UNIVERSITY OF CAPE TOWN

DEPARTMENT OF COMPUTER SCIENCE

COMPUTER SCIENCE HONOURS FINAL PAPER 2016

Title: Optimising Mathematical Content Creation and Management Systems for Dig-it

Author: Nathan Begbie

Project Abbreviation: DIGIT

Supervisor(s): Dr Melissa Densmore

Category	***Min***	***Max***	***Chosen***
Requirement Analysis and Design	0	20	15
Theoretical Analysis	0	25	0
Experiment Design and Execution	0	20	0
System Development and Implementation	0	15	15
Results, Findings and Conclusion	10	20	15
Aim Formulation and Background Work	10	15	15
Quality of Paper Writing and Presentation	10		10
Quality of Deliverables	10		10
Overall General Project Evaluation (*this section allowed only with motivation letter from supervisor*)	0	10	
Total marks		**80**	**80**

Dig-it 2.0: Optimising Mathematical Content Creation and Delivery

An Investigation into Tools for Creating and Optimising Mathematical Content

Nathan Begbie
University of Cape Town
18 University Avenue
Cape Town, South Africa
bgbnat001@myuct.ac.za

ABSTRACT

This project presents the development of tools for the creation and optimisation of mathematical content for Dig-it. Dig-it[1] is a mobile-first, online math-challenge platform for high school students. Dig-its current tools and workflows were found to be inefficient and costly. Given the cost of data and the importance of user experience in fast loading sites, it is also important that content delivered to users be optimised. This project was done in partnership with Praekelt.org, the implementers of Dig-it, to create a tool for the creation of mathematical content that could be used on their platform and optimise the content that was delivered to users. This project resulted in the creation of a browser-based tool that allows content creators to replace the use of their current tools used to create mathematical content. The creation of tools for automated content optimisation of content was successful in reducing page-size by more than half. These tools were integrated into a fully functional prototype which received high usability ratings and overall met the clients wishes.

Keywords

Equation Editor ,WYSIWYG, Content Creation, Content Optimisation, Mobile-first

1. INTRODUCTION

This project arose from a request by Praekelt.org[2], to assist them in optimising their mobile-first, math-challenge platform, Dig-it. Praekelt.org is a not-for-profit company focused on mobile platforms in health, job-creation and education.

[1]http://www.praekelt.org/digit/
[2]https://praekelt.org

This paper represents information relating to tools that were built for Dig-it, specifically for the creation and optimisation of content for users. The tools aim to assist in the creation of mathematical content within a web browser. This would replace Dig-it's current tool for mathematical content creation, Microsoft Word[3],in order to save the data in a structured, meaningful format. Assuming this data could be created and stored in a meaningful format, the second part of the project investigated to what extent the size of the content could be reduced, in order to reduce the cost of using the platform to users and to improve user experience. [4]

1.1 Project Significance

The current platform had a number of issues which the Dig-it team asked us to investigate and create solutions for. The tools aim to reduce work-flow complexity, replace existing tools that prevent optimisation, storing meaningful data as a single source of truth and optimising that content so that the content consumed by the user is as small as possible.

These tools were created in parallel with a management platform, created by Ethan Marrs. These tools were integrated at the end of the project, to create a completely functional prototype. The platform content management is not discussed in detail in this paper, but references are made to the platform as a whole. It is hoped that these tools and improvements will demonstrate a feasible progression of the Dig-it platform that Praekelt.org can use as a framework for updating their current platform.

1.2 Project Aims

The aim of this project was to optimise content creation, management and delivery. This part of the project in particular had two aims.

- create a tool that can replace Word as a tool for creating mathematical content and storing data as LaTeX.
- The second aim,was to create an automated tool to convert the stored data into mobile-first web pages and

[3]https://products.office.com/en-za/word
[4]This paper refers to supplementary materials that can be accessed via the UCT Computer Science online publications website: http://pubs.cs.uct.ac.za/. The abbreviation for this project is DIGIT. All references made to the online appendices in this paper refer to these materials.

minimise the size of the content that students would consume.

A number of meetings identified and clarified issues identified by the Dig-it team, in order to identify why these features were required and some of the more specific features that were needed. Thus we set out to investigate whether we could develop a tool that offered a What You See Is What You Get (WYSIWYG) equation editor that stored the underlying data as LaTeX, as well as text and images, as well as how to reduce the page load size to users.

1.3 Structure of Report

This report will provide some background and relevant literature to some of the topics in this paper. It will then present the design and implementation of the tools, followed by testing of these tools and an analysis of the results. This will be followed by a conclusion and brief discussion on future work that this work could generate.

2. BACKGROUND

2.1 Dig-it as it exists today

Dig-it is a mathematics mobi-site aimed at high school students. It supplies them with questions on a weekly basis and rewards them with airtime if a certain percentage of questions are answered correctly. Dig-it currently operates on a small scale with roughly 1500 students from Grades 10 to 12. In the next year, Dig-it will be scaling to a national level of over 100 000 students in 3 provinces.

In order to scale, it is important that the platform be managed properly. Unfortunately the platform faces a number of issues, in particular with how content is managed on the platform. This is partly influenced by the dependence on external tools to create mathematical content. The current workflow for content creation is that a question is created in a Word document, using text, images and Word's WYSIWYG equation editor to create equations. The text and images are then copied and pasted into a web-form, while screenshots are taken of the equations and uploaded to the web-form. The Dig-it team's reliance on this tool and this unwieldy, time-intensive and unoptimised process was identified this as one the main blockers to optimising Dig-it's work flows.

2.2 WYSIWYG Tools

Miner [18] points out the difficulty in using equations in an online context. Various different standards and methods mean that there is no universal equation language for the web. While there are various methods for structuring mathematical content, it is far more intuitive for users to interact with equations in a natural way. In discussing WYSIWG editors, Van der Hoeven [26] point out the steeper learning curve with purely text based structured text and lists controllability, transparency and readability as advantages of the WYSIWG editor. By using a WYSIWG editor, a person can intuitively create correctly typeset equations, without the need for further specialised knowledge and training. Thus, by relying on a WYSIWYG tool, Dig-it can reduce the barrier for entry for those it employs to create content.

