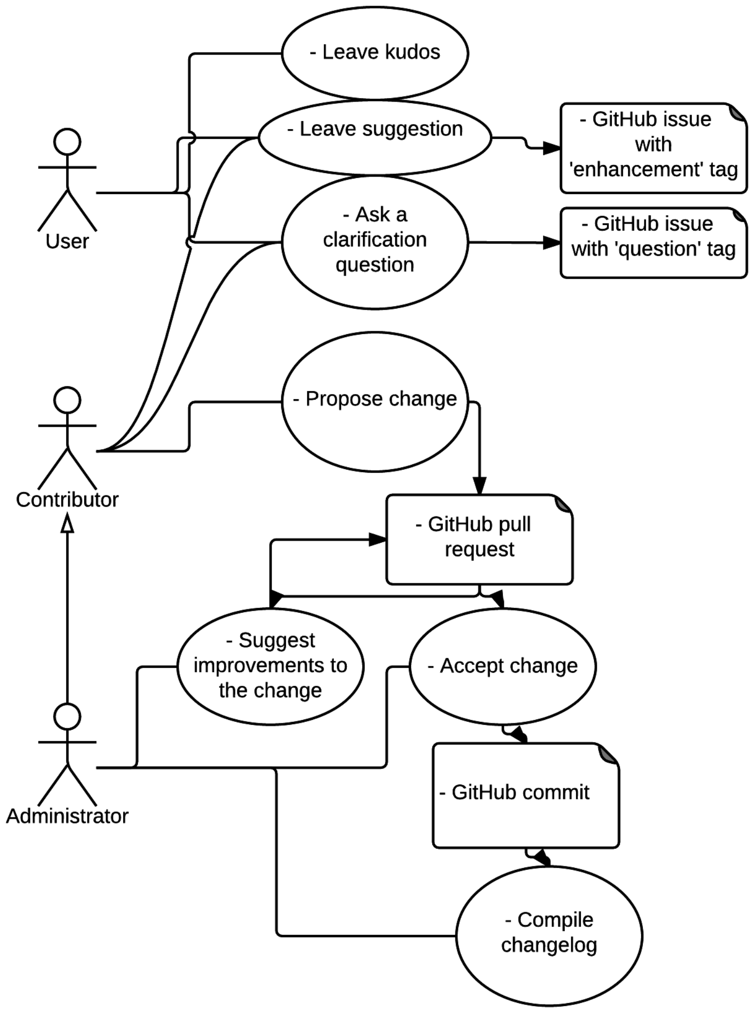


Figure 5: New CSFG contribution flows (hybrid UML use case/data flow)

