Philip Yeeles

Nico: An Environment for Mathematical Expression in Schools

Computer Science Tripos

Selwyn College

May 16, 2012

Proforma

Name:	Philip Yeeles
College:	Selwyn College
Project Title:	Nico: An Environment for Mathematical Expression in Schools
Examination:	Computer Science Tripos, May 2012
Word Count:	TBC[1] (well less than the 12000 limit)
Project Originator:	P. M. Yeeles (pmy22)
Supervisors:	Dr S. J. Aaron (sja55), A. G. Stead (ags46)

Original Aims of the Project

The aim of this project was to develop an application to act as an aid in mathematical education for pupils in Year 5. The software was to provide a new graphical metaphor for calculation with an accompanying working environment, with the intention of eliminating some of the problems that makes handwritten arithmetic unclear in many situations.

Work Completed

I have successfully designed and implemented an application using the Clojure programming language that allows users to express calculations using a graphical notation. The software is able to generate an abstract syntax tree from the graphical notation, evaluate it and pass the results back to the application in approximately

[1] This word count was computed using the web interface to `texcount`, which can be found at `http://app.uio.no/ifi/texcount/online.php`.

10ms. As an extension to the project, a user study was conducted to evaluate the utility of the software.

Special Difficulties

- Learning the Clojure programming language
- Learning to choose appropriate mutable data structures of those that Clojure provides
- Learning to incorporate tests into the development cycle

Declaration of Originality

I, Philip Michael Yeeles of Selwyn College, being a candidate for Part II of the Computer Science Tripos, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose.

Signed

Date May 16, 2012

Contents

List of Figures

Acknowledgements

My thanks to Luke Church for his advice regarding user studies, and to Alistair Stead and Sam Aaron for their encouragement and patience.

Chapter 1

Introduction

The aim of this project has been to design and develop a notation and accompanying application to act as a learning aid for pre-algebra arithmetic in schools. Using Petre and Green's 'Cognitive Dimensions' framework, the project aims to increase the *visibility* of the notation, to reduce the number of *hidden dependencies* between units of notation, to reduce the *viscosity* of the notation and to make the flow of data obvious to the user. [12] I have successfully developed such a system, intended initially for pupils in Year 5 (though extensible, through the creation of alternative question sets, to other age groups), and, as an extension, conducted a user study to assess its utility.

In this chapter, I will discuss my motivations for choosing this project, the pros and cons of the handwritten system it is attempting to augment, the technical challenges involved in developing such a system and related work that has previously been conducted with similar goals.

1.1 Motivations

My motivations behind this project lay in the limitations of the handwritten approach to solving mathematical problems that I had observed both in my own learning and in my own teaching experience. What follows is an assessment of the technical challenges involved in this project, and an evaluation of the pros and cons of the handwritten method of performing arithmetic calculations according to the 'Cognitive Dimensions' framework cited above, and a discussion of the properties that a useful alternative notation should have.

1.2 Technical Challenges

Developing such an application comprises three main challenges: developing a back-end that is capable of creating, storing, editing, deleting, evaluating, and nesting calculations, an interaction-handler layer that is able to interpret user input and events occurring in the user interface, and a graphical user interface that is able to render calculations into the devised notation, and allow the user to perform operations upon the notation that affect the underlying calculation.

Such a project will require a strong foundation from which to construct a user interface, as this is a crucial factor in its success. An external library or libraries to provide GUI functionality would be suitbale. The fact that it is also being built with the intention of being deployed in schools, in which there is a great variety of computing environments, necessitates a portable system.

With regard to the representation and processing of calculations, a framework in which calculations are easy to store and manipulate, and in which evaluation can be delayed, would be advantageous.

I chose to use the Clojure language as it provided many features that would solve the problems described above. As a dialect of LISP, Clojure is a homoiconic programming language; a programming language in which code is represented as a data structure. This made passing around and performing operations upon calculations themselves, rather than just their results, considerably easier. A calculation can simply be represented as a piece of code, which can then be utilised as needed.

Clojure also provides the aforementioned libraries needed for effective GUI implementation. It runs on the Java Virtual Machine (JVM), which puts Java's considerable standard library at one's disposal, whilst still being able to program in a LISP. As the author is familiar with Java and the Swing GUI libraries, it was advantageous to be able to leverage this knowledge in designing the application's interface.

In addition to this, running on the JVM means that the software is suitably portable. A Java-based solution standardises the *Nico* experience across many environments.

1.3 Previous Work

There already exists a wide variety of educational software for mathematics, often in the form of 'games', in which mathematical problems are presented in the context of a computer game. Although such games have repeatedly been shown to be of benefit to the learner, a barrier to their widespread use in education can be the preconceptions of the users. [24] [9] Indeed, Bragg notes that

> "One barrier for the employment of games as a pedagogical tool may be possible negative attitudes held by students towards the likelihood of the games' effectiveness in assisting mathematical learning." [6]

This is reinforced in a later paper by the same author, where Bragg concludes that

> "It appeared that game-playing negatively affected attitudes" [7]

Although it was also concluded that such problems could be addressed by dedicating lesson time to explaining the benefits of game-playing as a pedagogical tool, an alternative approach, and one that is taken by this project, is to introduce computers into the classroom as a **tool**, rather than as a means of playing games. In the same way that calculators are used in mathematics lessons, it is the intention of this project to introduce a new tool to complement learning, to be incorporated into students' repertoires.

There also exist a few applications intended to represent calculations on a computer in novel ways. A relatively common approach to this has been to try to make on-screen calculations more like on-paper calculations.

Pi Cubed takes this approach by trying to make complex calculations appear as they would be written in an exam or exercise book [17]. *Soulver*, conversely, tries to achieve this by simulating 'back-of-the-envelope' calculations, whereby notes in English augment the calculation [18].

Another approach is that of the *Scrubbing Calculator* [26], which extends the *Soulver*-style environment by helping the user to solve equations by dragging values to increase and decrease them, showing how changing a value affects the overall result. Values can be linked by dragging a line between them, which means that they are two instances of the same value – hence dragging one changes the value at every location in which it appears.

This is a neat means of visualising equations, but it, too, is not intended for use in education, and still requires the user to be able to formulate some kind of equation. The *Scrubbing Calculator* is more a tool for facilitating algebraic understanding, as

opposed to arithmetic understanding; indeed, it is inherently a **calculator**, and so does not encourage thinking about how to work out the arithmetic parts of a calculation manually.

1.4 Summary

Existing educational ‘games’ for mathematics, although valid as a means of teaching, face obstacles from teachers’ and pupils’ attitudes toward them, as a result of their status as games, rather than learning tools. There exists software to aid in calculation and arithmetic by representing it clearly, but it is not intended for educational use, and often its purpose is to make on-screen calculations appear as one would handwrite them. A new application is proposed to be used as a tool to complement teaching, as one would use a calculator.

There is a niche for a tool for use in education that represents calculations in a visual manner, with a particular focus on making the method by which arithmetic problems are solved clear, whilst remaining not a calculator. This project aims to provide such an environment, in which the user can explore the many ways in which a problem can be solved using a novel graphical notation.

Chapter 2

Preparation

This chapter concerns the work that was completed prior to beginning the project proper. It comprises a requirements analysis, followed by a discussion of the prototyping process of the graphical notation to be implemented in the final application. Finally, there will be a brief examination of the tools used in the development of the project.

2.1 Requirements Analysis

2.1.1 Current System

There are a number of problems with handwritten, pre-algebra arithmetic that this project seeks to rectify. An analysis of the handwritten method according to the 'Cognitive Dimensions' framework follows.

24+35=59	24+35=59	24+35= ~~59~~ 49
12+48=60	12+48=60	12+48=60
59+72=131	59+72=131	~~59~~ 49+72= ~~131~~ 121
1+60=61	1+60=61	1+60=61
131+61=192	131+61=192	~~131~~ 121+61= ~~192~~ 182
(a)	(b)	(c)

Figure 2.1: Illustrating the hidden dependencies and viscosity inherent in pre-algebra, handwritten arithmetic. The original calculation is shown in (a), has its otherwise-hidden dependencies highlighted in (b), and is altered slightly in (c).

First of all, the fact that it is handwritten entails a high level of *viscosity*: it is difficult to make changes to a written calculation without sacrificing clarity. In particular, there is a lot of *repetition viscosity* involved in the modification of an existing piece of work; if a number is changed that is used in several calculations, then it is time-consuming to change it everywhere it appears in the working.

If several calculations are dependent upon each other, then this entails a lot of *knock-on viscosity* in recalculating each stage after changing the number. This is exacerbated by the *hidden dependencies* between chained calculations in handwritten arithmetic (*Fig. 2.1*).

The problem of *hidden dependencies* is made worse by the low *juxtaposability of notation* of the system; although the notation is quite *visible*, in that every calculation can be seen easily on the page, juxtaposing two sets of calculations entails considerable *premature commitment* on the part of the user, as components cannot easily be edited or relocated due to the system's high *viscosity*.

Whilst *viscosity* can be acceptable in some situations, it is harmful with regard to modification and exploration within a notational system, once again requiring a non-trivial amount of *premature commitment* on the part of the user. In many cases, it can actually quicker for the user to start all over again, as opposed to making the changes required to rectify their calculations.

Handwritten arithmetic does have one key advantage: it has a very low initial *abstraction barrier*. Other than learning the appropriate symbols for each operation and digit, and how to combine them, the traditional system of arithmetic allows a learner to begin using it almost immediately. Algebra, on the other hand, has a much higher *abstraction barrier*, requiring the much more abstract concept of a variable, rather than a set quantity, to be used effectively. Although it is possible to add abstractions to arithmetic by the use of secondary notation, there is no provision for abstraction included in the primary notation. It can, therefore, be said that algebra is an *abstraction-hungry* system, whereas arithmetic is an *abstraction-hating* system. Without abstractions, arithmetic is easy to get started with, but can be a very verbose and inefficient notation with low *visibility of notation*, as outlined above.

2.1.2 Proposed System

2.1.2.1 Overview

To improve upon the standard approach of listing the steps comprising a calculation, a system must acknowledge and try to overcome the drawbacks listed above. To this end, I have designed and developed a notational system and accompanying applciation that aims to overcome many of the disadvantages inherent in traditional, handwritten arithmetic. The intention of the system is to reduce *viscosity*, increase *visibility of notation* and to remove many of the *hidden dependencies* that beleaguer the traditional method.

2.1.2.2 Functional Requirements

The new system must satisfy the following functional requirements to be considered acceptable.

- Is able to present the user with a series of problems to solve
- Accepts a file containing the set of questions to be answered
- Displays the current question being answered
- Is able to determine whether or not the user's solution is correct
- Progresses through the current question set as the user answers each question correctly
- Is able to interpret the user's work and display the result interactively [1]

2.1.2.3 Non-Functional Requirements

The system must also satisfy a number of non-functional requirements. These are listed below.

- Offers a significant improvement in *visibility of notation* over handwritten arithmetic

[1] Roca and Rousseau have this to say on the subject of interactivity: "An abundance of studies into user tolerance of round-trip latency [...] has been conducted and generally agrees upon the following levels of tolerance: excellent, 0-300ms; good, 300-600ms; poor, 600-700ms; and quality becomes unacceptable [...] in excess of 700ms." [21]

- Offers a significant improvement in *juxtaposability of notation* over handwritten arithmetic
- Reduces *premature commitment* relative to handwritten arithmetic
- Reduces *hidden dependencies* relative to handwritten arithmetic
- Has a visually simple interface whilst remaining accessible to older users

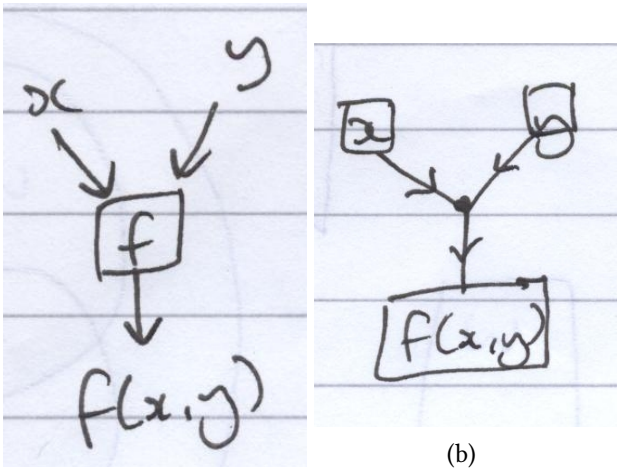
2.2 User Interface

As this project is primarily concerned with human-computer interaction, the user interface and experience, and the design thereof, constitutes a significant part of this project. As such, this section outlines the initial development stages of the user interface, first introducing several designs for the graphical notation, and then discussing in more detail three that were developed further.