2.3 Content Size and Data Costs

Performance of mobile sites is important for a number of reasons. Users will leave or stop using your site if they perceive it as slow to load [11]. Chen *et al.* [7] note that if page load times are high under low bandwidth conditions, users are more likely to abandon the task they set out to do. It will also decrease the number of returning users [5]. In a South African context, we face some of the highest data charges in the world, relative to income [12]. Dig-it is used by high school students, a demographic with very little purchasing power. It is important that the site cost them as little as possible to use, as it would otherwise diminish the value of the airtime incentive for continued practice. Thalia [12] notes that latency on mobile platforms has a large impact on load times. Latency is exacerbated by multiple requests for resources by the browser [9]. Thiagarajan *et al.* [24] note that unoptimized web pages also impact the battery usage of mobile devices. The larger the size of page size, the longer it will take to load and the more expensive it will be and the user experience. This project will thus use the size of content created as a proxy for user experience and price of content. It will attempt to reduce the size of the content as much as possible.

3. DESIGN

3.1 Design Methodology

It was important that a design methodology be used in order to maximise the usability and functionality of the end product. With this in mind, user-centred design was a central methodology to this project.

3.1.1 User-Centred design

User-centred design (UCD) is a method for involving the user in the design process [1]. The level of user involvement varies, but what is important is that the intended user provides insights into the product or the issues driving its creation and is used as an important consideration when making decisions. This can help to eliminate the biases of a single designer, increase product ownership and increase the relevance of the end-product for the user. UCD can range from initial requirements questionnaires to collaborative design and consultation throughout the process [27]. This project held a number of investigative meetings with the Dig-it teams and users of the current platform and continued to meet regularly with the intended users throughout the project, to receive feedback on ideas, designs and implementations. Users were also central to evaluating whether the design was effective, which is discussed later in this report.

3.2 Requirements Analysis

3.2.1 Initial Requirements

Our client had a number of initial requests about what the tools they wanted should accomplish. In order to clarify these requirements, a number of interviews with the Dig-it team were conducted, in which it was established who would be using the tools and clarification on a number of issues. There were also requirements added as the project progressed, due to the agile nature of the project. The requirements for the mathematical, content-creation tool were as follows:

- Content Creation Tool must be browser-based

- The creation of mathematical content should be done in an intuitive manner, replicating Microsoft Word's WYSIWYG equation editor.

- The tool should be able to replicate Microsoft Word's functionality, in terms of the content produced.

- Types of content could be divided up into text, equations Latex and Images

- The content should be structured in blocks, using SirTrevorJS[5] as an example of how to implement this.

- For equations, the underlying data store should be LaTeX.

- The content creator should be able to see a preview of the work that they had created

In terms of content delivery and optimisation, the requirements were as follows:

- Investigate ways in which mathematical content could be delivered to the user

- Convert stored LaTeX, Images and Text into HTML (a format that is understood by the browser)

- The page size should be as small as possible

In terms of the project as a whole, there should be at least 80% test coverage for the code and the creation of documentation to facilitate understanding of the project.

3.2.2 Current Code-base Analysis

A decision was made early on in the project, that we would create a new project, in order to avoid the unnecessary time that it would take to familiarise ourselves with the current code base and the potential pitfalls that may arise from attempting to engage with the users. However, the old code was still studied in order to understand how the current system works, in order to avoid changing functionality unnecessarily. It was also decided that as much as possible, my project partner and I would work in parallel and integrate our tools at the end of the project, to avoid any dependencies. It also became apparent that the initial decision by Dig-it, to use a particular open source WYSIWYG equation editor had not been researched in depth, necessitating research into alternatives, evaluation and selection of the appropriate tool before tweaking, integrating and enhancing it in relation to the rest of the tool.

3.2.3 Use Case Analysis

A number of interviews were conducted to understand the roles of those who would be using the tool and what they would be required to do, particularly in the context of structured data. The two identified roles of those who would use the tool, in the broader context of the platform was the content creator and content moderator. They both had identical use cases. These use cases can be seen in the Use Case Diagram in Figure 3.2.2.

[5]http://madebymany.github.io/sir-trevor-js/example.html

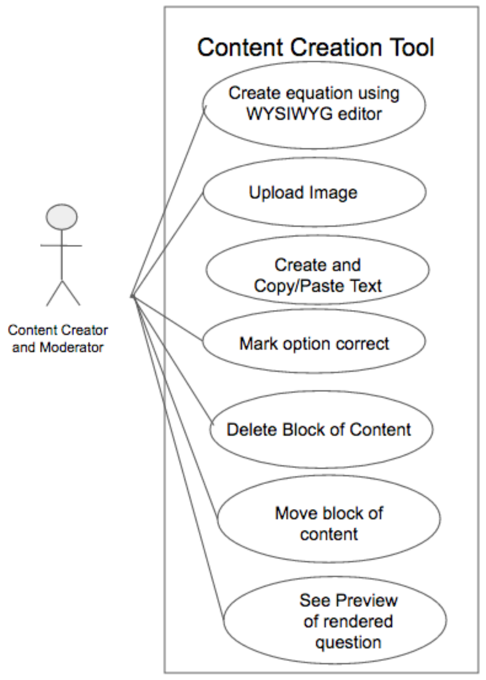

Content Creation Tool
Create equation using WYSIWYG editor
Upload Image
Create and Copy/Paste Text
Mark option correct
Delete Block of Content
Move block of content
See Preview of rendered question
Content Creator and Moderator

Figure 1: Use Case Diagram for the Content Creator and Moderator

3.3 Evaluating Equation Editor Libraries

The initial requirements of the project involved the use of the open source EquationEditor[6] that could operate in the browser. The initial project brief specified that the Equation Editor tool would be used. However in the initial stages of the project, we decided to evaluate whether there were other libraries that could be used. A week was designated to exploring the code and estimating which elements would be required for the porting of this code into a usable equation editor. Although not a direct design consideration, it was important that this work was completed before considering other architectural aspects, as selection of this tool had various repurcussions in the design process. We used a similar framework to Kennedy[15], in evaluating the use of an OSS project. These were fulfilling functionality, continuing development, community support and documentation. MathQuill[7] was identified as another WYSIWYG equation editor that worked in a browser. It was found that MathQuill was a more suitable editor based on a number of criteria.