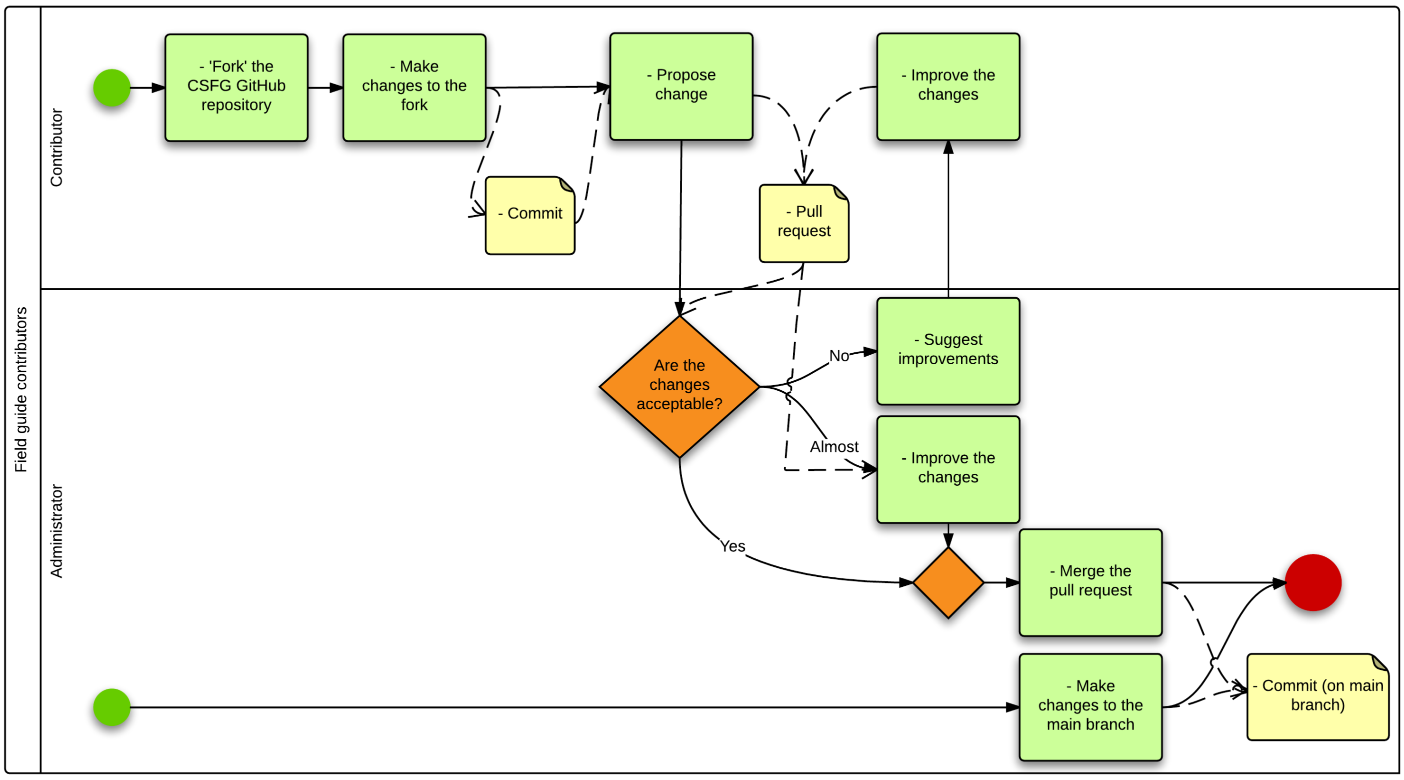


Figure 6: CSFG contributor and administrator contribution flows (BPMN)

3.7 Evaluation and feedback on the new system

After Part 1 was completed, the lead CSFG developer implemented a custom Markdown parsing process built on top of the markdown2 Python package. This was the start of the system transition. Other changes (not prescribed by this project) were made as the CSFG was shifted to the new structure; Bootstrap.js CSS styling for output was replaced with Foundation.js (for easier customisation), and the file structure of the CSFG was refined over several iterations, among other changes. After the new system had been in use for some time, an evaluation was carried out. The evaluation was conducted as an interview with the lead developer. His experiences are summarised below.

The development team quickly realised the huge benefits of distributed version control. The structure of the guide is constantly being tweaked and improved, which previously meant all other authoring needed to stop, so that the developers could run the output generation scripts to confirm that their changes had been successful. With the new system, the distributed VCS allows people to write content while the CSFG structure is being reshuffled, and simply merge their changes into the new structure once they've finished writing. Conversely, if someone was in the middle of writing text content, the web output could not be generated, so any slight bug fixes were made by manipulating the HTML directly until the fix could be applied properly. Again, these situations are no longer a problem.

The issue tracker is being put to good use; the developers use it to document bugs, suggestions, and enhancements. Issues are often linked to specific commits for traceability, making it easy to see when an issue was fixed or introduced. The issues act as a formal to-do list. Commit messages are also being used as intended, meeting the standards we have recommended. They focus on forming a readable repository history. Each message contains what the commit has changes, but also future behaviour that might benefit from the changes, and anything the changes might have broken.

As hoped, GitHub plugins are also being used. The Travis CI tool for continuous code integration runs a test with every commit (and successful pull request) to check that the CSFG output can be generated correctly, so that the CSFG is only re-published after successful builds. The tool could be used to run additional tests in future.

The access control provided by the new system will help the developers regulate CSFG contributions. A GitHub 'organisation' was set up, and CSFG developers and team members were assigned different levels of privilege. Developers can make commits directly to the any branch, be it the guide content or development branch, without restrictions. This allows them to make changes quickly and easily. CSFG team members and other contributors must submit a GitHub 'pull request' upon authoring a change, indicating they would like their change merged with the main branch. A CSFG developer then approves or rejects the pull request as appropriate, based on whether it maintains the integrity of the CSFG.

The CSFG has the potential to be taken and adapted by anyone, as was intended. Rather than existing as a collection of disparate scripts, files and Google Docs pages, the entire source code of the CSFG can be now be downloaded. The lead developer supported this by ensuring that all the publishing system dependencies were stored in the repo, meaning anyone could create and host their own CSFG pages.

The revision control and file history features of the system have been put to good use. On several occasions git's 'blame' feature has been used in the GitHub interface to see who might have changes particular pieces of code.

File management has been vastly improved over the old system; all files are in same repo instead of being spread across the internet and university file servers.