2.2.1 Prototyping

2.2.1.1 Low-Fidelity Prototyping

To try to get some initial ideas for what could become the calculation metaphor of choice for the project, a target was set of devising at least twenty, significantly different, potential designs. These were recorded as very rough sketches, a few of which were refined in larger examples, and three of which were deemed good enough to warrant an application mockup, as detailed in *Sec. 2.2.1.2*.

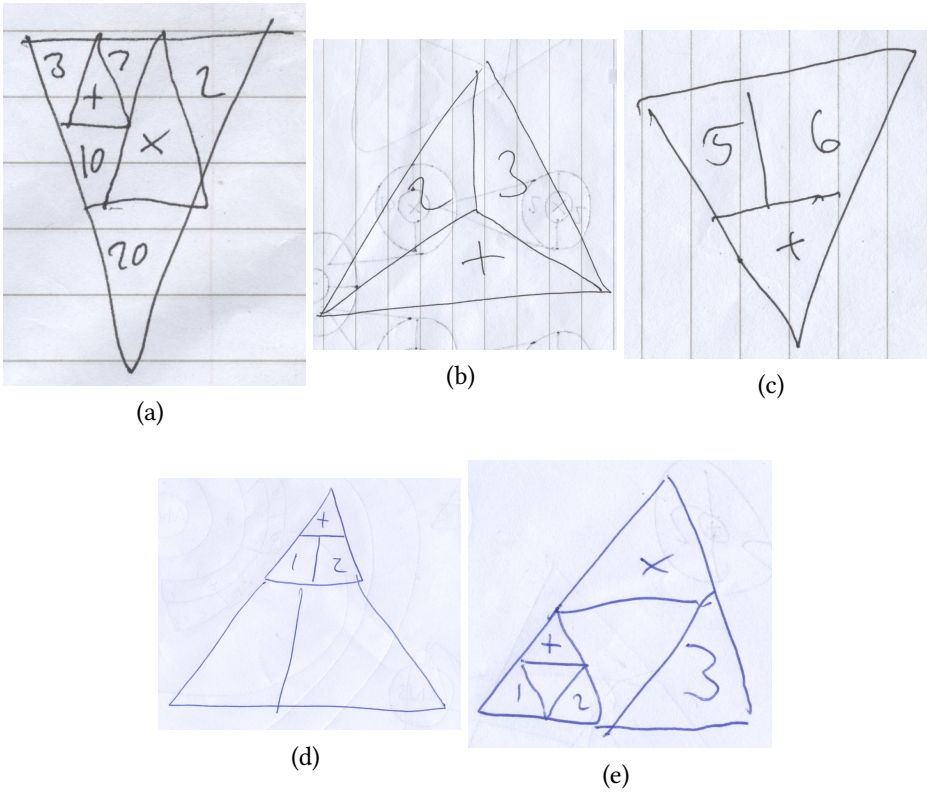


(a) (b)

Figure 2.2: Early designs for flowgraph-based calculation metaphors.

The designs in *Fig. 2.2* use a flowgraph-based metaphor, in which variables and functions are represented as nodes in the graph. Dataflow is extremely clear in such diagrams, giving these designs high *visibility of notation.* These designs are similar to the original one illustrated in the project proposal (*Appendix C*). The boxes and arrows used in such notation also occupy relatively little screen estate, which is advantageous in cases where a question necessitates a lot of subcalculations. *Fig. 2.3* shows a more refined version of a flowgraph notation, with a potential environment for it to exist in.

Figure 2.3: A more refined sketch of a flowgraph-based language and application.



(a) (b) (c)

(d) (e)

Figure 2.4: Early designs for triangle-based calculation metaphors.

Triangle-based notations were also considered. Drawing inspiration from the formula triangles used in science education (*Fig. 2.4, c)* and *d)*) and from the Sierpiński triangle fractal (*Fig. 2.3, a)* and *e)*), such notations not only appear in a form familiar to many users, but also have high *visibility of notation*. *Juxtaposability of notation*, however, is low, as this design enforces a rigid structure upon calculations. Nevertheless, a more refined sketch is shown in *Fig. 2.5*.

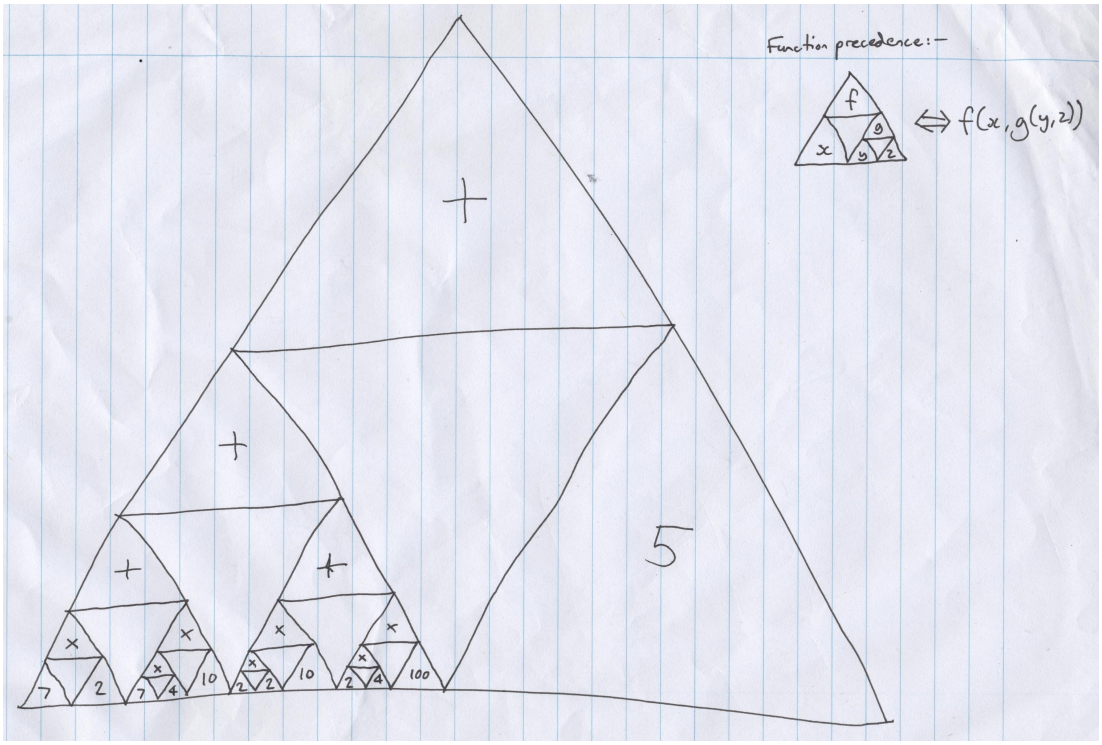


Figure 2.5: A more refined sketch of a triangle-based language.

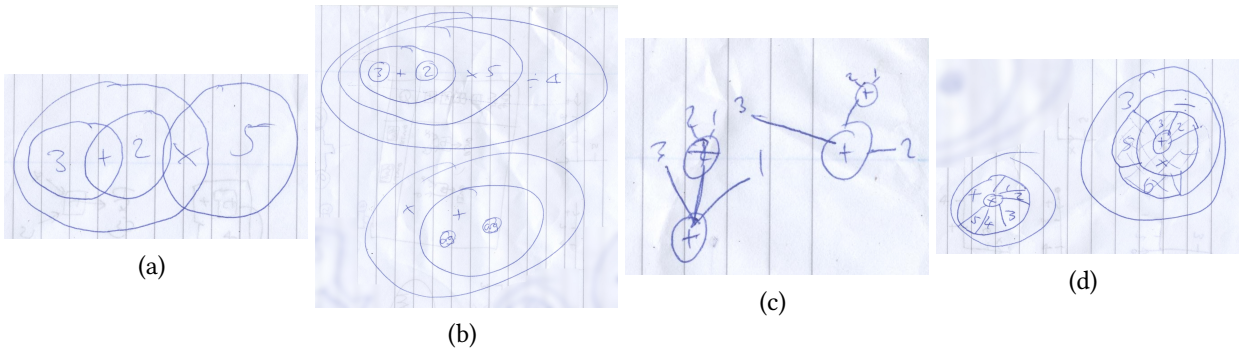


Figure 2.6: Early designs for circle-based calculation metaphors.

Several circle-based designs were also considered, the focus here being on subcalculations, rather than numbers and operations separately, as individual units within a larger solution to a problem. Such a notation provides high *juxtaposability of notation*, as whole subexpressions can easily be relocated, and also offers an improvement in *visibility of notation* over handwritten arithmetic. A more detailed sketch is shown below.

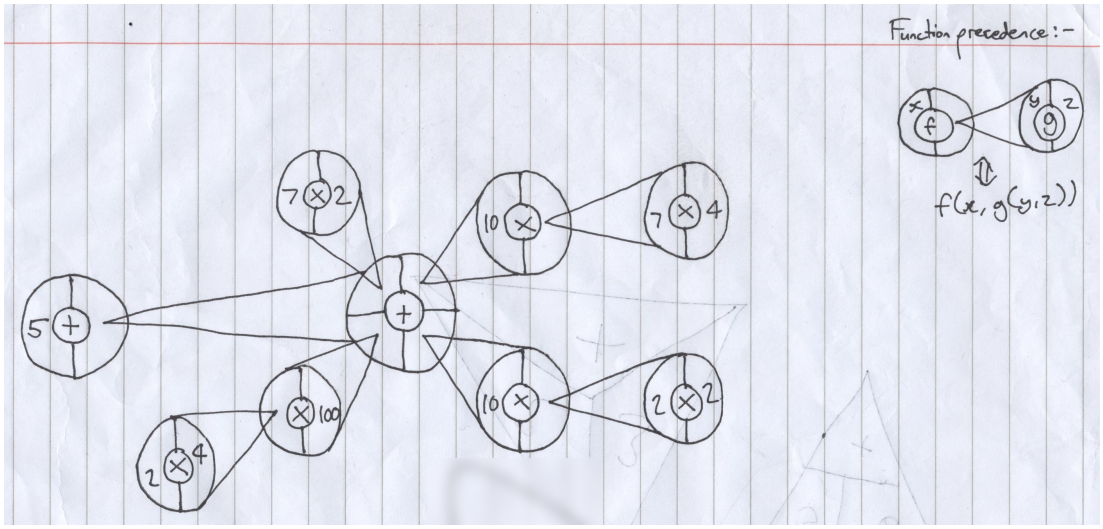
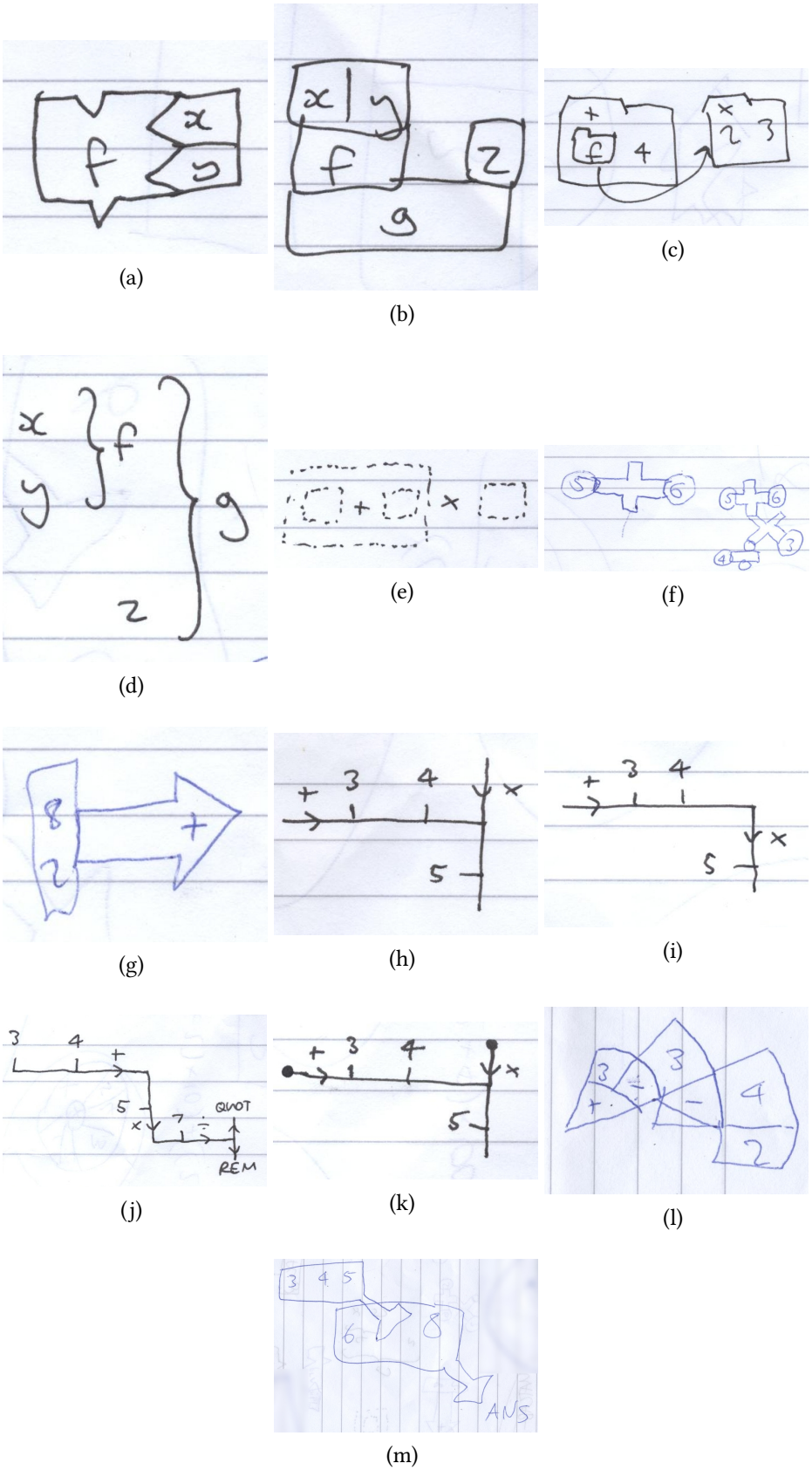


Figure 2.7: A more refined sketch of a circle-based language.