3.3.1 Functionality and additional work

Both editors fulfilled the WYSIWYG and worked in the browser. However EquationEditor's, had not fully implemented transforming the data into a latex equation. This would have increased the work load of the project. MathQuill fully implemented transforming the data to Latex, but did not have a toolbar that would allow a user to input information, which EquationEditor did. MathQuill allowed for an easy implementation of this toolbar however.

3.3.2 Continuing Development

[6]https://github.com/camdenre/equation-editor/
[7]https://http://mathquill.com/

Continuing development on EquationEditor had a single contributor and development on the project had ceased on December of 2015. MathQuill had ongoing commitment to development by a number of core developers and had received corporate funding.

3.3.3 *Community and Support*

EquationEditor did not have any means of communicating with the developer, whereas MathQuill had a Slack Group and Internet Relay Chat (IRC) for communicating directly with developers. MathQuill also had a code of conduct for community members.

3.3.4 *Documentation*

MathQuill had ongoing documentation and tutorials for the code. EquationEditor did not.

3.4 System Architecture and Design

3.4.1 *Web Client and REST Framework*

The creation of a browser-based tool that could create structured, mathematical content lead to a number of decisions. Firstly, the toolbar would be a stand-alone, JavaScript web application. This means that the tool would be able to operate in the browser, independent of the back-end infrastructure. It would communicate with a back-end infrastructure using a REST API. This decision was made for a number of reasons. It would be easier to test as an independent unit. This decoupling increased the value of the tool as an open-source tool, for wider use outside of the Dig-it platform. This approach also favoured the parallel way in which the project was run, allowing us to work independently of the construction of the larger platform so as to reduce development dependencies. There were some drawbacks to this approach, namely that it increased the work necessary to integrate the toolbar with the final prototype, but this was made easier by the intentionally modular design of the code's functionality and setup could be completed with minimal code on the front end.

3.4.2 *MathContentCreator*

The optimisation and data handling section of this project was split out into its own Python package. This was done for a number of reasons. Firstly, this enables it to be used both in the python scripts used to test the effectiveness of the content optimisation, as well as integrating with the final prototype. Secondly, by making it a package, the install process becomes much simpler to incorporate code in different platforms. This adheres to the software engineering principle of high cohesion and low coupling, as well as code-reuse [13]. The code had a clear, defined purpose which reduces the coupling required. Thus it follows the pattern of separation of concerns. It was also helpful in forcing the classes to be completely self contained, although the code had to make some assumptions about storing files.

3.4.3 *Micro-service architecture pattern*

Micro-services are a modular way of separating code functionality into separate back-end 'services' and having them communicate via an API, rather than having a single entity responsible for the functionality, with communication being done by objects [19]. Micro-services were needed for 2 elements, converting latex to images and converting latex to

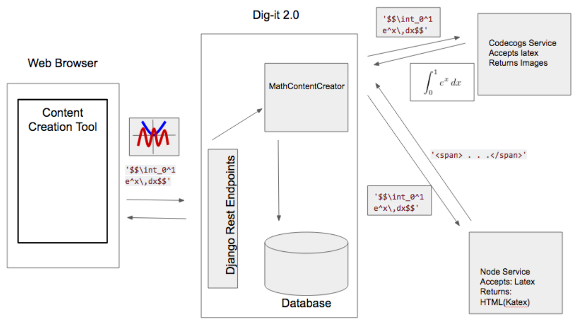


Figure 2: An architecture diagram depicting the overall design of the system

rendered KaTeX. The use of micro-services as an architectural approach stemmed from a number of things. Firstly, the rendering of KaTeX required a JavaScript environment to run in. Given that the Dig-it framework (Django) is python based, this meant that we could either run the JavaScript using a Python wrapper, or separate the JavaScript into a separate service, that Django could communicate with using HTTP. Upon consulting Praekelt.org engineers and the literature, the micro-services approach was decided upon for a number of reasons. Micro-services, when running as separate instances, were much simpler to install on different operating systems, easing the install process and reducing steps to use the product. The Python wrapper required different installation steps depending on the operating system in use.

This pattern encourages the separation of concerns - this means that single entity should be responsible for a single purpose. This is known as the single responsibility principle and help with decoupling software [19]. While the use of a Django 'monolithic' framework was used for the majority of the project, it is also important to be able to make changes to the internal workings of an entity, without needing to make changes to the rest of the code. This also encourages high cohesion and low coupling, which in turn increases the maintainability of the code. Another reason for using micro-service was a practical one; it reduced the amount of code that needed to be written. There already existed a public, available API for converting latex to images, and thus this would avoid rewriting existing functionality, which adheres to the Do Not Repeat Yourself principle (DRY) [13]. In the future, Dig-it may want to build out its own image creation service, but the public service currently used was sufficient for the purposes of it's project. The one drawback of this approach is that if the external services fail, this needs to be communicated to the framework and the delay handled by the system.

3.4.4 *Database Design*

During the database schema re-design, a number of design decisions had to be made on the storage of data, in particular for the mathematical content. In the current Dig-it system, data is simply HTML, stored as a string. The new system stores JSON as text, which acts as a single source of truth. The content of the JSON is then extracted and converted into 2 different data types. These two different types of data are to do with how the latex equations are displayed to the user. In the first case, the equations are converted to images, the images are stored and the images references along with the rest of the content are then stored in HTML, as a string. This approach has some drawbacks. Namely it violates the Normalisation rules of database design, in particular that the same data should not be replicated in a particular schema or database. However, performance is a valid reason for processing and storing the data in this way. If the database was normalised, then the content would be processed each time a client made a request for the content, increasing the processing that would need to be accomplished by the server and slowing response time for clients. This is especially important in light of Dig-it's need to scale to tens of thousands of users across South Africa. In order to mitigate the risk of incorrect duplication of data, safeguards would be built into code that manages the database, to ensure that there is no incorrectly replicated data.