Overall, the CSFG seems to be evolving naturally as an open source project; better structures and features can be easily tested. The experiences of working with the system have been overwhelmingly positive.

4 INVESTIGATION OF TRANSLATION MANAGEMENT

Translation handling at some level is a key part of the CSFG's future. The problem is not a new one. Other systems have dealt with them in the past, with varying degrees of success.

4.1 Issues with translations

Supporting translation of content is much easier said than done. There are many potential issues and pitfalls to consider [11].

- General translation contribution issues
  - Translations add extra complexity on top of normal contribution.
  - Translation requires tool support.
  - Effective translating is difficult in this sort of technical and educational context.
  - Translation review processes might be required to preserve integrity.
- Quality control issues
  - Translations need to convey concepts & meaning correctly in the target language and dialect.
  - Translations must maintain learning objectives at the chapter and section levels.
- Potential layers of extra handling
  - Translation data must be stored somewhere, in way somewhat parallel to how the original data is stored.
  - Guidelines for translating must be provided at the documentation level.
  - Combining translations with untranslated material (like some interactives) must be managed somehow.
  - Images of diagrams must be re-captured by tweaking original diagram (which might no longer exist).
  - Character set and alphabet mismatches may need to be resolved.

  - Translation of videos (with dubbing or subtitling) must be done in a way that preserves their integrity.
  - Translation of interactives may be required, which could require refactoring several of them to store translations internally.

- Versioning issues
  - Commits to the CSFG may make translations outdated.
  - Languages are likely to have mixed levels of translation completion, making small bug fixes tedious to merge.

- Storage difficulties
  - Low level.
    * Custom file format may be needed.
    * Existing file formats in the repo may need reworking.
  - High level.
    * Github repositories.
    * Dedicated translation management software.

- Extra localization is needed
  - Example scenarios based on culture must be converted to equivalent cultural examples.
  - Language-based humour is often difficult to translate.
  - Videos must be culturally sensitive, or replaceable for different translations.
  - Alphabet-based visuals and examples translate poorly.

4.2 Translations in other systems

To gain a perspective on how translation management can be dealt with, two large, well-established projects were analysed for their handling of translations; Wikipedia, and DiscoverMeteor.

4.2.1 Wikipedia

The Wikimedia Foundation supports the translation of Wikipedia pages with a well-defined technical process [12].

Articles on the same topic are often written independently of each other, and no synchronization process takes place to verify that they correspond. Many Wikipedia articles are translated from other languages, however. This was done on a paragraph-by paragraph basis by the volunteer translators. In February 2015, a new tool, Content Translation was deployed to support translators. Its features include side-by-side article viewing, a dictionary, and a tool to adapt links and images between pages. However, this did not affect the lack of synchronization between articles. The burden is translators though; there is no editorial committee or chief editor appointed to handle synchronisation.

Because Wikipedia treats translated pages as independent, their backend database storage reflects this; it treats each page as its own entity [13, 14]. Translated pages are added to the database directly after being written thanks to the Content Translation tool. However, the pages are still dealt with separately based on language. Each group of translators for a specific language has their own database to work with. The database is structured as based on a general template, but includes localisations for different language dialects.

Because Wikipedia is translated in an open source way, we can draw parallels to the CSFG, and make an assumption that the two translation efforts are similar. Translators need to have knowledge of the two languages they are working with, as well as domain knowledge (which for the CSFG is computer science knowledge and experience in educating).

4.2.2 Discover Meteor

Discover Meteor is an open source book which helps developers learn to build web apps with Meteor.js. It has a translation project which effectively crowdsources translations.

Each translation of the project is stored in its own Git sub-repository. All the book chapters are in the top level of the repositories, with each chapter written in Markdown with embedded Ruby code. Each 'sub-repo' has its own guidelines and rules, created by the most active translators. Most sub-repos work in a single branch, with any slightly erroneous translation commits being corrected in subsequent commits. Translation is fairly incremental, so there is no need for branches to support the development and merging of larger changes.

Discover Meteor shows that repo branching is not always needed in a GitHub project; in each sub-repo, change are all made on the master branch, and the contributors themselves resolve any complex change merges that are required. However, this highlights that the CSFG needs its master branch to be clean as it is being deployed directly to students. The resource needs to be available at a high level of quality at all times. This is distinct from the Discover Meteor project, where an erroneous translation is of little consequence and will likely be fixed quickly. Changes to the CSFG must be made on another branch, and merged into the main branch by administrators close to the project. The branching strategy is planned to match the Driessen's popular strategy [15].