(a) (b) (c) (d) (e) (f) (g) (h) (i) (j) (k) (l) (m)

Figure 2.8: Assorted early designs for other calculation metaphors.

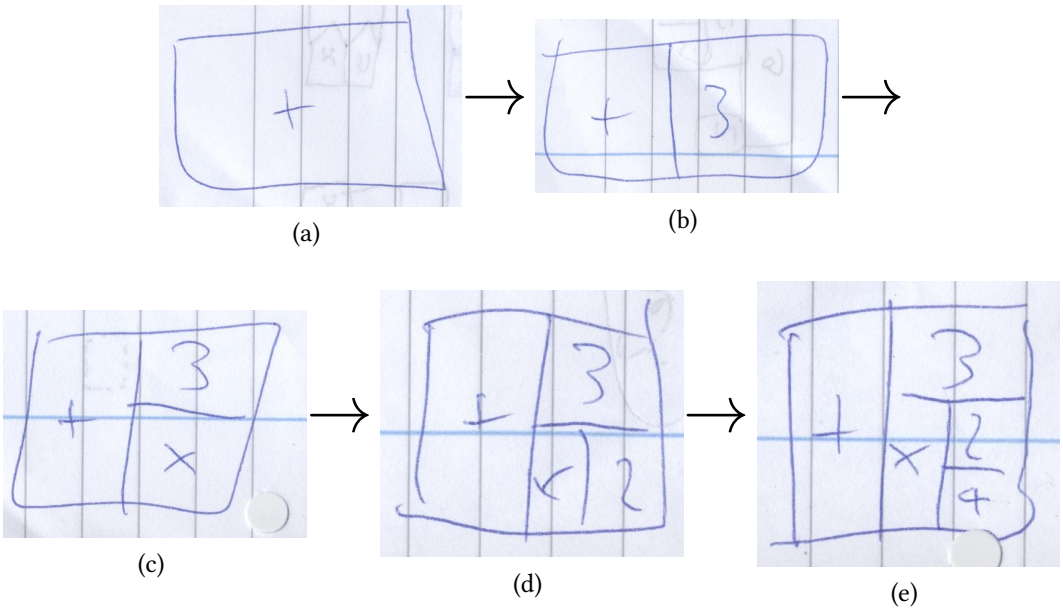


Figure 2.9: A sample progression of a calculation in a language based around infinitely halving a rectangle.

Various other metaphors for calculation were also considered, as illustrated in *Figs. 2.8 and 2.9. a)* was considered too similar to Google/MIT's *App Inventor* software. *c)*, *f)*, *l)* and *m)* were deemed too cumbersome, and *d)*, *e)* and *h)* to *k)* were considered not visually appealing enough, looking too similar to existing mathematical notations and constructs. *b)* and *g)* were inflexible, and would have scaled poorly to larger calculations. Finally *Fig. 2.9* was rejected as it was too similar to the Sierpiński triangle metaphor shown in *Fig. 2.5*.

2.2.1.2 Detailed Mockups

Some preliminary designs for the application and graphical language follow, being more mature versions of three of the low-fidelity prototypes presented above. First is a combined flowgraph representation and Read-Evaluate-Print-Loop (REPL) design. The second design uses the Sierpiński triangle as a basis for the visual metaphor. The final design is a circle-based language, similar to the one shown in *Fig. 2.7*.

2.2.1.2.1 Flowgraph Metaphor

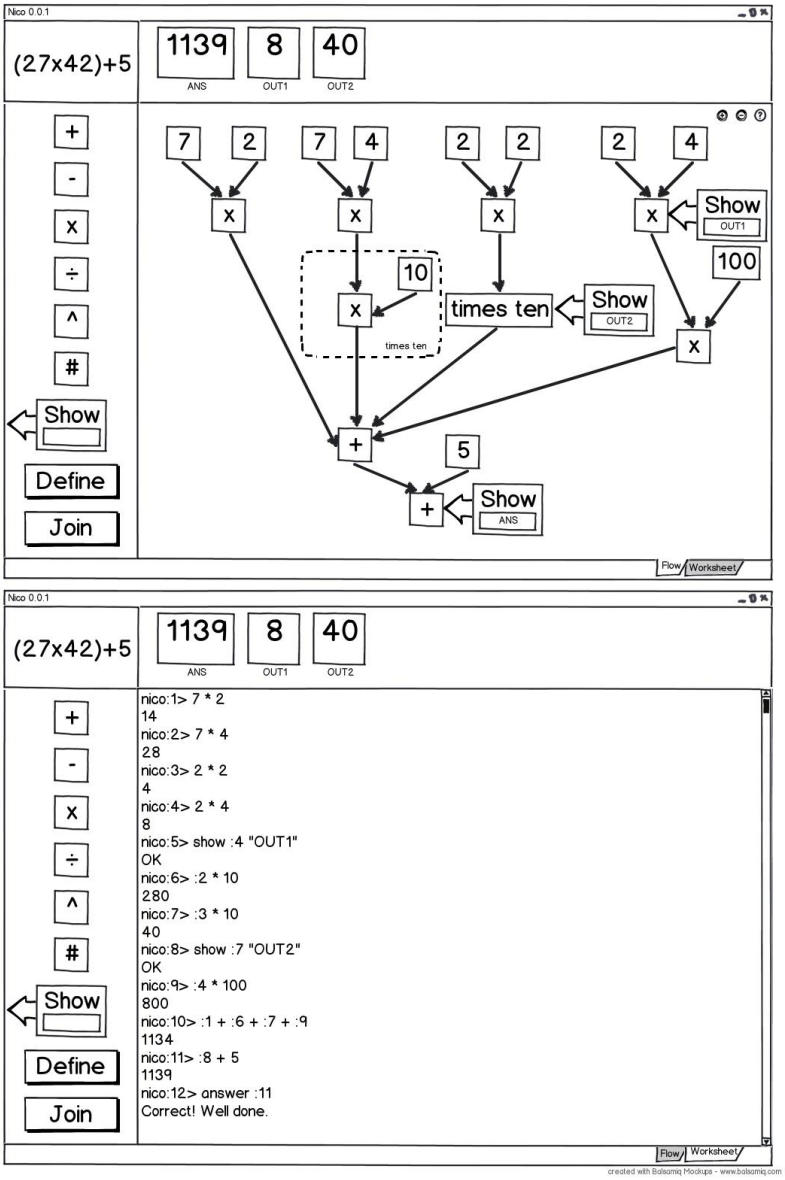


Figure 2.10: The prototype flowgraph-style language.

The flowgraph idea initially outlined in the project proposal and included in the low-fidelity prototypes above appears here as a more mature, revised prototype, showing two screens of a sample application. The top screen shows the proposed application manipulating the graphical language, whilst the bottom screen shows a REPL utilising a simple text-based language.

The application is comprised of one window containing information panels at the top, a ‘toolbox’ of available functions that can be dragged on to the canvas at the

side and a canvas area upon which calculations are constructed. There are buttons to zoom in and out, and a help button. The user is able to use the tabs at the bottom of the canvas to switch between the graphical mode (Flow) and the REPL mode (Worksheet).

This design is moderately faithful to the origin application sketched out in the project proposal. The application layout has been expanded to include the toolbox, and the Worksheet mode has been added. The ability to define functions has also been included.

The flowgraph metaphor was used as it offers much *visibility of notation*, making the flow of data very clear to the user and allowing them to see much of their structure at once. The notation also occupies little screen estate, allowing it to accommodate large calculations.

The toolbox was included to make it clear to the user what functions are available to them. In the original design, no control scheme was specified; the user is expected to have control over many different functions, so the control scheme either needs to be simple and memorable, or the user, espcially the novice, needs guidance.

In the toolbox are two buttons Define and Join that change modes such that dragging the mouse either joins boxes with arrows, or draws a box around a subsection of the diagram to define a function. It was decided to use these modes, combined with the dragging of boxes in the normal mode, so that the entire control scheme could be implemented using solely one mouse button.

The zoom buttons were included to help accommodate particularly large diagrams, and allow the user to view their work at a size that is comfortable for them. If the user requires more canvas space, wishes to focus on a particular part of the diagram or would prefer to have the notation made larger, they are able to do so using these buttons.

The help button was included to provide users with a simple system to aid in their getting used to the system. Activating help would bring up an information box, allowing users to look through a manual.

Function definition was introduced to allow the user to create abstractions for repeated tasks, thus increasing the compactness of the language.

The Worksheet mode was included for users who may feel more comfortable working in a written environment, rather than using the graphical metaphor.

This design was abandoned as it was deemed to be too complex for the target audience (Year 5). The intention of this project is not to implement yet another

programming language for learners, and by providing features like the REPL and function definition, the system unnecessarily overprovides. The REPL itself is unnecessary, as users comfortable working in a textual environment are likely to be comfortable using the traditional, handwritten method. The REPL simply adds complexity to the existing system.

In addition to this, the flowgraph representation had a number of shortcomings. For example, it is not immediately clear in what order the arguments feed into an operation, which is confusing for non-commutative functions such as subtraction. Furthermore, the language is quite verbose. Although this is not inherently a disadvantage, a more compact representation would be preferable, in order to make expressions written in the notations more readable to people other than the original author and to decrease the time spent by the user searching for information within the notation. It is also fairly *viscous* at the scale of replacing whole components, rather than just numbers or operators, though less so than handwritten arithmetic.

2.2.1.2.2 Sierpiński Triangle Metaphor

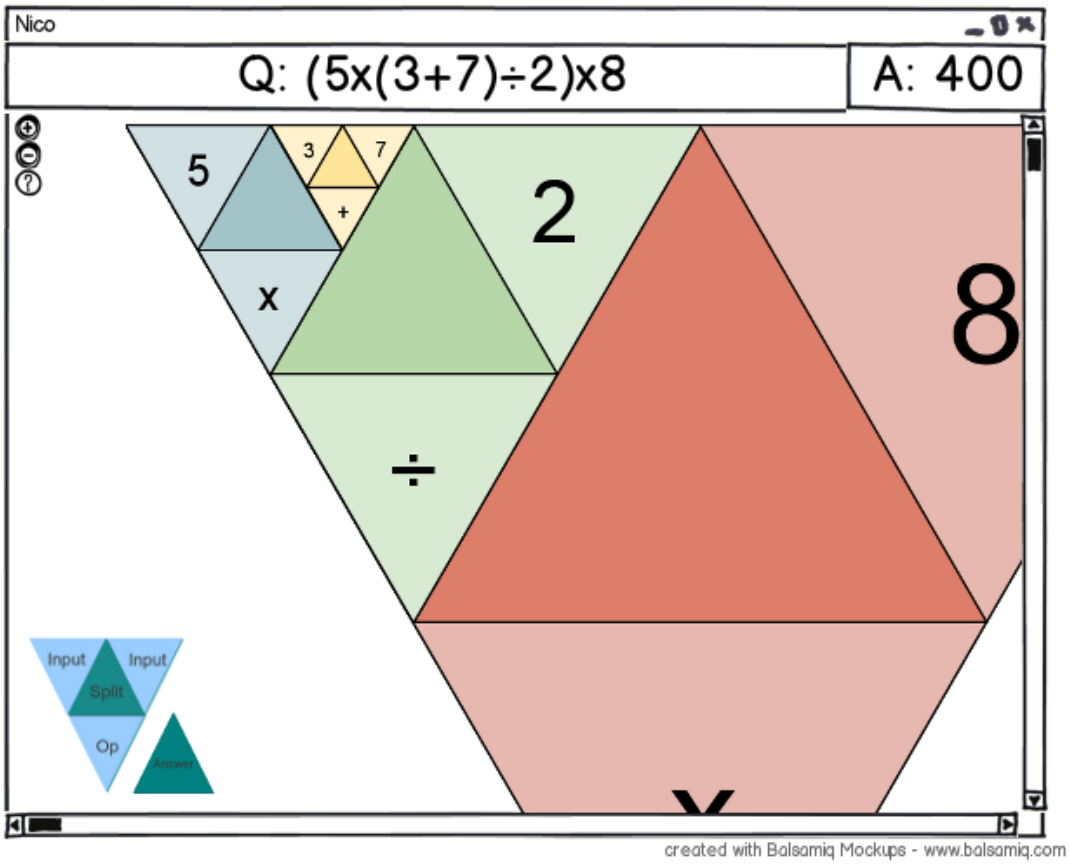


Figure 2.11: The prototype Sierpiński-triangle-based language.