3.4.5 User Interface Design

The design of the user interface was iterative and often involved redesigns or adjustments based on the input of the Dig-it team. This section explains some of the visual and user-experience decisions that were made and why.

While attempting to replicate the functionality of Word, the tool and in particular the user interface have a few advantages over it. In particular, the creation of equations has two enhancements. The first is that when a mathematical content area is clicked on, the toolbar for generating symbols and mathematical relationships automatically appears. This differs from MS Word, which often involves clicking through at least twice, to reach the equation editing tools. This design decision was motivated by the Principle of Least Effort, which makes suggests making frequent things easier to accomplish in the interface and unlikely things harder less obvious [8]. This was also motivated by the 'just-in-time' principle of user experience, which attempts to provide the user with the tool that they need, when they need it, in order to improve their experience [14].

The other enhancement is the use of typed shortcuts when creating mathematical content. Instead of stopping typing, finding the correct button on the toolbar and clicking it, users can type in shortcuts that automatically replace text with the appropriate symbol. For example, typing \sqrt will automatically create the $\sqrt{}$ symbol in its place. This allows those who use the tool often to create content faster and thus the tool accommodates both beginners and power users.

It was discovered during requirements gathering and feedback from the content up-loader that the content creator often creates content that is not suitable to be consumed on a small screen. For example, they may use a large graph that when re-sized to fit on a small phone screen, lost its legibility. This was not apparent to the content creator, because they were not prompted to think about the end-user in the context of the device they were using. This is known as encouragin user-empathy, which improves the end-product because it gets the creator to understand how an end-user might better use their product [29]. Thus in order to increase user-empathy, uploaded images are re-sized when they are uploaded and displayed, to a maximum width of 300 pixels. This should encourage the content creator to evaluate whether or not their work was legible, increasing the quality of the content created for students.

The structured and block-like nature of creating the content unfortunately hindered the intuitive creation of content and seemed counter to the benefits of a WYSIWYG equation editor. However this was a non-negotiable requirement from the client. In order to mitigate this dissonant way of creating content, a preview bar was added to the side of the toolbar, which allowed the content creator and moderator to see what the content would look like. This also helped them to understand the narrow frame in which the content would be delivered, further enhancing their user-empathy as alluded to in the previous paragraph.

As can be seen in figure 4.1.1, the colours used in the toolbar varied greatly from iteration to iteration. Complaints from the Dig-it team resulted in the choice of muted colours which allowed for a more pleasant user experience. However, a blue that contrasted to the muted colours was used to draw attention to the math toolbar so that users became aware when it appeared.

4. IMPLEMENTATION

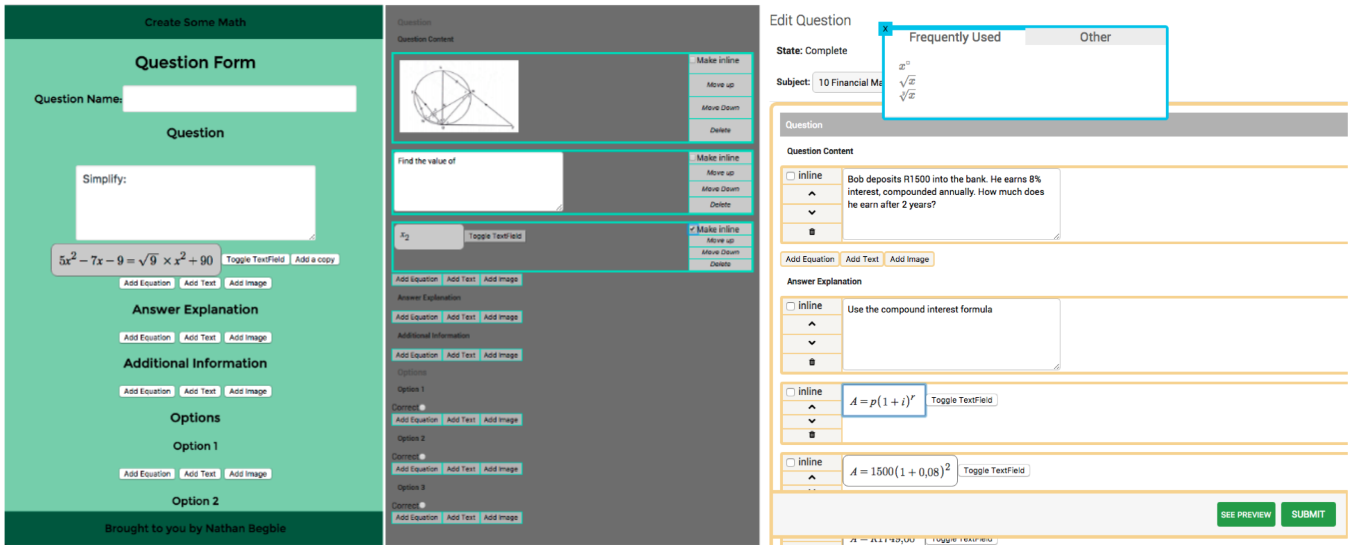
4.1 Software Development Methodology

4.1.1 Agile Development Methodology

This project used an Agile and iterative approach to development. This project involved solving a solution for a client, Praekelt.org. This meant that a system would need to be established in order to meet the clients demands. The selected methodology was Agile. While the client had set out requirements at the beginning of the project, which would have suggested that a waterfall method would have been more appropriate. This proved to be a good decision, as the requirements changed as the project progressed. Meetings were arranged specifically with the content uploader and the Dig-it project manager and service-designer for the Dig-it. While the content delivery system is able to be evaluated quantitatively, the requirements for the content creation system required feedback as to what it would achieve and some feedback on the visual appeal of the toolbar. Agile was suitable for a number of reasons; it allowed for the client to make requests or changes while development continued. We were unable to implement a SCRUM approach for tighter project management because meetings with the client were ad-hoc and due to constraint on both our side and the client's we were unable to have daily stand-ups and in some cases weekly meetings were unable to be held.