4.3 Conclusions on translation

Translation handling has been shown to be an extremely complicated task, with many factors to consider. Eventually we recommended that file translations be handled by storing them in the main CSFG repo, in the same folders as their associated English files.

5 SUPPORTING OPEN-SOURCE CONTRIBUTIONS

After the implementation of the main system, the focus of the project was shifted to another high priority goal; supporting contributions from multiple authors in the open-source context. This involved improving contributor access to the CSFG source material, and improving the ease of that access. There are several levels of contributors, stemming from peoples' various ideas and motivations. As described by Davis and Blossey [7], these motivations could be at odds with the motivations behind the CSFG. However, a person's ideas might still be valuable, so it is important to allow them to contribute in some way.

5.1 Description of different levels of contributions

People engage with the CSFG project at various levels. Some are casual readers, who might notice something trivial they would like to change. Others are core authors, responsible for designing and compiling CSFG content. To identify who we are assisting, it is important to group people along the contribution spectrum.

We categorise people based on the amount of content they are willing to contribute. This represents a level of enthusiasm or commitments to the project. 'Intermediate contributors' are those who want to add paragraph level content, such as teacher-only segments of the CSFG. Another category was the 'bold ideas' group, who, with good intentions, wish to contribute to the CSFG. They may wish to contribute content that does not fit into the CSFG well. However, the ideas may have some value, even if the specific content or writing style is inadequate. These classes of contributors are most at risk of being deterred by the developer workflow for adding to the guide (as shown in Figure 6) [16]. Another class of contributor consists of teachers and less tech-savvy people wanting to contribute ideas to the guide.

To reduce the barriers for intermediate contributors, an improved workflow was designed. The main focus of the workflow is application for easily editing CSFG content. Figure 7 gives an overview of the workflow (again using BPMN), which could be achieved with the introduction of an editor application. This will be the focus of the following sections.

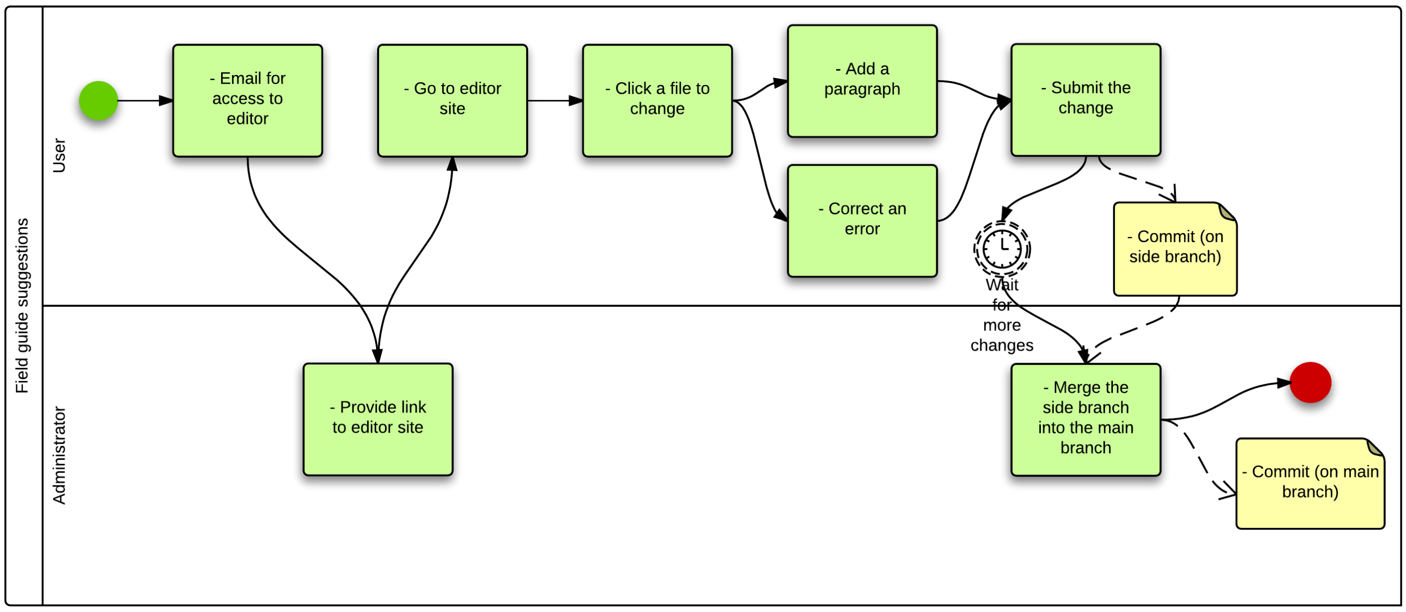


Figure 7: Workflow for a contributor using an editor application to commit to the CSFG (BPMN)

6 Building on an existing application

Using another project as a starting point for an editor is much more realistic than developing one from scratch due to the complexity. The application could easily be shared with potential contributors, and avoid flooding . Specifically, a web application would be preferred, given the web-based nature of GitHub connectivity.