This prototype uses a zooming instance of the Sierpiński triangle fractal as the basis of its notation. It is shown in *Fig. 2.11* manipulating a simple calculation.

The application consists of a single window with information panels and a canvas area, featuring zoom and help buttons, as well as control buttons for the construction of an answer.

This prototype marks a significant departure from the flowgraph metaphor outlined above and in the project proposal; to address the problems with the flowgraph notation that were discussed in the previous section, new notations were considered. The application in which the notation is used also had to change to suit the new notation: in this case, a toolbox would not be suitable, as the user is limited to one structure: the triangle.

The triangle notation was chosen as it offers more *visibility of notation* than the handwritten method, and places considerable emphasis on structure, making it clear how smaller calculations can be built up to form larger ones by using a fractal-based

notation. It is colour-coding by the amount which each triangle is nested, to clarify at which level the user is working and allowing the user to abstract away smaller triangles as simply units of the calculation to be manipulated, as one would a number.

Zoom buttons were included as, although the size of the overall structure always remains the same in this notation, it becomes necessary to read very small sections quite quickly, as nested calculations are built up. Indeed, each section of the triangle has a quarter of the area of its the triangle that contains it, meaning that the triangle containing the value 3 in *Fig. 2.11* has $\frac{1}{256}$ of the area of the overall structure, and this is with only four levels of nesting.

As in the previous prototype, a help button was included to ease initiation into using the software.

A control scheme using buttons was chosen, again, so that the user is only required to use one mouse button. The control buttons map to the structure of a triangle unit. This design was chosen to make it clear to the user how each control will affect the structure being worked upon: each `Input` button allows the user to modify the corresponding section of the triangle, and similarly for the `Op` button. The user is able to submit answers using the `Answer` button, to allow them to check and change their answers before submission.

This design was ultimately abandoned as it was deemed to be awkward for a number of reasons, not least because it was based solely around binary operations, meaning that to solve the simple question 1+2+3+4, one has to either calculate ((1+2)+3)+4, which is inefficient, or mentally calculate that 1+2+3=6, and then calculate 6+4, which is obviously unacceptable for a system that intends to help the user express themselves mathematically.

The severe lack of juxtaposability is also a problem: as each expression must fit into the larger structure, it is not possible to reorder them at will, without fundamentally altering the calculation being performed. It also enforces a top-down approach to calculation: one is required by the system to begin with the outermost calculation and work inwards. There is no provision for putting a larger triangle around an existing calculation.

In addition to this, an informal survey of potential notations was conducted, asking approximately ten participants to answer some sample questions in this notation and the circle-based notation (below). Participants found this notation less clear, and were much better able to complete questions correctly using the circle notation described below.

2.2.1.2.3 Circle Metaphor

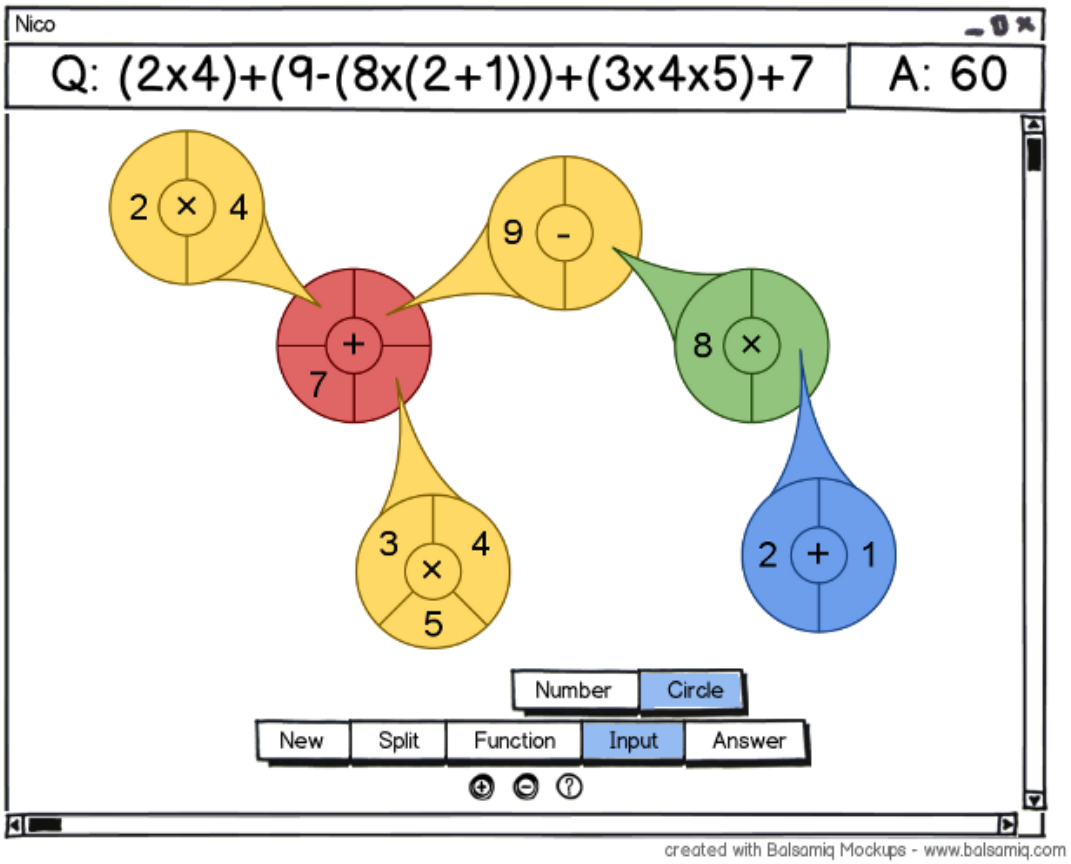


Figure 2.12: The prototype circle-based language.

This prototype uses a notation based around circles as individual units of calculation, linked in such a way as to indicate the flow of information. Each circle consists of an operator, around which arguments orbit, being evaluated clockwise from 0°. Although it is related to the flowgraph metaphor (above), it has a number of key differences.

The application comprises a single window with information panels and a canvas, which includes control, zoom and help buttons, similarly to the above prototype.

Once again, this prototype is markedly different to the original idea for the project, as a means of addressing its shortcomings. The application has also changed to accommodate the new notation. A toolbox is unsuitable for this notation, as it only uses one fundamental structure, being a circle, and the controls used in the Sierpiński model above (*Fig. 2.11*) do not map to the circle structure.

The circle metaphor was chosen for similar reasons to the triangle model: it emphasises the idea of subcalculations as individual units, to be abstracted away and treated as any other entity being provided as an argument to an operator. This model has greater *visibility of notation* than handwritten arithmetic, and also offers more flexibility than the previous two models, decreasing *viscosity* by allowing the user to edit whole units (i.e. circles) within a calculation, and to swap them in and out as needed. *Juxtaposability of notation* is increased using this metaphor, as whole subcalculations are easily repositioned.

Unlike with the Sierpiński metaphor, the user is not restricted to one order or pattern of constructing a calculation: calculations can be performed bottom-up, top-down or a mixture of both, as is comfortable for the individual user. The circle 'tails' make the flow of information through the diagram clear, and circles are also colour-coded by their level of nesting, to further clarify dataflow and to eliminate as many *hidden dependencies* as possible.

The system also requires little *premature commitment*, as any piece of the diagram can be created, deleted and modified at will, so it is ideal for exploration, allowing the user to try many ways of solving a problem within the application, rather than having to use a secondary notation or device to help work through a problem in a more familiar manner and then transcribing that solution into the new notation.

As in the flowgraph-based prototype, zoom buttons are included to accomodate large diagrams, to allow the user to focus on small sections and to allow the user to work at a size that is comfortable for them. Again, a help button is included to ease initiation.

Similarly to the Define and Join buttons in *Fig. 2.10*, the buttons here specify mouse modes: New causes clicking to create new circles, Split causes circles that are clicked to gain an argument, and so on. This had similar motivatoins to the modes in the flowgraph model: to be able to control the software using just one mouse button. The user is able to submit answers using the Answer button, to allow them to check and change their answers before submission.

This design was the one that was ultimately chosen as the basis for the project as it neatly solved, or at least ameliorated, many of the problems listed with handwritten arithmetic, without being too complex for the target audience (as with the flowgraph system), and without being too restrictive with regard to mathematical expression. In addition to this, the results of the survey mentioned above also indicated that people got on well with this notation; the participants found it considerably easier to use than the Sierpiński notation, and made fewer mistakes in

handwriting this notation than the alternative.

2.3 Third-Party Tools

What follows is a list of the third-party tools that were used in the development of the project.

- Ubuntu Linux 10.04, Arch Linux 2010.05, Microsoft Windows 7
- Clojure 1.2.0
- Leiningen 1.6.1.1
- OpenJDK 6
- Seesaw 1.3.1-SNAPSHOT
- swank-clojure 1.3.4-SNAPSHOT
- GNU Emacs 23.1.1
- A modified version of Overtone's Emacs configuration [22], including:–
  - SLIME/SWANK (revision as of 15/10/2009)
  - clojure-mode 1.11.5
  - undo-tree 0.3.3
- Git 1.7.0.4
- GitHub
- Balsamiq Mockups
- Google Docs
- R 2.15.0

2.4 Summary

In this chapter, the work done in preparation for beginning the main implementation of the project was detailed. A thorough requirements analysis has been conducted to outline the expected behaviour of the product, contrasting it with the existing system of handwritten calculations. In *Sec. 2.2*, prototype designs for a

suitable graphical notation were introduced. Three in particular were explored in detail, before deciding on the final notation upon which to base the project. The next chapter contains a detailed analysis of the work completed to implement the project.

Chapter 3

Implementation

This chapter details the implementation of the project. The code itself totals approximately 1,500 lines of Clojure, and the resultant application, *Nico*, satisfies the requirements laid out in the project proposal, namely:–

> "[to be] able to generate an abstract syntax tree in Clojure from the graphical language and evaluate such a tree, passing the results back to the graphical application and displaying this to user in less than 300ms."

In addition to this, an extension to the project was completed, in the form of a study that tested the software on real people.

3.1 System Architecture

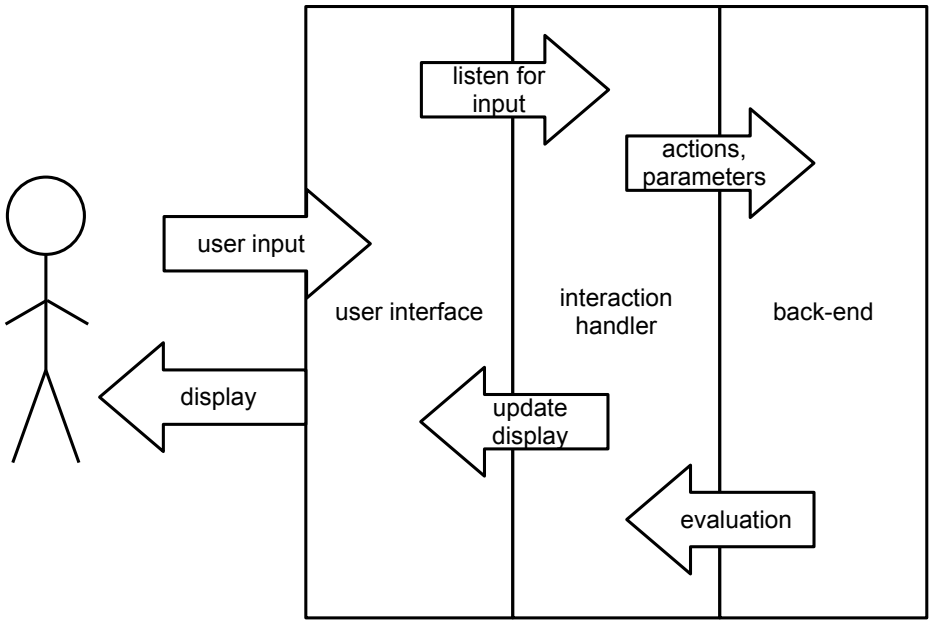


Figure 3.1: A model of *Nico*'s three-layer architecture.