4.1.2 Iterative Development and User Centred Design

for analysing the functionality that would had to have been implemented, periodical meetings were arranged with the content up-loader. While it would have been ideal to include the content creators in this process, they were contractors and thus unavailable for the project. The content up-loader was given 3 questions randomly selected from the database and asked to use the tool to upload the content using the toolbar. The content up-loader then gave feedback on the tool, expressing need for missing functionality and ideas for improvements. The toolbar was also demo-ed at less regular intervals with the project manager, who gave



Simplify:

$5x^2 - 7x - 9 = \sqrt{9} \times x^2 + 90$

Find the value of

x_2

Bob deposits R1500 into the bank. He earns 8% interest, compounded annually. How much does he earn after 2 years?

Use the compound interest formula

$A = p(1+i)^r$

$A = 1500(1+0{,}08)^2$

Figure 3: Screen Snapshots of earlier prototypes, with varying functionality, used for iterative feedback.

input and made requests for features and refinements. This effectively took into account the demands of the client while balancing their time and capacity constraints as well as those of the development team. A summation of the requirements for the toolbar and how they were met are detailed later in this report.

4.1.3 Testing, Documentation and Maintainability

In order to ensure that the code functioned appropriately, unit testing within a larger testing framework was created both for the integrated Dig-it platform and the content creation tool. These test were created and run throughout development to ensure that the code functioned as expected. While test driven development (TDD) was proposed at the beginning of the project, it was only used in a small number of cases. The main reason preventing this was the constant iteration of the front end javascript code, which made testing very hard to anticiapate in advance.

Documentation was written for all major sections of this project, and appropriate code comments left where needed to explain functionality.

These tests and documentation should ensure that the project can be maintained and easy to continue development on, should the Dig-it team wish to do so.

4.1.4 Dogfooding

While working on the content creation tool development, one of the software engineering techniques in improving the quality of the software was to use the toolbar to convert sample questions in the platform, as detailed later in optimising the content. This process is known colloquially as 'dogfooding' in the software development industry [23]. The name is supposedly derived from an advertisement in which the CEO of a dog food company fed his own dog's his companies product. Is is a process in which a product is used as an end product by the developers who are currently working on it. One of the primary reasons for this process is to help developers find bugs in the software, that a quality assurance check may miss [10]. It also helps the developer empathise with their users. This process was helpful in aiding the detection of a number of anomalous behaviours that had not been considered in writing the code and thus unit tests had not detected them. It also led to a number of suggestions from the developers point of view as to improvements that could be made to the toolbar.

4.2 Development Framework and Methodology

Python and JavaScript were the programming languages used in this project. Below is a breakdown of the tools, libraries and packages used to implement each section of the project and motivations for their use.

4.2.1 Content Creation

Due to the requirement that the tool be browser-based, the content creation toolbar was created using HTML, CSS and JavaScript. Much of the functionality was implemented using the JQuery library[8]. The WYSIWYG equation editor was built around the MathQuill library[9]. In order to prototype the toolbar before integration with the Dig-it platform, a small Flask[10] application was written to host the toolbar. It was also used to create the files that were subsequently used for content optimisation, discussed in the next section.

4.2.2 Content Optimisation

Python was used for scripting purposes in order to extract, reformat and store data from the database. It was used to create the mathcontentcreator library that encapsulates the logic and processing of the formatted content. A Node[11] micro-service, using the Express library[12] to implement the server and KaTeX to render the HTML. CodeCogs[13] API was used to convert LaTeX to images. The Python Requests library to facilitate communication between the two services

[8]https://jquery.com/
[9]mathquill.com/
[10]flask.pocoo.org/
[11]https://nodejs.org/
[12]expressjs.com/
[13]https://www.codecogs.com/latex/eqneditor.php

it depends on for content conversions. For optimisation of the images, Wand, a python library and ImageMagick[14] were used. The python standard library was used to measure the file-sizes.

4.2.3 *Statistical Analysis and Graphing*

Python libraries, Numpy [15] and Scipy [16] were used for analysing and displaying the data. Matplotlib[17] was used for generating the graphs. These libraries are well-known and supported, and are used in a number of publications on statistical computing and analysis [20, 16, 21].

4.2.4 *Data Formats*

JavaScript Object Notation (JSON) was used to store the data. JSON is the primary data format for passing information around the web. Its use of key-names to reference objects and ability to store ordered data, meant that it was fully capable of representing structured data [3]. JSON is able to be translated directly into JavaScript objects, reducing the complexity of extracting and restructuring the data [2]. However it was not used to store raw data images (images converted to a long string of text), as it increased the storage size. Images were stored as separate files, with references to the images stored in the JSON. JSON was used as the primary means of moving data from the client-side JavaScript framework, to the back-end Python web server.

4.2.5 *Unit Testing*

Pytest[18] was used for testing Python code and Coverage was used to provide information on coverage. This project used QUnit for the testing of JavaScript. QUnit is a popular JavaScript unit testing framework, particularly with client-side code (JavaScript that runs in the browser) [22, 25]. Is is open source, and receives active development and maintenance.

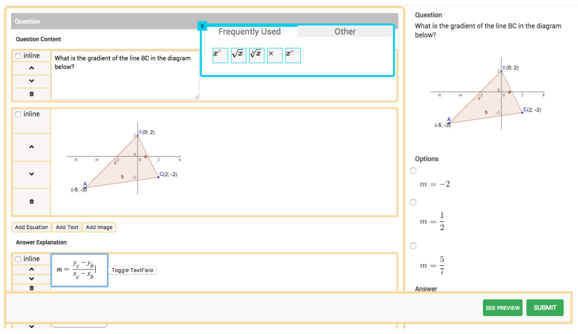
4.2.6 *Documentation*

For both the Python and JavaScript work, Sphinx[19] was used to document the classes and functions, as well as provide a broader overview of the code's functionality. Sphinx's auto-generation of documentation from code comments was particularly appealing and reduced the programmer workload.