For this application, we chose an existing Markdown editor to extend. The editor should meet as many of the requirements of the ideal application as possible. Requirements were elicited in a collaborative brainstorming meeting.

6.1 Requirements of a CSFG editing program

The most significant requirements of the program are given informally below. Because no application meets all these requirements, the chosen base program would need to be extended to meet unmet requirements (such as custom CSFG rendering).

- Plain text or rich text Markdown editing is mandatory, with an acceptable amount of features expected of a text editor [6].
- Rendering of the CSFG Markdown text (preferably in real time) is another essential requirement. This allows contributors to get a visual grasp on what their content will look like.
- Importing of GitHub files would allow contributors to work on the most recent CSFG files, rather than needing to download or be sent the files they want to work on.
- Like all material used with the CSFG, the editor should be open source.
- The ability to push changes to the GitHub repo is also required, to avoid these contributors the hassle of executing their own git push.
- Authentication using a pre-determined GitHub account is another requirement, so that contributors do not need to go through the effort of creating a GitHub account just for small changes.

6.2 Choice of base application

Initially, two open source Markdown editors were identified for possible extension. These were Dillinger [7] and StackEdit [8]. These applications exceeded the requirements for rendering and editing plain text Markdown, the two have different types of connectivity with other platforms. Both connect to GitHub, but Dillinger has a single file import feature, while StackEdit allows commits to be made to a GitHub repository. Dillinger is written using the the less familiar AngularJS Javascript framework, so StackEdit was determined to be more accessible, and therefore easier to extend overall.

Unfortunately, StackEdit was found to be too difficult to extend. The app and the new CSFG system had fundamental conceptual differences in how Markdown parsing was handled at a coding level. The JavaScript and Python disparity made converting even more

[6]https://en.wikipedia.org/wiki/Text_editor#Typical_features
[7]https://github.com/joemccann/dillinger
[8]https://github.com/benweet/stackedit

complicated, and not an effective use of time. StackEdit performed a chain of pre-processing regex replacements for various parsing operations, and in contrast to the single regex passes done by the CSFG system for each syntax item to be parsed.

To improve similarity with the CSFG, and to enable better synchronisation between the two programs, we decided to require a Python application as a base editor. Several popular Python Markdown editors were examined; the Niw Markdown Editor, ReText, and Markie. Niw was built with Python 2 by a single author, as was Markie, and both had not been updated for several years. ReText is an active project in GitHub with multiple contributors, so it was chosen as a follow-up option to StackEdit. While it lacked the flashiness of the StackEdit webapp, the ease of development made up for the user interface trade-offs.

6.3 Tools and libraries used during development

Three main tools were used to assist development on ReText. The Pycharm IDE for Python was used for development. It features a wide range of developer tools and effective code navigation. Changes to the code were stored using Git, to help track local changes and restore older versions when needed. Git was also used to fetch parts of the CSFG system that needed to be integrated.

Postman, a tool for testing APIs, was used to gain an understanding of the GitHub API. Postman enabled RESTful requests to be sent to the GitHub API. The request syntax and responses were compared with the calls being made by the Python GitHub connector libraries that were trialled for the app. This helped immensely when debugging various library functions.

A local Git repository was used to maintain a version history of the additions and changes to the base ReText code.

GitHub suggests several Python connector libraries that can be used to make calls to the GitHub RESTful API. However, all except the agithub library were dropped for various reasons. Pygithub3 was incompatible with ReText because of its implementation in Python 2. None of PyGithub, libsaas and Github3py had been updated with the recent convenient 'Commits' API feature. The implementation of agithub (the Agnostic GitHub API) means it is 'agnostic' of the actual API itself, and will function even if changes are made to the API in the future. This made agithub a useful choice.

ReText makes use of PyQt, the Python bindings to Qt, a popular cross-platform application framework [9]. The PyQt afforded the application a low barrier to development by

[9]http://wiki.qt.io/About_Qt

encouraging sensibly structured GUI code using its Widgets module. It also has easy-to-use internationalization and localization features.