The system comprises three distinct layers of operation: the user interface, the interaction handler and the back-end. The user interface is what the user directly interacts with, with events in this layer being processed by the interaction handler, with the resultant data being passed to the back-end, which handles the mathematical and memory-management operations, to interpret. Data can also flow in the other direction, with output from the back-end being passed up to the interaction handler to cause effects in the user interface. An analysis of each layer in greater detail follows.

3.1.1 Back-End

The application back-end handles the mathematical aspect of the application; that is, it is able to interpret data gathered from the user's input and process it as a calculation.

Circles could not simply be represented in the application as S-expression versions of the calculations they stood for, as they were not defined solely by their expressions. Data were needed regarding their position on the canvas, so it was

decided to use an associative map to represent a circle, containing co-ordinate data as well as the calculation itself.

Originally, the application was to use two macros to be able to define circles as new local and global variables at runtime, but it became apparent that such variables were not accessible by the interaction handler whilst the program was running. It was, therefore, decided to use a mutable data structure to store a list of all circles currently in use, and to add 'name' data to the circle maps, storing each circle's label as a string. This solved the problem, providing a useful means of storing data whilst making it available to be accessed and changed at will during the program's execution.

3.1.1.1 Management of Mutable State

Nico's back-end stores and manages circles and questions using just six mutable data structures. They are detailed below:

- The list of circles, as detailed above
- The current question set, a list of maps containing the question number and the S-expression that constitutes the question to be answered
- An integer counting how many circles have previously been created whilst answering this question
- A map containing the current co-ordinates of the mouse cursor
- An integer indicating the number of the question currently being answered
- A temporary store for information about circles that are currently being repositioned by the user

Mutable data structures were chosen for these structures as they needed to be continually updated whilst the program was running.

3.1.1.2 Circle Management

Two main functions for the management of circles have been implemented: creation and deletion. New circles are initialised such that they are centred on the current co-ordinates of the mouse cursor, with a name of the format "c*n*", where *n* is the number of circles currently in use. Upon creating a circle, its expression is

initialised to `(\? \? \?)`, a meaningless expression using the question mark character `\?` as a placeholder value for both the operator and the two arguments.

It was decided that circles should be created with a placeholder expression so that the user is able to create a circle without having to specify parameters for it beforehand, which would entail some *premature commitment*, as well as the *hard mental operation* of having to visualise a circle before committing to creating it. Such an approach was exhibited with a previous revision of the software, requiring parameters to be specified in a large dialogue box (*Fig. 3.10*) before a circle would appear.

It was also decided that the circle should use explicitly placeholder values, rather than an expression of value 0 (such as 0+0) so that new circles did not affect the current total displayed by becoming the new root circle (see *Sec. 3.1.1.3* for a full discussion of how diagrams are evaluated). The placeholders also indicate to the user, when displayed, that action is needed, whereas an expression of value 0 could appear to be a deliberate inclusion in the user's calculation.

Deletion is a more complex function that removes a circle from the list, by searching through the main circle list and returning a version which lacks the circle matching either the name or the location supplied, which the circle list is updated to.

This naïve implementation of deletion led to many problems: the function described above does not deal well with circles that are used as arguments to other circles. When such a circle is removed from the main list, there still remain references to it in the circles that were using it as part of a larger calculation, causing the application to throw `java.lang.NullPointerExceptions`. As such, a 'safe' version of circle deletion was implemented.

The 'safe' deletion function works by first checking if any of the circles currently in the main list contain references to the circle being deleted. If it is found that there are circles that contain references to it, the function determines if that circle is a root circle (*Sec. 3.1.1.3*), and, if so, uses the previous method of deletion to remove the circle and replace it with a similar circle that uses placeholder values where there were references that would have been invalidated. If the circle is non-root, the deletion function recursively calls itself upon the circle, thus removing all circles that formed the original chain dependent upon the circle to be deleted. The circle can then safely be deleted using the previous method, and the intermediate circles are replaced.

This solved the problem of the leftover references to non-existant circles, but it has also given rise to a new problem: using this method, and in particular due to the

stage in which the safe deletion function must call itself upon the other circles in the chain, deleting a circle that forms part of a chain deletes all of the links between the rest of the circles in the chain, from the circle being deleted down to the root circle. On reflection, a better way to implement safe deletion would have been to include either as part of the rendering loop or as part of the safe deletion function a function to inspect the main circle list, checking for any invalid references after deletion and replacing them with `\?`s, rather than doing so at delete-time.

3.1.1.3 Evaluation

There are two main stages required to evaluate the user's work: finding the root circle, into which all other circles feed, and evaluating a circle or collection of circles to give a valid, computable S-expression.

The root circle is the one circle in a finished diagram that is not used as an argument to any of the other circles. Therefore, it can be found by searching through the list of circles, comparing each one to the rest of the list and discarding it if there are references to it in other circles. As stated, in a **finished** diagram, there can only be one root circle, so this method is sufficient for use in marking answers.

Whilst the user is working, it is possible that they may construct two or more calculation structures in parallel, with intent to join them later. In this case, there can be multiple root circles. The method chosen to identify the root returns the first instance of a root circle that it comes across. It would have been possible to, for example, add an extra answer display for each root and maintain a total for each, but it was decided that this would be confusing for the user, who may not expect the structure of the application window to change. This would also require a labelling system for circles and structures, which was abandoned in the user interface design as it is an unfamiliar concept to non-programmers and those without algebra (the majority of the target audience), so it was decided to simply display the result for one structure.

Once the root has been found, the user's diagram can then be evaluated to give an S-expression. As each circle contains an S-expression for the calculation unit that it represents, the references to other circles can be resolved, creating a nested S-expression that can simply have Clojure's `eval` called on it to return an answer.

This was, in fact, a significant reason for choosing Clojure, as a dialect of LISP, as the development language for *Nico*; as a homoiconic language, code to be evaluated later can easily be built up as a data structure, which can then be passed around,

evaluated and otherwise utilised as is convenient. By creating one unified S-expression representing the entire diagram, not only can the value of the structure be calculated, but other operations, such as the selective highlighting of the question display (*Sec. 3.1.2.2*), can be performed.

3.1.1.4 Question Management

External files containing sets of questions are used to provide tasks for the user to complete. An NQS (*Nico* Question Set) file is simply a plain text file contaning S-expressions corresponding to questions, separated by newlines. This particular approach was chosen so that questions can easily be read into a list using functionality provided by Clojure's standard library, meaning that it is possible to navigate between questions by setting the current question to the value of the desired question in the list.

A future extension to this project could be to develop a partner application to *Nico* to aid tutors unfamiliar with LISP to develop their own question sets in a more 'user-friendly' manner, rather than by transcribing lists of problems into S-expressions.

3.1.2 Interaction Handler

The interaction handler is a collection of functions that form the 'middle layer' of the application (*Fig. 3.1*): that is, the functions that arbitrate between the back-end and the user interface.

The rendering function is called every time the mouse is moved, or when what is displayed on the canvas needs to be updated. It paints a white rectangle over the canvas area, draws a bin icon in the corner (for circle, deletion, see information on the control scheme below) and iterates across the circle list, drawing a circle to match each element. References to other circles are visualised as lines extending from the source circle to the appropriate argument slot on the recipient circle. By re-rendering every time the mouse is moved, rather than continously, the application does not have to render when the canvas is not being changed, as every action in the control scheme requires some use of the mouse.

3.1.2.1 GUI Libraries

The original intention of the project was to develop *Nico* using the JavaFX library. However, a number of difficulties were encountered with this: firstly, JavaFX version 2.0 was not available for Linux, the primary development environment, at the time of beginning the project. Installation was attempted on a Windows machine, but caused a number of problems. In the interest of making progress, it was decided to proceed using the Eclipse Foundation's SWT toolkit and Szymon Witamborski's corresponding Clojure bindings, *GUI FTW!* [27].

During the development of the user interface, many problems were encountered with SWT and *GUI FTW!*, particularly regarding drawing arbitrary shapes on the canvas area and extracting user input from dialogue boxes. To keep the project on-schedule, the application was migrated to Java Swing, using David Ray's *Seesaw* library to provide bindings for Clojure. [20] This allowed for more efficient development, given the author's previous experience using the Swing library.

3.1.2.2 Main Window

At the heart of the interaction handler is the structure defining the main window, which comprises the majority of the user interface. It controls what is to be displayed and the actions to be performed under certain conditions. It listens for certain events (i.e. user input), and prompts the user for input when required. The main window is defined as one large structure as this is how Seesaw structures interfaces: each window is defined separately, with appearance, listeners and other parameters set as options within the structure. Other structures, such as panels and canvases, can then be used as content.

Embedded into the main window structure are several listeners that monitor the canvas for mouse-based events. While the application is running, it listens for five mouse-based events: motion, dragging, a button press, a button release and a button click.[1]

When motion is detected, the canvas is cleared and re-rendered, as mentioned above. Motion also triggers a check to determine whether or not the mouse is over a circle; if it is, a highlighting function is called. This draws a ring around the circle currently being moused over, and also performs a string comparison: using the

[1]It should be noted that a mouse click is defined by Java as a button press followed by a button release, and that, upon a click, it issues a press event, followed by a release event, followed by a click event. [8]

same function written to convert the questions into traditional mathematics for display in the information panel, the circle's S-expression is converted and compared to the question string to see if it appears as a substring. If so, the appropriate section of the question string is also highlighted, to help reinforce the notion of the circle notation as a metaphor for calculation. It was decided to implement this a string search, rather than by using the actual S-expressions for the question and for the circle's calculation as although this would be a more efficient method to determine whether or not a circle corresponds to a part of the question, it is less efficient to do so and then calculate where the matched S-expression appears, rather than just matching a potential substring against a longer string.

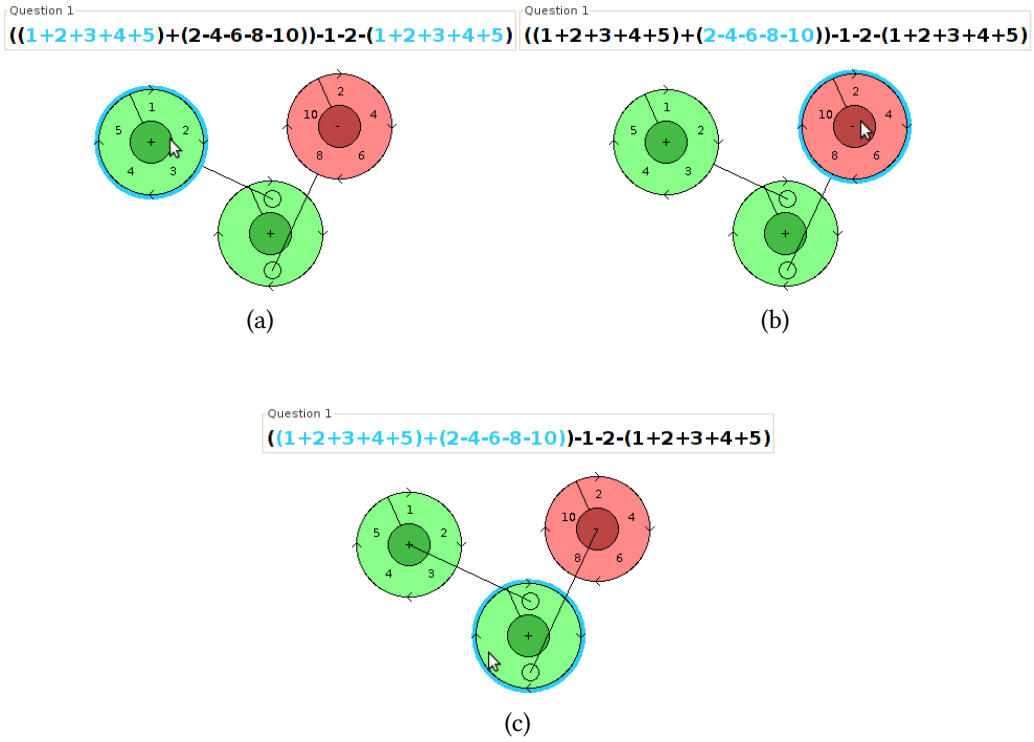


Figure 3.2: Three instances of the question text being highlighted as a corresponding circle is moused-over.