4.3 Platform Integration

Once the content creation and optimisation tools were completed to a satisfactory degree, they were integrated with the Dig-it administrative, platform prototype, created by Ethan Marrs. This involved working with the Django[20] web framework, by creating REST API endpoints for creating and storing data. Furthermore, the content optimisation process was also integrated in order to fulfil end-to-end creation and consumption of mathematical content. within the platform

[14]https://www.imagemagick.org/
[15]www.numpy.org/
[16]https://www.scipy.org/
[17]https://matplotlib.org/
[18]doc.pytest.org/
[19]sphinx-doc.org/
[20]https://www.djangoproject.com/



What is the gradient of the line BC in the diagram below?

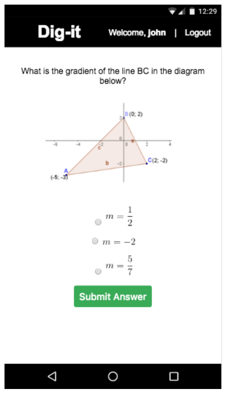
$m = \frac{y_c - y_b}{x_c - x_b}$

$m = -2$

$m = \frac{1}{2}$

$m = \frac{5}{7}$

Figure 4: Screenshot of the final, integrated content creation tool



What is the gradient of the line BC in the diagram below?

$m = \frac{1}{2}$

$m = -2$

$m = \frac{5}{7}$

Figure 5: Screenshot of the optimised content, served to a phone browser

4.4 Content Creation Tool and Screen

The final 'realisation' of the content creation tool was integrated with the Dig-it platform and was able to successfully store and retrieve data from the web-framework. Details about the design of the interface are given in

4.5 Student Screen

The student screen as used as proof that the content created by the tools and then automated, created legible and working content. Seen in figure 4.4, this is how the content is finally consumed by the students.

4.6 Development Environment

A text editor, Submlime Text 3[21] was used for writing the code, with the use of plug-ins to aid in the programming process.

4.6.1 *Version Source Control*

Git was used for distributed version control. Github was used for hosting and distributing the code. The Centralized Workflow model, as described by Chacon [6] was used to control the flow and integration of code, where code needed to be integrated.

4.6.2 *Project Management*

In order to keep track of tasks, what was being done and what was determined to be out of scope, Trello was used to

[21]https://www.sublimetext.com/3

keep track of these things.

5. TESTING AND EVALUATION

5.1 Unit Testing

Unit tests were written to ensure that code functioned appropriately. One of the initial goals of the project was that there by at least 80% code coverage. The use of code coverage tools was used with the testing frameworks to cover the required use cases and

5.2 User Evaluation

While an agile and iterative approach was adopted during the project, which enabled client feedback and user input, it did not allow for a quantifiable way to measure the usability of the finished product. In order to gauge the level of usability, a framework was adopted called the System usability Scale (SUS). It provides a simple way of allowing users to provide feedback in a measurable way, using a Likert scale in response to ten questions about the users' experience of using the product. Questions switch between positive and negative phrasing. The questions are as follows:

1. I think that I would like to use this system frequently
2. I found the system unnecessarily complex
3. I thought the system was easy to use
4. I think that I would need the support of a technical person to be able to use this system
5. I found the various functions in this system were well integrated
6. I thought there was too much inconsistency in this system
7. I would imagine that most people would learn to use this system very quickly
8. I found the system very cumbersome to use
9. I felt very confident using the system
10. I needed to learn a lot of things before I could get going with this system

5.3 Metrics for success

In order to gauge on a larger scale whether the tool creation had been successful, simply involved checking the list of requirements and whether the tool met these requirements.

5.4 Content Optimisation

5.4.1 Methodology for Evaluating Content Optimisation

This part of the project assumed that the data was represented in a structured format in JSON. It also assumed that equations were stored as latex and that images were already stored in a database of some kind. This section of the project would attempt to optimise the content sent to users. The client had requested that this be done in two ways, optimising for different users. The data was to be transformed from Latex to images for feature phones and rendered into

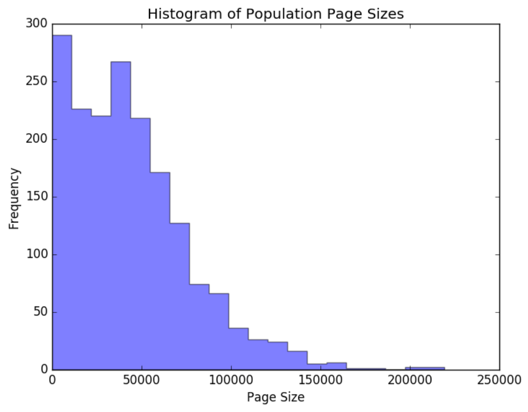


Figure 6: The distribution of the entire question data set

HTML using KaTeX for users on smart-phones. In order to measure the success of the optimisation process, it was first required to select a sample of existing content on the Digit platform. There are currently around 1800 questions on the platform, of which 1700 were relevant to this exercise. The first step was to receive a database dump of the content from Praekelt.org.

Due to the fact that this would require a dump from the production database, it was first scrubbed by Praekelt engineers of personal data of its users. A file dump of the static content (i.e. images and images of equations). A script was then written to extract the contents of each question, answer and answer options, stitch them together and store the images. The size of the questions was then measured using a script. These metrics could then be used to a baseline, to assess to what extent the size of the content had been reduced.

The second step in the process was to extract a sample of these questions to then convert into structured content and optimise. The questions are divided up into modules which for the most part, represent different mathematical topics. It was important that each of these modules be used in the sample, as some topics, such as geometry use images more intensively, while some, such as Algebra, use a number of equations. To give some modules more representation than others, would have skewed the results. While it would have been ideal to convert the entire set of questions, this was a labour intensive exercise, and a sample of a big enough size would be sufficient to demonstrate whether or not the change in size was significant. In order to select the sample questions, a script was written to go through each module and randomly select 3 questions from the sample.