6.4 Features implemented in ReText

Several features were implemented in ReText in support of the project goals. ‘ReText’ refers to the modified application from this point forward.

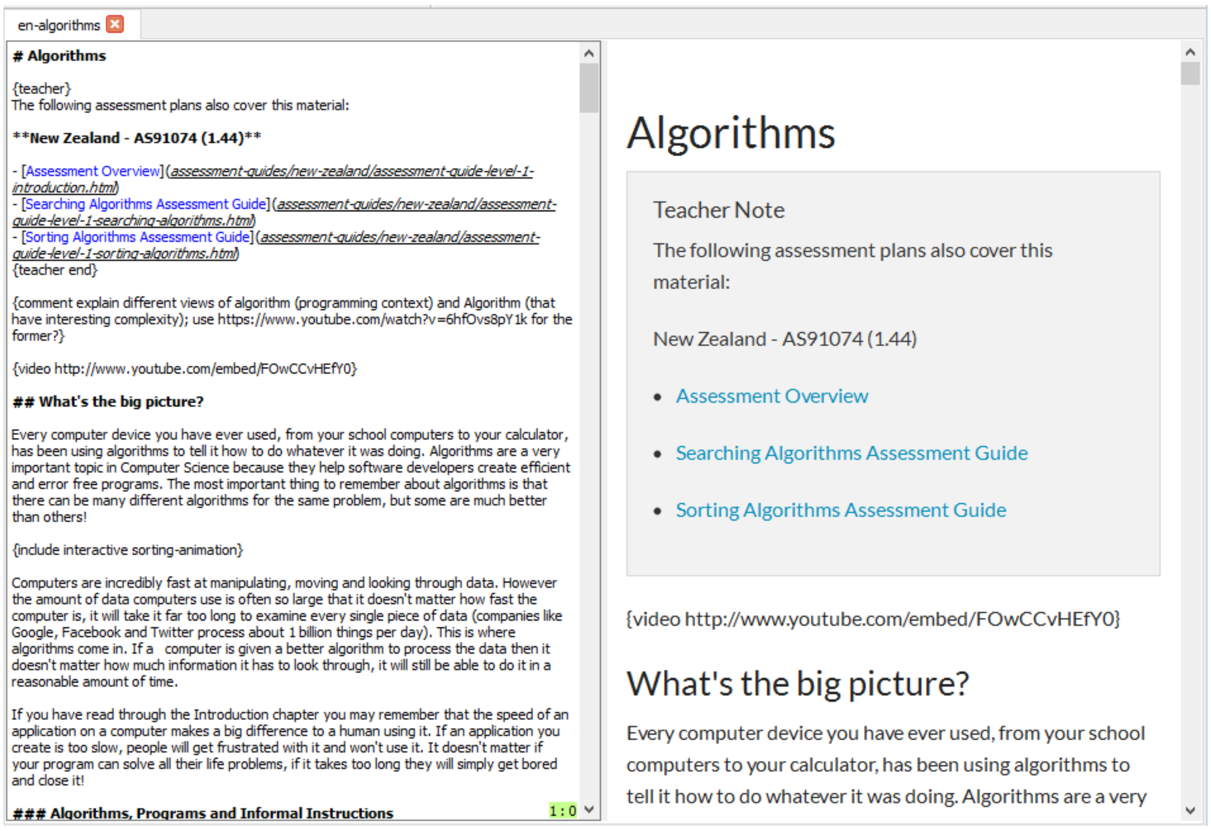
Firstly, functionality was added to connect to a GitHub repo and load in multiple chapter text files. Using the GitHub API, a modal list is loaded with files from a predefined branch of the CSFG repo. Users can select and open multiple files for editing, at which point the file contents are loaded from GitHub.

After editing a file, the user can press a ‘Commit’ button. This prompts them for a descriptive commit message. Upon submitting, a commit is sent back to the CSFG repo via the GitHub API. Here it waits to be merged with the guide-content branch (used for chapter edits). Obviously network connectivity is required for these two GitHub operations.

To perform these API operations, we use a generic ‘CSFG contributor’ account to generate a GitHub OAuth authentication token. This means application users don’t need to create a GitHub account in order to make these changes, as planned.

Another key feature was added; live previewing of the CSFG-specific Markdown. The parsing rules from CSFG generator were integrated with ReText’s own live Markdown parsing and rendering. By removing the parsing of video, file, and interactive insertion points, the parsing is fast enough be useful in real time ($\tilde{2}$s).