3.1.2.3 Control Scheme

Upon the detection of a mouse click, the software must determine the exact context in whcih the mouse was clicked, as this constitutes much of the application's control scheme. The mouse's current co-ordinates on the canvas, as well as

information regarding which mouse button was pressed and any modifiers that were active, are used to determine the correct action to take, as illustrated in *Fig. 3.3*.

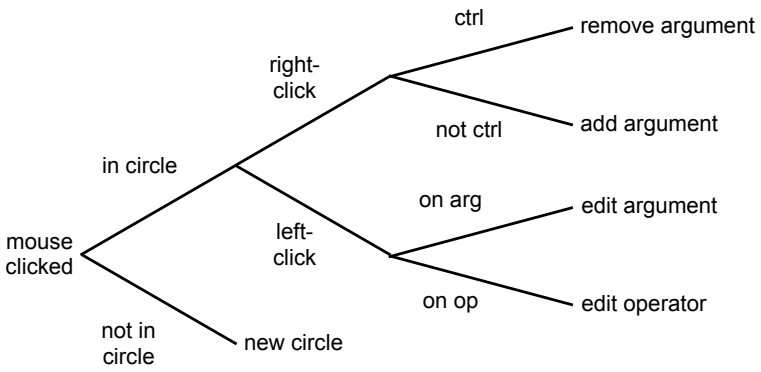


Figure 3.3: Decision tree to determine the appropriate response to a mouse click event.

A more in-depth analysis of the control scheme follows.

- Mouse not in circle
  - A blank circle is created, centred on the current location of the mouse cursor. It has two arguments, ? and ?, and its operator is ?.
- Mouse in circle
  - Left-click
    - Mouse on argument
      - A small blue circle is drawn around the argument and a dialogue box is launched, containing a spinner with which to set the new value of the selected argument.
    - Mouse on operator
      - A small blue circle is drawn around the argument and a dialogue box is launched, containing four radio buttons corresponding to the four arithmetic operations, with which to set the new value of the operator.
  - Right-click
    - Ctrl key pressed

        - The final argument is removed from the circle, down to a minimum of two arguments. An alert box warns the user if they try to remove arguments when there are only two remaining.

    * Ctrl key not pressed

        - A placeholder argument is added to the circle, up to a maximum of eight arguments. An alert box warns the user if they try to add more than the maximum number of arguments.

Drag-and-drop functionality is also included in the control scheme. When the application detects that a mouse press has occurred, if the event was triggered whilst the mouse cursor was over a circle, that circle's map is stored, along with the current system time in milliseconds and the mouse button that was pressed, in a mutable data structure. When a drag event is detected, that information is then used to perform one of two operations upon the circle.

If the left mouse button is held whilst dragging, the circle's positioning information is updated to the current co-ordinates of the mouse cursor. If the right mouse button is held whilst dragging, a line is drawn from the circle to the current location of the mouse cursor, indicating that two circles are to be joined. When the mouse passes over the argument slot of another circle, that line is fixed, and the target circle is updated such that the argument now has the value of a reference to the source circle.

The drag-and-drop functionality was implemented thus as a drag-and-drop action constitutes a press event, followed by a drag event, followed by a relase event. By storing information about the circle being dragged upon detecting a mouse press, the function called when dragging is detected can use and modify this information easily, without encountering the problems of defining new variables at runtime, as mentioned in the discussion of the application back-end. This does not impact upon other mouse events involving presses and releases (e.g. clicking), as this information is discarded upon every mouse release, regardless of whether or not it was being used.

The design of the control scheme is discussed further in *Sec. 3.2*.

The control scheme is almost entirely mouse-based as the mouse is a far less complex device than the keyboard, and is familiar to most computer users in a school environment, regardless of ability (indeed, the use of applications that require mouse control, such as many word processors, is mandated by the National Curriculum). By limiting control to the mouse, the user is required to remember

fewer button locations, and is likely to be able to guess controls correctly if forgotten. A caveat to this is that the use of the Ctrl key in combination is required, as it was decided that this was preferable to mouse chording (unfamiliar to novice users), or to requiring a three-button mouse (diverse computing environments in schools mean that three-button mice may not always be available).

3.2 User Interface

The user interface comprises the parts of the application which are user-facing; that is, they are that which the user directly interacts with. The development of *Nico*'s user interface was key to the success of the overall project, being primarily an experiment in human-computer interaction.

During its development, *Nico*'s user interface went through a number of revisions. Details of how the UI design developed over time follow.

3.2.1 Application

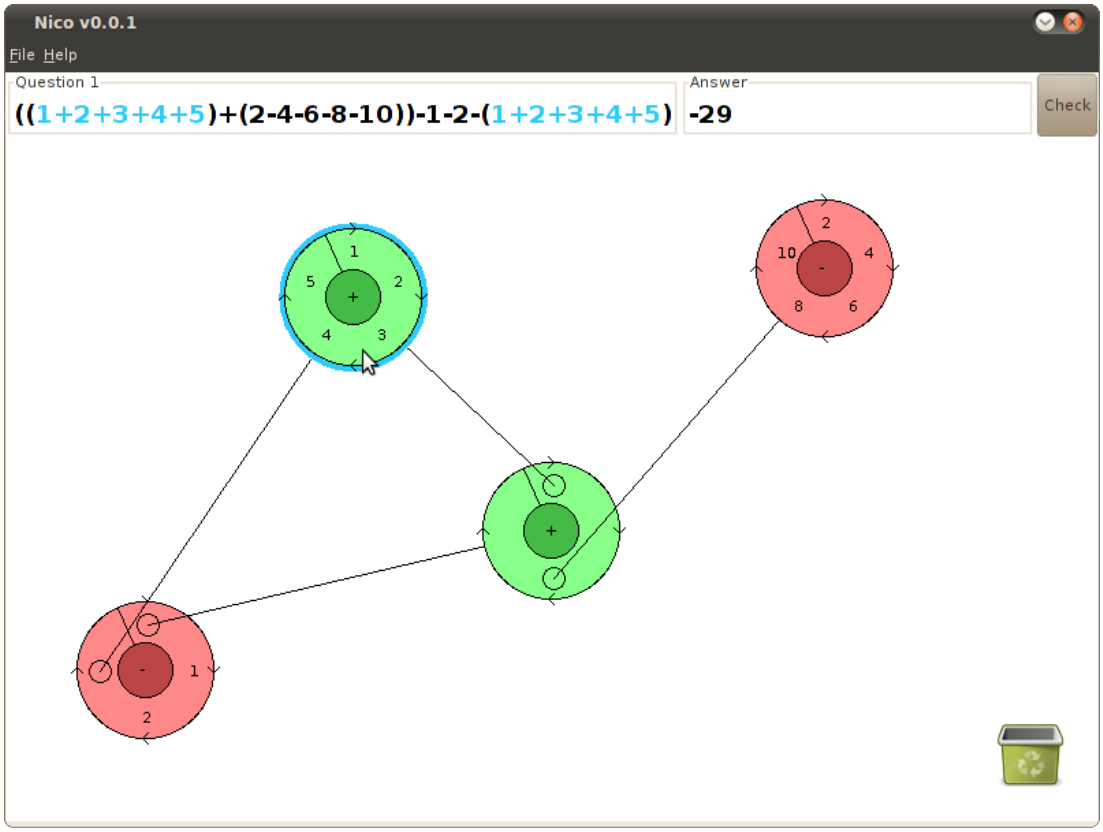


Figure 3.4: A screenshot of a *Nico* session.

The application itself was kept fairly minimal: the aim was to emphasise the notation as the most important aspect of the software. As such, the main application window, as shown in *Fig. 3.4*, is mostly dedicated to the canvas area in which the user answers the question.

The main application window is split up into two main sections, the information panels that display the question being answered and the current total of the user's calculation, as well as the `Check` button to submit the current total as the answer, and the large canvas area in which the user is able to construct diagrams with which to answer the question.

Initially, the application was intended to use more buttons (see the prototype in *Fig. 2.12*, above), having more controls immediately visible to the user.

In the first usable version of the interface, there were a number of buttons that appeared down the side of the application. These had been relocated to save screen

estate, keeping the canvas free for the sole use of the notation. Each button (New, Edit, Delete, etc.) would cause a dialogue box to appear to the user, which required a circle to be specified by name (at this point in the development of the project, it was required that the user name each of their circles) before any operations could be performed upon it.

This design was abandoned as the separation between the user and the diagram itself was too great; creating circles by specifying options in many only slightly different dialogue boxes and watching the results appear on the canvas did not feel like a natural way to interact with one's calculation, and hence would have been inappropriate for the target audience. Indeed, forcing the user to commit to a design before even seeing it entailed much *premature commitment*, and having to re-express circles to edit them greatly increased *viscosity*.

Later revisions of the software removed the buttons and implemented a system of context menus with which to construct answers, to further increase the space available for the user's answer, and to make the link between performing an action and its consequence more evident (e.g. right-clicking on a circle and seleting 'Delete', rather than choosing to delete a circle and specifying the target's name); this way, the user was able to interact with the canvas itself, rather than pressing buttons at the side. The many dialogue boxes were combined into one (*Fig. 3.6*), used for both the creation and modification of circles.

Answer-checking was also made automatic: as soon as the total reached a value equal to that of the answer to the question, the user was immediately told so, and moved on to the next question. The intention here was to streamline the user's experience, increasing their efficiency (i.e. the speed with which question sets were completed).

This design was ultimately abandoned as, again, it separated the user from their own work too much. Although this design was an improvement, allowing the user to interact directly with the canvas to perform operations upon circles that were able to specified by pointing with the mouse, rather than by specifying an action and typing out a name, the dialogue box was too complex, and still entailed the *premature commitment* of the user by requring a circle's specification before creating one.

The automatic answer-checking was also a problem, as it did not give the user a chance to review their answer before proceeding. Indeed, it did not give the user a chance to learn from their mistakes; for example, if they had accidentally reached the right answer as part of a larger, incorrect calculation that they had previously

intended to make.

The final revision of the software reintroduced the `Check` button to address the issues with the automatic answer-checking in the previous version of the application, and replaced the system of context menus with a new control scheme, as discussed in *Sec. 3.2.1*.

OLD VERSIONS OF THE APP

Figure 3.5: *Nico*'s user interface went through a number of revisions before reaching its current state.

3.2.1.1 Dialogue Boxes

Dialogue boxes have been a consistent feature of the user interface since its first revision. Initially, dialogue boxes were triggered by the buttons at the side of the application (*Fig. 3.5*), and consisted of a simple text field, into which the user typed a circle's name, followed by, in the case of the creation and editing of circles, an S-expression specifying the mathematical expression that the circle was to represent.

This was never intended to be the final version of the user interface, but was useful for testing the rest of the user interface and interaction handler at the time. Requiring a user to be familiar with LISP syntax ran somewhat counter to the intentions of this project, being to provide an accessible alternative notation for arithmetic for the non-programmer, pre-algebra user.

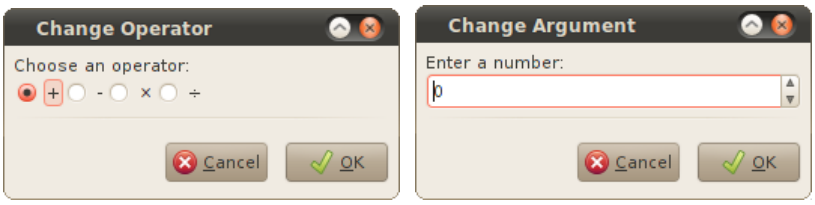
GRAND UNIFIED DIALOGUE BOX

Figure 3.6: The unified dialogue box.

In later revisions of the software, as mentioned in the previous section, the many dialogue boxes were combined into one, with many options available to specify the parameters of a circle. This was implemented to eliminate the need for typing in S-expressions, as in the previous incarnation of the user interface, and dynamically generated its layout, making new circles available to use as arguments as they were created.

It was decided to combine all parameters into one dialogue box (*Fig. 3.6*), rather than many smaller dialogue boxes, to minimise the time spent searching for controls by the user. However, the grand unified dialogue box still required the user to perform operations without it being entirely obvious what the consequences would be, thus

requiring an unnecessary amount of *premature commitment* from the user. In addition to this, the act of visualising the circle to be created whilst setting the many options available in the dialogue box constituted a superfluous *hard mental operation*.