This sample was then converted from text and images to text, latex and images, in a structured JSON format. As alluded to in the section 'Dogfooding' most of this conversion was done using the content creation tool, while the rest was done by manually transcribing images of equations to latex. The code would then be written, to accomplish two things. Firstly, to convert the structured JSON and unoptimised images into the two formats available for feature phones and smart-phones, and secondly, to optimise the content as much as possible, so that the size of the content was reduced. Finally, the optimised content would be measured in size and compared to the original content, in order to measure the degree of success.

5.4.2 *Statistical Method*

In order to prove that the results were statistically significant, a t-test would need to be performed. A t-test assumes that the data is normally distributed. This was found not to be the case. This can be seen intuitively by examining the distribution of all questions in figure 5.3. We also ran the D, Agostino and Pearson test [?, ?] on the data to check for normality, using the scipy library. Attempts to normalise the data did not work. Thus a non-parametric test was needed to determine whether the change in page sizes was statistically significant. Thus this required the use of the Wilcoxon Rank-Sum test [28]. This is a non-parametric test, meaning that the populations are not assumed to be normally distributed. This test would confirm whether the average page sizes from the samples were different enough, to be from different populations. This simply confirms that the page sizes were indeed made smaller, with statistical certainty.

6. RESULTS AND DISCUSSION

6.1 Content Creation

In order to evaluate the efficacy of the toolbar there were two methods of evaluation.

Firstly, did it meet the requirements set forth in the initial project requirements set out by the client?

Secondly, did it meet the requirements of those who would use the tool, in terms of usability?

The second question is discussed in the subsequent section.

The first requirement was easy to measure as it simply involved checking against the listed requirements.They were as follows;

1. The equation editor should work in the browser.

2. It should be a WYSIWYG editor.

3. It should be a 'structured' editor, such as SirTrevorJS.

4. The editor should be capable of creating the same content as the current tool, MS Word's MathType equation editor.

5. It should be able to store the underlying data as latex.

6. It should be able to integrate with the Django framework to fetch, edit and send the content to the server.

All of these requirements were met completely, with the exception of being able to completely replicate functionality for Word. It was discovered during the dogfooding process that MathQuill is unable to render hats. Hats are used to indicate an angle like so: $A\hat{B}C$. After investigation, it appears that this functionality will soon be added to the library[22]. This slight reduction in functionality and solution was accepted by the client.

[22]https://github.com/mathquill/mathquill/pull/646

6.2 System Usability Testing

The results of the SUS were overwhelmingly positive, with a score of 81.6. This is well above the average of 68 and is considered 'good' according to Brooke [4] because it is over 73. These results should be taken with a pinch of salt however, given that only three stakeholders engaged in this process. McLellan *et al.* [17] warn that the more experienced a user is with the platform, the higher the usability rating will be. Given that the users were developers and/or involved in providing feedback on the Dig-it prototype, these results may be positively skewed. Some more measured feedback was also given after the demonstration, particularly in regards to the content creation tool. One user pointed out the usability issues created by the structured flow of creating content and some of the buttons were described as unintuitive. However, this failed to diminish the glowing reviews that accompanied the demonstration and interaction with the final prototype, with one stakeholder exclaiming, “I love it”.

6.3 Content Optimisation Evaluation

6.3.1 *Converting latex equations to Katex*

The conversion of latex equations into katex failed to reduce the size of web-pages. Once the Katex library has converted latex to its HTML representation, it was still larger on average than the alternative of images, despite being text. Once the web assets necessary for rendering the content in the browser was accounted for, it was almost 3 times the size of the original web-page, on average. Thus it seems unlikely that this technology would warrant implementation, as images appear to be a more effective means of communicating mathematical knowledge, given the technological and monetary constraints of mobile consumption. The only counter point for keeping Katex was during feedback in which a stakeholder remarked that Katex appeared better, relative to text.

6.3.2 *Converting latex equations to images*

The optimisation for converting equations to images was successful. On average, for the test sample, the content was only 52% of the size of the original webpage size. After the optimisation of uploaded images, the page size was reduced to 41% of the original page size, on average. The Wilcoxon Rank Sum test confirmed that both of these results were statistically significant to a very high degree of certainty. More details of the results can be found in the online materials. This former result is however more important, as it suggests that the best way in which to reduce the page size is to change the underlying way in which the data is stored. This is significant because while it would be easier to simply re-size images in the current platform, it would be far more effective to replace the underlying data store with latex and automatically convert them into images.

6.4 General feedback

Throughout the project, we received feedback and positive criticisms about the platform. Overall the response to the platform was positive, but the Dig-it team did acknowledge that converting this prototype into production would take a large investment of time and resources. Thankfully this was not in the scope of the project and the client felt that the tools as functional prototypes were completely adequate.

Table 1: Average Page Size based on content optimisation type

Optimisation type	Average Page Size (bytes)	Percentage of original
Oringinal content	36431	100%
Katex without resized image	28565	78%
Katex with resized image	24448	67%
Converted Images	19105	52%
Converted Images with resized images	14965	41%

7. ETHICAL, PROFESSIONAL AND LEGAL ISSUES

This project required access to the current Dig-it database, which contained user data. It was imperative that user data not be distributed or leaked. The database was anonymised by Praekelt.org engineers and kept on a single device. The original data set is not included in the hand-in, to further ensure that there are no possible data leaks. The questions that were created, are also the intellectual property of Dig-it and so only the sample of questions used are uploaded, to verify the analysis of the data.

All interviews and surveys were completed by Praekelt.org employees. This did not raise any ethical issues, although some names are anonymised by request, in the meetings notes. Individuals are referred to by their roles at Praekelt.org or Dig-it.

8. CONCLUSIONS

In this section, the conclusions that this project has reached will be discussed.

This project aimed to create a web-based tool that could replace Word as a tool for creating mathematical content and that stores the underlying equation data as latex. Based on meeting the initial requirements and client feedback, this project has been successful. However, the client felt that there were still some improvements to be made, although they did not necessarily fall within the initial scope of the project. Some of these improvements are discussed in section 9. Given the small number of individuals who used and evaluated the content creation tool, further user testing should be conducted to better evaluate the overall usability of the content creation tool.