The CSS styling of the CSFG has also been applied to live previewing, so that users can really see how their changes will look on a live page. This is hugely valuable for novice users. Figure 8 shows what a user would see while editing a CSFG chapter with live previewing. ReText also renders the mathematical formulae from the CSFG correctly using its WebKit renderer.



```
# Algorithms

{teacher}
The following assessment plans also cover this material:

**New Zealand - AS91074 (1.44)**

- [Assessment Overview](assessment-guides/new-zealand/assessment-guide-level-1-
introduction.html)
- [Searching Algorithms Assessment Guide](assessment-guides/new-zealand/assessment-
guide-level-1-searching-algorithms.html)
- [Sorting Algorithms Assessment Guide](assessment-guides/new-zealand/assessment-
guide-level-1-sorting-algorithms.html)
{teacher end}

{comment explain different views of algorithm (programming context) and Algorithm (that
have interesting complexity); use https://www.youtube.com/watch?v=6hfOvs8pY1k for the
former?}

{video http://www.youtube.com/embed/FOwCCvHEfY0}

## What's the big picture?

Every computer device you have ever used, from your school computers to your calculator,
has been using algorithms to tell it how to do whatever it was doing. Algorithms are a very
important topic in Computer Science because they help software developers create efficient
and error free programs. The most important thing to remember about algorithms is that
there can be many different algorithms for the same problem, but some are much better
than others!

{include interactive sorting-animation}

Computers are incredibly fast at manipulating, moving and looking through data. However
the amount of data computers use is often so large that it doesn't matter how fast the
computer is, it will take it far too long to examine every single piece of data (companies like
Google, Facebook and Twitter process about 1 billion things per day). This is where
algorithms come in. If a   computer is given a better algorithm to process the data then it
doesn't matter how much information it has to look through, it will still be able to do it in a
reasonable amount of time.

If you have read through the Introduction chapter you may remember that the speed of an
application on a computer makes a big difference to a human using it. If an application you
create is too slow, people will get frustrated with it and won't use it. It doesn't matter if
your program can solve all their life problems, if it takes too long they will simply get bored
and close it!

### Algorithms, Programs and Informal Instructions
```

Algorithms

Teacher Note

The following assessment plans also cover this material:

New Zealand - AS91074 (1.44)

- Assessment Overview
- Searching Algorithms Assessment Guide
- Sorting Algorithms Assessment Guide

{video http://www.youtube.com/embed/FOwCCvHEfY0}

What's the big picture?

Every computer device you have ever used, from your school computers to your calculator, has been using algorithms to tell it how to do whatever it was doing. Algorithms are a very

Figure 8: Customised live previewing of the Algorithms chapter in ReText

6.5 Evaluation and initial feedback

The ReText application was examined and subjectively evaluated by the lead CSFG developer. The feedback was mostly positive (supporting the app's fulfilment of its requirements), but some improvements to the app were suggested.

Rather than making commits to a separate repo branch, which would require monitoring and periodic merging, ReText should make GitHub pull requests via the API instead. This fits with the workflow that the CSFG developers have adopted, and encourages case-by-case merging of changes via ReText.

Default authentication improved ease of access for contributors, by not requiring them to create a GitHub account. However, being able to view who made a change was seen as more valuable. Logging in with a personal GitHub account would allow new contributors to take ownership of their changes, and have their work attributed to them. It would increase the connection people had to the project, and increase community involvement as GitHub's contributor list for the repo grew larger.

Non-chapter files that someone might want to edit cannot be modified using ReText; it only loads chapter-level Markdown files. However, a user might want to edit one of the school curriculum guides. ReText could be relatively easily extended to support this .

6.6 Further limitations and future work

Because ReText development was started during the later part of the project, there were limitations in the implementation.

Updating ReText when the CSFG parsing rules or styling change is currently inconvenient.

The generic OAuth token used to authenticate over the GitHub API poses serious security issues. Since the token is hard-coded into the application, anyone with the application could obtain the token (by decompiling the packaged application's bytecode, for example). This is equivalent to credential theft; the attacker could use the token to perform unintended or excessive API access, or to manipulate data. The risk would be limited by the limited privileges of the generic account, but some annoying branch damage could still be caused.

More importantly, the OAuth token could violate the GitHub terms of service if the application is passed to more than a few people. The terms forbid the sharing of GitHub accounts between individuals. To prevent this breach from occurring, ReText should be extended to require users to log in with their personal account, as mentioned above. This would need to implemented as securely as possible.