(a) Editing a circle's operator (b) Editing a circle's argument

Figure 3.7: The dialogue boxes used to modify existing circles in the application.

In the final revision of the software, the dialogue boxes were reduced down to two small instances, as shown in *Fig. 3.7*, used in combination with the new control scheme. It was acknowledged that the large dialogue box was unwieldy and dense, requiring the user to search for controls within it and increasing the *viscosity* of the notation by requiring all parameters to be set every time one needed to be edited.

The two simple dialogue boxes served to greatly decrease this *viscosity*, making it so that single arguments or operators could be edited independently of the rest of their circles. In combination with the improved control scheme, this helped to eliminate much of the *premature commitment* of previous designs, by making the creation of a circle an incremental process, rather than a single event.

3.2.2 Control Scheme

Initially, the application used a collection of buttons and dialogue boxes, as detailed above, to control the circles that appeared on the canvas. This was implemented to ensure that the user interface was functional, and so that circle operations could be tested. It was abandoned as it required too much *premature commitment* from the user, who would have to consider the consequences of performing an action before doing so.

In later revisions, a system of context menus was implemented to decrease the separation between the user and their work. Right-clicking on a blank patch of the canvas gave the user the option to create a new circle, and right-clicking on an existing circle presented the user with the option to either edit or delete that circle.

This required less forethought on the part of the user, but the use of the unified dialogue box (*Fig. 3.6*) still entailed some *premature commitment*.

The control system is now predominantly mouse-based for the construction of calculations in the application's notation, as detailed in *Sec. 3.2.1.3*.

Left-clicking on a blank patch of canvas creates a new circle, initialised with two arguments, `?` and ?, and with operator ?. This approach was chosen, rather than providing the user with a null circle such as (0+0), to make it clear to the user that the new circle required attention, and so that it wouldn't interfere with the evaluation of the diagram; a new circle on its own could be found to be a root circle, leading to the possibility that the user's total could suddenly drop to 0.

Right-clicking on a circle increments its number of arguments by one, up to a maximum of eight. Holding the Ctrl key and right-clicking on a circle decrements its number of arguments by one, down to a minimum of two. The user is warned by a popup if the exceed either of these boundaries. The limit of two to eight arguments was decided upon as fewer than two arguments is an unfamiliar and not often useful concept from the perspective of the target audience. Although Clojure is able to evaluate, for example, `(+ 1)` (returning 1), this is liable to lead to confusion on the part of the user. The upper limit of eight arguments was set to prevent circles from becoming overcrowded and thus harder to read.

Circles can be moved around the canvas by left-clicking and dragging the circle to the desired location. If a circle is dragged to the bin icon in the bottom left-hand corner of the screen, then it is removed. Originally, deletion was mapped to a mouse control, but in the interest of a simpler control scheme and with a limited number of mouse buttons, it was decided that an icon should be used instead. This also has the advantage of demonstrating to the new user that work can be easily undone, encouraging exploration.

Left-clicking on a circle's operator brings up a dialogue box with a selection of operators in it, with a radio button for each (see *Fig. 3.7*). Similarly, left-clicking on a circle's argument brings up a dialogue box containing a spinner, which can be used to set a new value for that argument, between -10 and 10.

The spinner is limited to this particular range as this prevents the user from using the software as a calculator. The limited range of numbers available means that the user must still consider strategies such as partitioning or long multiplication, focussing on how to solve the problem, rather than simply transcribing the question into one circle that will complete the calculation for them. The user is not required

to calculate manually expressions using such small numbers, as it is assumed that the target audience are familiar with simple arithmetic.

To use one circle as an argument to another, right-clicking and dragging allows the user to drag a line ending in a circle to the desired slot on the recipient circle. By having the user pull links out from existing circles, as opposed specifying references and links consequently appearing, the direction of dataflow is emphasised.

This new control scheme allows the user to feel more like they are directly manipulating their work in progress, with each action having clear consequences. It reduces the *viscosity* of editing circles from the previous controls schemes of butons and context menus, and greatly reduces the amount of *premature commitment* required of the user when creating circles. It creates an environment that encourages exploration, and does not require the user to spend time searching for information within parts of the user interface.

3.2.3 Colour-Coding

Nico uses colour-coding in various areas of the application, to draw attention to important elements of the application. Colour-coding has been shown to faciliate more effective learning. Indeed, Lamberski and Dwyer note that

> "[...] younger learners, for whom color in passive materials has been found to facilitate performance in less complex concept attainment tasks because of color's attention of motivation elements. However, when younger learners have been given self-paced instructional materials containing color-coded reading and phonic concepts, similar positive results have been attained." [16]

Nico's colour scheme was implemented incrementally over the course of the project. It was originally intended, as shown in *Fig. 2.12*, to be based on how nested a circle was rejected on the grounds that it could become confusing if a circle were reused (see *Fig. 3.8*). Consequently, circles are coloured according to their operators, clearly separating different kinds of operations and making the notation more readable.

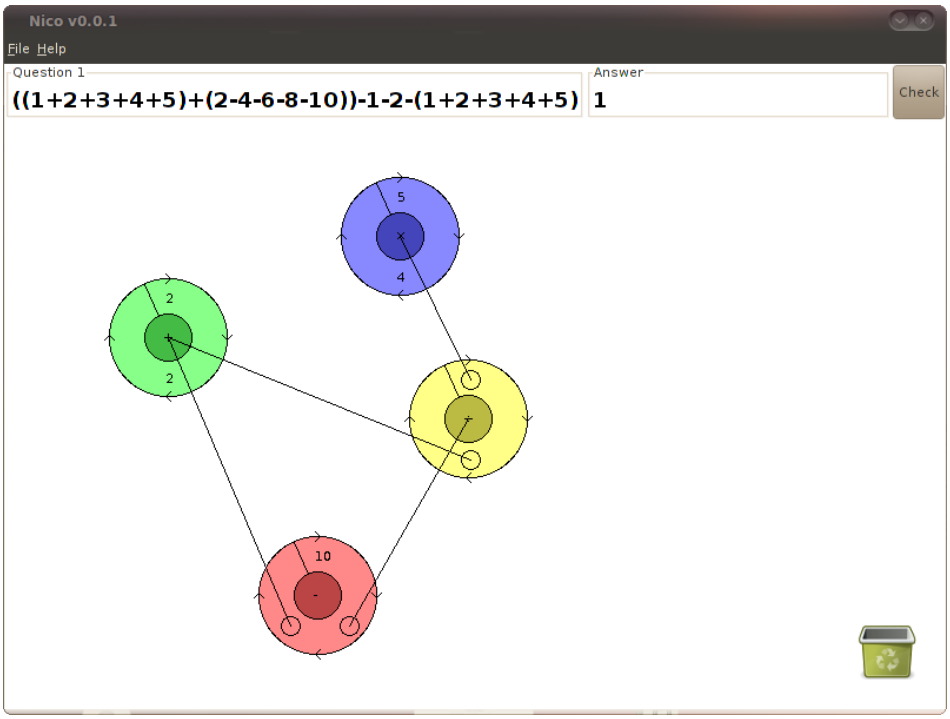
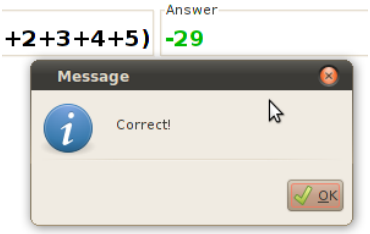


Figure 3.8: An example of where nesting-based colour-coding would be confusing. The circle containing (2+2) can be considered to be both one and two circles removed from the root circle.

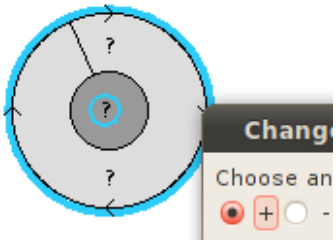
Colour-coding is also used in the information panels: positioning the mouse cursor over a circle highlights the part of the question that it is likely to represent in blue, and also highlights the circle with a blue outline.

When an answer is submitted, the text in the answer panel turns red if the answer was incorrect, or green if the answer was correct, providing an unintrusive form of feedback, allowing the user to continue uninterrupted (i.e. no input is required to continue) if their answer is wrong.

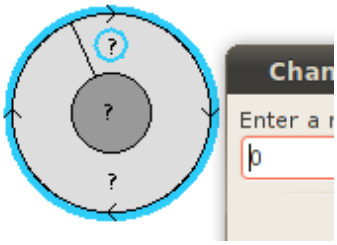
(a) Red text indicates that the answer submitted was incorrect



(b) Green text indicates a correct answer; a dialogue box allows the user to progress to the next question when they are ready



(c) The circle's operator is highlighted when it is being edited



(d) An argument is highlighted when it is being edited

Figure 3.9: Examples of colour-coding in *Nico*.

3.2.3.1 Colour Blindness

As *Nico*'s notation and user interface are quite reliant upon colour to express or emphasise information, colour-blind users are liable to find some parts of the application confusing. As such, although no functionality to address this was included in the application itself, the colours used thoughout are easily replaced in the source code, being defined early on and with the variables, rather than simply numbers, referred to throughout. By changing the variables for each circle's colours, it is possible to create customised versions of the software to suit individual cases of colour blindness.

3.2.4 Notation

OLD VERSIONS OF THE NOTATION

Figure 3.10: The notation used also went through a number of revisions before reaching its current state.

The notation itself was also revised several times during the development of the application. The original vision of the notation can be seen in *Fig. 2.12*. It was decided to move slightly away from this model, as much of the circles' decoration occupied an unnecessary amount of screen estate. As such, it was decided to replace the curved link design with simple lines, allowing many linked circles to be closer together but still readable.

The first implementation of the notation consisted of uniformly-coloured circles connected by lines feeding into the centre of the recipient circle. The lines fed into the centre of the circle so that links to circles would not restrict further the number of available arguments per circle. The colour scheme had simply not yet been implemented.

There were no placeholder values; these were unnecessary as the system of dialogue boxes meant that at no point could a circle have any aspect of itself in an undefined state. There was also no indication of argument ordering, which is acceptable when the operator is commutative (i.e. addition or multiplication), but for operations where this is not the case, this can quickly become confusing.

Although the ordering and placement of arguments was defined, with no indication of this, the user is required to figure this out through experience, which is counter to the intentions of the software. It should not be required that the user spend a significant amount of time learning how to use the system, especially due to deficiencies in the notation.

Another, greater, problem related to argument ordering was that of the circles that linked to other circles. With all input circles pointing to the operator at the centre of their target circle, it was not at all clear in which order the input circles evaluated, neither relative to each other nor to the other arguments in the circle, which would have been problematic in situations where one circle was used as an argument to several others.

The design of the notation was changed to address the problems listed above. In its final revision, the notation has had a number of features added.

Firstly, the circles are now colour-coded by operator, making it easier for the user to distinguish between different kinds of circles. The circles themselves have also been embellished slightly with the addition of a line denoting the start of the expression, just to the side of the first argument at the top of the circle. Small arrows have also been added to the edge of the circles to indicate in which direction the arguments should be read. Finally, the links between circles have been changed such that they

now occupy an argument slot that could otherwise be taken by a number, with a circle on the end of the link line to clearly show where the link terminates.

3.3 Testing

Throughout the development of the software, unit tests were performed to ensure the integrity of the code. GNU Emacs, in combination with SLIME/SWANK and clojure-mode, includes functionality such that commented-out lines of code can be evaluated at will. As such, it was possible to write unit and integration tests into the main body of my code and to evaluate them whenever a change may have affected how a function behaved. In the interests of tidiness and code readability, many tests were removed as they became unnecessary later on in the development in the project.

Assertions were also used to perform tests: Clojure provides the developer with the ability to specify runtime tests as part of function definition, throwing exceptions if any assertions. Such tests were used in addition to the previous method to unit-test and integration-test the functions within the application.

3.4 Summary

In this chapter, the implementation of the project was discussed in some detail, as well as the challenges that were faced during this development phase.

The software itself fulfils and exceeds the criteria for success outlined in the project proposal, being a complete and usable system that improves greatly, in terms of cognitive dimensions, upon the traditional method of handwritten arithmetic.

Third-party libraries, in particular Seesaw, were used where appropriate.

The application divides into three main sections: the back-end, handling mathematical operations and memory management, the interaction handler, helping the user interface to interface with the back-end, and the user interface itself.

The user interface was revised many times to correct deficiencies in the design that would have led to a poor user experience, and the back-end and interaction handler were also both restructured as new demands upon the software became apparent.