This project also aimed to create automated tools to convert structured mathematical content into a form that focused on mobile devices and was as lightweight as possible. Two methods were investigated for the displaying of equations on mobile devices; converting latex to images and rendering the content on client devices using the Katex library. The use of Katex gave poor preliminary results which lead to it not being incorporated in the final product, as it increased rather than decreased the page size. However, converting equations to images in a controlled, automated fashion was highly effective, reducing the page size to 52% of the original size on average. Page size was further increased due to optimising image size for phone screens. These results strongly suggest that Dig-it should implement these tools in order to improve their student's experience and save both parties money.

Furthermore, these tools were able to be integrated into a unified, fully-functioning prototype that received praise from the client, and received favourable scores in usability testing.

9. FUTURE WORK

This project would require further work before it is put into production as there are a number of processes which would need to be made more robust. There were a few feature requests that were made during the project that were defined to be out of scope of the project, given the time constraints. For the content creation tool, these include keyboard shortcuts and live or continuous question preview generation. For the content optimisation, automating the stitching together of vertically adjacent images to reduce page load time in high-latency environments, by reducing the number of HTTP requests that the browser has to make. It may also be useful to create or incorporate graphing tools within the platform that enable the creator to consider how content will look on a feature phone.

On a broader scale, given that stored data in the platform is now meaningful, the creation of 'programmable' questions, that can create a variety of possible questions, would be very interesting and a practical way of driving down Dig-it's cost per question created. This project has successfully laid the groundwork for this to occur.

10. ACKNOWLEDGEMENTS

I would like to thank our supervisor, Dr. Melissa Densmore, for her patience and guidance throughout this year. I would also like to thank Michelle Kuttel, our second reader, for her time. I would like to thank our client, Praekelt.org for their co-operation on this project. In particular, thanks goes to Lauren KotzÃl' for her enthusiasm and engagement and Milton Mdanda for his guidance. Finally and most importantly, I would like to thank my team member, Ethan Marrs, for his trust, help and dedication in completing this endeavour. It was much appreciated.

11. REFERENCES

[1] C. Abras, D. Maloney-Krichmar, and J. Preece. User-centered design. *Bainbridge, W. Encyclopedia of Human-Computer Interaction. Thousand Oaks: Sage Publications*, 37(4):445–456, 2004.

[2] A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers, Principles, Techniques.* Addison wesley, 1986.

[3] T. Bray. The javascript object notation (json) data interchange format. 2014.

[4] J. Brooke et al. Sus-a quick and dirty usability scale. *Usability evaluation in industry*, 189(194):4–7, 1996.

[5] J. Brutlag. Speed matters for google web search. *Google. June*, 2009.

[6] S. Chacon. *Pro Git*. Apress Distributed to the Book trade worldwide by Springer-Verlag, Berkeley, CA New York, 2009.

[7] J. Chen, L. Subramanian, and K. Toyama. Web search and browsing behavior under poor connectivity. In *CHI'09 Extended Abstracts on Human Factors in Computing Systems*, pages 3473–3478. ACM, 2009.

[8] M. Collan. Lazy user behaviour. 2007.

[9] I. Grigorik. *High-performance browser networking*. O'Reilly, Sebastopol, CA, 2013.

[10] W. Harrison. Eating your own dog food. *IEEE Software*, 23(3):5–7, 2006.

[11] L. Hogan. *Designing for performance : weighing aesthetics and speed*. O'Reilly, Sebastopol, 2014.

[12] T. Holmes. datamustfall âĂŞ how cellphone giants are milking you, September 2016.

[13] A. Hunt. *The pragmatic programmer : from journeyman to master*. Addison-Wesley, Reading, Mass, 2000.

[14] S. S. Intille. Ubiquitous computing technology for just-in-time motivation of behavior change. *Stud Health Technol Inform*, 107(Pt 2):1434–7, 2004.

[15] M. Kennedy. Evaluating open source software. Technical report, DTIC Document, 2010.

[16] W. McKinney. *Python for data analysis*. O'Reilly, Beijing, 2013.

[17] S. McLellan, A. Muddimer, and S. C. Peres. The effect of experience on system usability scale ratings. *Journal of Usability Studies*, 7(2):56–67, 2012.

[18] R. Miner. The importance of mathml to mathematics communication. *Notices of the AMS*, 52(5):532–538, 2005.

[19] S. Newman. *Building microservices : designing fine-grained systems*. O'Reilly Media, Sebastopol, CA, 2015.

[20] C. Pilon. *Bayesian methods for hackers : probabilistic programming and Bayesian inference*. Addison-Wesley, New York, 2016.

[21] W. Richert. *Building Machine Learning Systems with Python*. Packt Publishing, Birmingham, 2013.

[22] D. Sheiko. *Instant testing with QUnit : employ QUnit to increase your efficiency when testing JavaScript code*. Packt Pub, Birmingham, UK, 2013.

[23] F. Shipman. Managing software projects in spatial hypertext: Experiences in dogfooding.

[24] N. Thiagarajan, G. Aggarwal, A. Nicoara, D. Boneh, and J. P. Singh. Who killed my battery?: analyzing mobile browser energy consumption. In *Proceedings of the 21st international conference on World Wide Web*, pages 41–50. ACM, 2012.

[25] M. Trostler. *Testable JavaScript*. O'Reilly Media, Sebastopol, CA, 2013.

[26] J. Van Der Hoeven. Gnu texmacs: A free, structured, wysiwyg and technical text editor. *Le document au XXI-ieme siecle*, 39:40, 2001.

[27] K. Vredenburg, J.-Y. Mao, P. W. Smith, and T. Carey. A survey of user-centered design practice. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 471–478. ACM, 2002.

[28] F. Wilcoxon, S. Katti, and R. A. Wilcox. Critical values and probability levels for the wilcoxon rank sum test and the wilcoxon signed rank test. *Selected tables in mathematical statistics*, 1:171–259, 1970.

[29] P. Wright and J. McCarthy. Empathy and experience in hci. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 637–646. ACM, 2008.