The ReText interface has been modified very little from its original look. It would be beneficial to reduce the complexity of the interface by removing features which are irrelevant or not useful for our purposes, such as the ability to switch the editor to parse reST syntax.

Currently ReText requires instructions for a novice user to use the app as we intend them to. They need to be made aware of the functions on the commit and import features, and enable live previewing manually. In future, live preview should be enabled by default for loaded documents, and instructions, highlighting, or an animated tutorial could be used to demonstrate how the app is best used.

Beyond the ReText, the project had other limitations. The decision to adopt a simple translation mechanism has incurred technical debt. As more of the CSFG is translated, it may be necessary to completely restructure the way translations are stored. This might be to done improve localisation support.

In general, this research presented extremely optimistic system use cases. At the time, little consideration was given towards misuse cases and potential security threats to the system. Future work should be done to develop threat models to the new system, to better understand the vulnerabilities of the new system.

7 Conclusions

Open source interactive textbooks face a wide range of problems. These problems range from literary challenges such as writing coherency, to security challenges, such as mitigating the theft of user data. They require deliberate development of strategies to support open-source-ness. The new CSFG system we have presented meets the requirements that we crafted. The positive feedback from the authoring and development team has validated out decisions, and the system can adapt and grow to meet future needs.

The difficulties of translation management have been investigated to a useful degree. This will assist translation considerations and decisions for the CSFG in future.

The modified ReText application has been an effective proof of concept for how novice contributors could be able to positively impact the guide. The numerous design flaws in the application all have practicable solutions, and have not incurred large amounts of technical debt.

References

[1] T. Bell, C. Duncan, S. Jarman, and H. Newton, "Presenting computer science concepts to high school students," *Olympiads in Informatics*, vol. 8, pp. 3–19, 2014.

[2] B. N. Miller and D. L. Ranum, "Beyond pdf and epub: toward an interactive textbook," 2012.

[3] B. Miller and D. Ranum, "Runestone interactive: tools for creating interactive course materials," 2014.

[4] R. J. Ross and M. T. Grinder, *Hypertextbooks: Animated, Active Learning, Comprehensive Teaching and Learning Resources for the Web1*, vol. 2269 of *Lecture Notes in Computer Science*, book section 21, pp. 269–283. Springer Berlin Heidelberg, 2002.

[5] C. A. Shaffer, T. L. Naps, and E. Fouh, "Truly interactive textbooks for computer science education," in *Proceedings of the Sixth Program Visualization Workshop*, pp. 97–103.

[6] T. Bell and M. Stenfert Kroese, "The computer science field guide," 2015. [Online] Available: `http://nzcommons.org.nz/the-computer-science-field-guide/`.

[7] M. A. Davis and B. Blossey, "Edited books: The good, the bad, and the ugly," *Bulletin of the Ecological Society of America*, vol. 92, no. 3, pp. 247–250, 2011.

[8] R. Lewis, "Books with multiple contributors present multiple editing challenges," 1996. [Online] Available: `http://www.the-scientist.com/?articles.view/articleNo/17899/title/Books-With-Multiple-Contributors-Present-Multiple-Editing-Challenges/`.

[9] M. Kroes, "Creating the new etextbook," 2013.

[10] E. Fouh, V. Karavirta, D. A. Breakiron, S. Hamouda, S. Hall, T. L. Naps, and C. A. Shaffer, "Design and architecture of an interactive etextbook âĂŞ the opendsa system," *Science of Computer Programming*, vol. 88, no. 0, pp. 22–40, 2014.

[11] T. Bell, B. T. Wada, S. Kanemunu, X. Xia, W. Lee, S. Choi, B. Aspvall, and A. Wingkvist, "Making computer science activities accessible for the languages and cultures of japan, korea, china and sweden," in *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education, SIGCSE 2008, Portland, OR, USA*, 2008.

[12] A. Aharoni, “How does wikipedia handle page translation?,” 2015. [Online] Available: http://www.quora.com/How-does-Wikipedia-handle-page-translation.

[13] T. W. Foundation, “Translatewiki faq,” 2015. [Online] Available: https://translatewiki.net/wiki/FAQ.

[14] “Wikimedia servers,” 2015. [Online] Available: https://meta.wikimedia.org/wiki/Wikimedia_servers.

[15] V. Driessen, “A successful git branching model,” 2010. [Online] Available: http://nvie.com/posts/a-successful-git-branching-model/.

[16] T. Matthews, “Encouraging contributions to online communities,” 2012. [Online] Available: http://www-10.lotus.com/ldd/lcwiki.nsf/dx/Encouraging_contributions_to_online_communities.