In the next chapter, the software will be evaluated, discussing test results and the user study that was conducted as an extension to the project.

Chapter 4

Evaluation

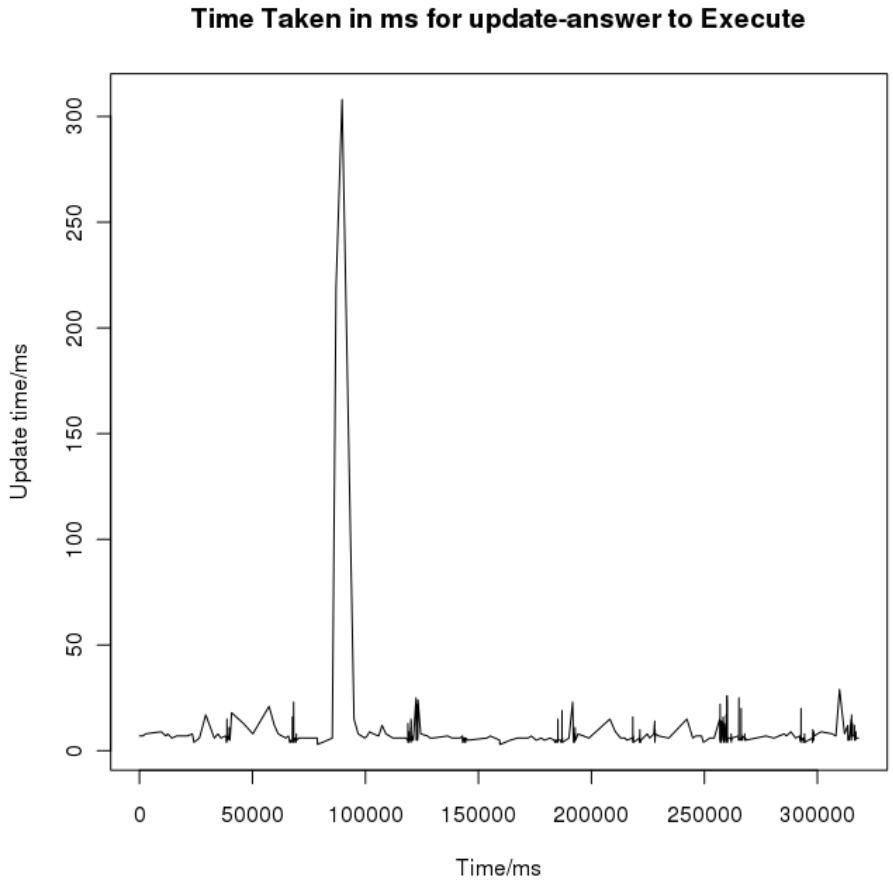
This chapter concerns the testing and evaluation of *Nico*; carried out to determine the quality and utility of the software.

4.1 Testing

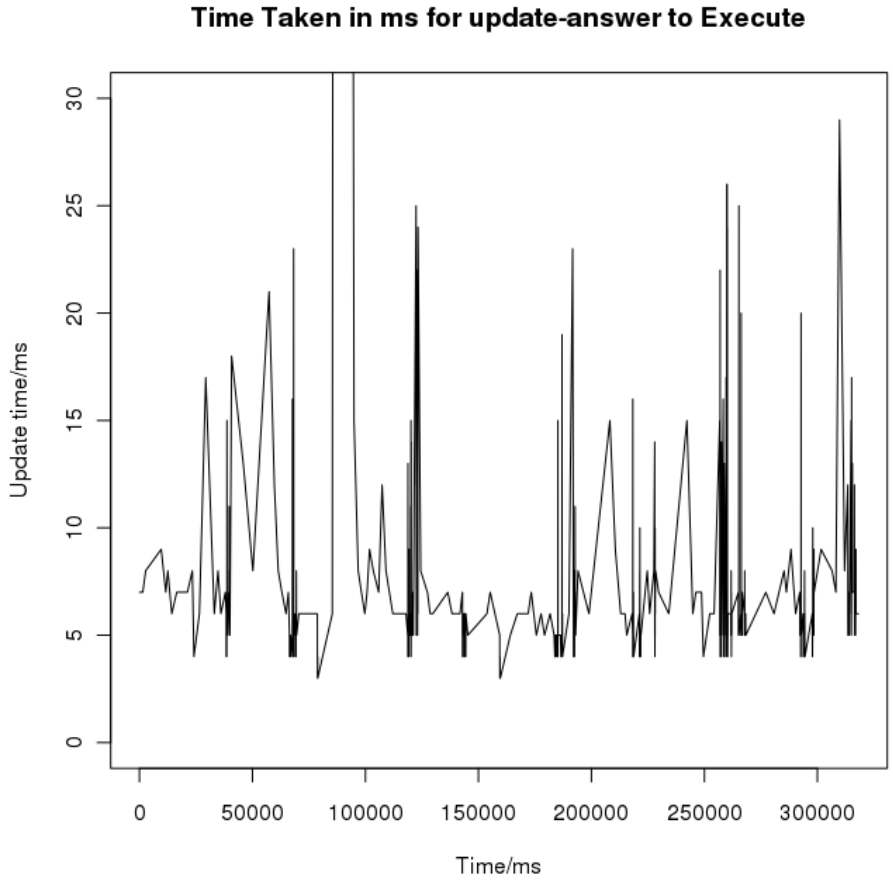
4.2 UI Evaluation

The time taken to generate an S-expression, evaluate it and display the result to the user was required by the project proposal to be less than 300ms. To test this, a *Nico* session was completed, using the same questions as in the user study (below), but with a modified version of the function `update-answer`, which generates an S-expression from the user's diagram, evaluates it and displays the result to the user, that included functionality such that the system time upon entering and exiting the function was recorded in an external log file.

A plot of the time taken for each call to `update-answer` to complete over time is shown below.


Time Taken in ms for update-answer to Execute
Update time/ms
0
50
100
150
200
250
300
0
50000
100000
150000
200000
250000
300000
Time/ms

(a)


Time Taken in ms for update-answer to Execute
Update time/ms
0
5
10
15
20
25
30
0
50000
100000
150000
200000
250000
300000
Time/ms

Nico, in almost all cases in which a measurement was taken, was able to interpret the user's diagram and display the result in much less than the upper limit of 300ms, with a mean time of 6.751852ms. Of the 810 samples taken, there was one anomalous time of 308ms.

4.3 User Study

To assess the utility of the software relative to handwritten arithmetic, a user study was conducted, allowing real users to get to grips with the system, and provide feedback on it. To this end, users were given a tutorial video and a questionnaire (see *Appendix B* for the full questionnaire) to complete whilst using the software.

4.3.1 Experimental Design

Originally, it was the intention of this project to perform a user study upon a 'test class' of Year 5 pupils at an actual school, but the ethical complications involved with working with vulnerable individuals were so great as to make such a study impractical, given the time constraints of this dissertation; in addition to obtaining the approval of the Ethics Committee and the permission of a school to conduct the study, consent forms would have to be distributed to the test class, taken home, signed by the pupils' guardians, returned to the school and returned to the author. It was, therefore, decided that the software should be tested upon a group of students not currently reading a 'mathematical' subject, such as the Natural Sciences Tripos, the Mathematical Tripos or the Computer Science Tripos, which necessitate a high level of mathematical aptitude and experience.

Participants were first asked to sign a statement of informed consent, giving their consent to participate in the study, and were then asked if they had previously used *Nico*, and to indicate their confidence in their ability to solve simple mathematical problems on a five-point Likert item.

Users were then divided into two groups. The first watched the tutorial video and were allowed to explore the application in a 'sandbox' environment for five minutes. They then completed one question set using *Nico*, and answered the same questions by hand. The second group completed the questions by hand before watching the video and using the software.

The time taken to answer each question was recorded, with *Nico* including functionality to query the system time as the user progressed and the handwritten questions being timed using a stopwatch.

Both groups were then asked to complete a series of questions to provide feedback on the software, taking questions from Blackwell and Green's sample questionnaire. [4]

This approach was chosen for a number of reasons. *Nico* was compared to questions answered by hand, rather than a calculator, as its intention is not to calculate answers for the student, but as more of a technique of visualising a mathematical method (hence the decision to limit available arguments to functions to a range of values between -10 and 10). To compare *Nico* to questions answered using a calculator would not be a useful comparison, as a calculator is able to perform calculations using larger numbers without requiring the user to put into practice similar methods to those that they would use when calculating by hand.

The users were divided into two groups, each completing the same tasks but in a different order, to eliminate any bias that might have arisen from both groups having already completed the questions using one method, before using the other.

The decision to use two methods of evaluation of the software (timing users and the cognitive dimensions questionnaire) allows several aspects of the software to be assessed: by timing users' responses to questions both by hand and using the software, we are able to gauge the length of time taken by the participant to think about how to answer a question.

It is expected that *Nico* should decrease the amount of time taken to work through complex problems involving several subcalculations. The cognitive dimensions questionnaire also allows users to appraise the design of the user interface themselves, criticising aspects they disliked and praising those that they found useful.

With regard to the design of the questionnaire itself, it was decided to use a Likert item to assess the users' initial level of confidence in their arithmetic ability as this provides a simple approach that can easily map to numerical data, which can then be analysed more easily than a verbal response.

A tutorial video was decided upon, rather than a textual approach, as it provides a means of demonstrating an example session in a standardised way, rather than a live demonstration, in which important elements could potentially be accidentally omitted by the demonstrator, or a textual description, which may not make certain

terminology or processes as clear to the reader as a demonstration.

4.3.2 Pilot Study

A pilot study was conducted with one participant, to assess the feasibility of the experiment that had been designed. The outcome of this study was to make changes to several areas of the test, and a small change to the control scheme of the application itself.

Firstly, the participant in the pilot study noted that the original version of the tutorial video for *Nico* was hard to follow; it used subtitles to convey information, which the participant felt were hard to read and understand in the time given.

The participant also noted that some of the questions in the feedback section were confusing or seemingly irrelevant to this particular application, and was unsure of how to answer them.

The final piece of feedback that the participant provided was that the control scheme of the application seemed counterintuitively focussed upon the right mouse button, where it would have felt more natural to the user to have used the left.

In the interests of improving the study, I acted upon the participant's advice. The video was reproduced using a voiceover rather than subtitles, making it easier to follow. Some of the less relevant questions were removed from the feedback section, leaving those more applicable to *Nico*. Finally, the control scheme was revised, assigning the left mouse button to primary functions such as the creation and editing of circles, and the right to functions such as changing the number of arguments.

4.3.3 Results

The results of the tests are shown in *Table 4.1*, listing the times taken for each participant to answer the questions both by hand and using *Nico*.

	Group 1							
	Participant 1		Participant 2		Participant 3		Participant 4	
Question	Hand	*Nico*	Hand	*Nico*	Hand	*Nico*	Hand	*Nico*
Q1	4600	19547	3100	34094	1800	24657	1300	38891
Q2	8800	23313	7000	24329	6300	17437	5400	12031
Q3	20600	53500	13600	36360	10100	40798	10700	31813
Q4	36200	54375	26400	60156	14200	79704	23800	50094
Q5	61900	103344	39600	79844	26500	103221	33400	61064
Q6	213400	224406	190300	215376	84900	172940	181700	154691
Q7	235200	165813	1996000	103376	909000	48922	194200	52595
Q8	259500	306625	282500	158094	147200	138581	256700	155347
Q9	304400	230953	313500	123626	159800	183722	271700	196144
Q10	416500	307734	357700	213642	197100	245161	322200	235364

	Group 2							
	Participant 1		Participant 2		Participant 3		Participant 4	
Question	Hand	*Nico*	Hand	*Nico*	Hand	*Nico*	Hand	*Nico*
Q1	1200	22299	2100	23516	2600	16142	2600	21985
Q2	2200	24530	5600	19015	7200	16875	7200	23422
Q3	17300	47620	7600	29735	18400	27923	15900	48470
Q4	23000	50704	20700	43094	25300	37516	38600	174160
Q5	46600	134828	33300	70313	35500	58954	72400	98846
Q6	199900	629835	136400	164891	99000	113659	266600	252708
Q7	202400	80277	142200	48890	1106000	52954	272900	72767
Q8	295900	248736	182000	107375	184700	78001	348900	257224
Q9	312900	152185	204100	118594	200500	67485	384500	144753
Q10	357300	238300	261600	203110	237500	324581	472900	270177

Table 4.1: Table showing the time in milliseconds taken by each participant to complete each question, by hand and using the software.

This data can be visualised using a series of plots, as below. The complete set of plots can be found in Appendix A.

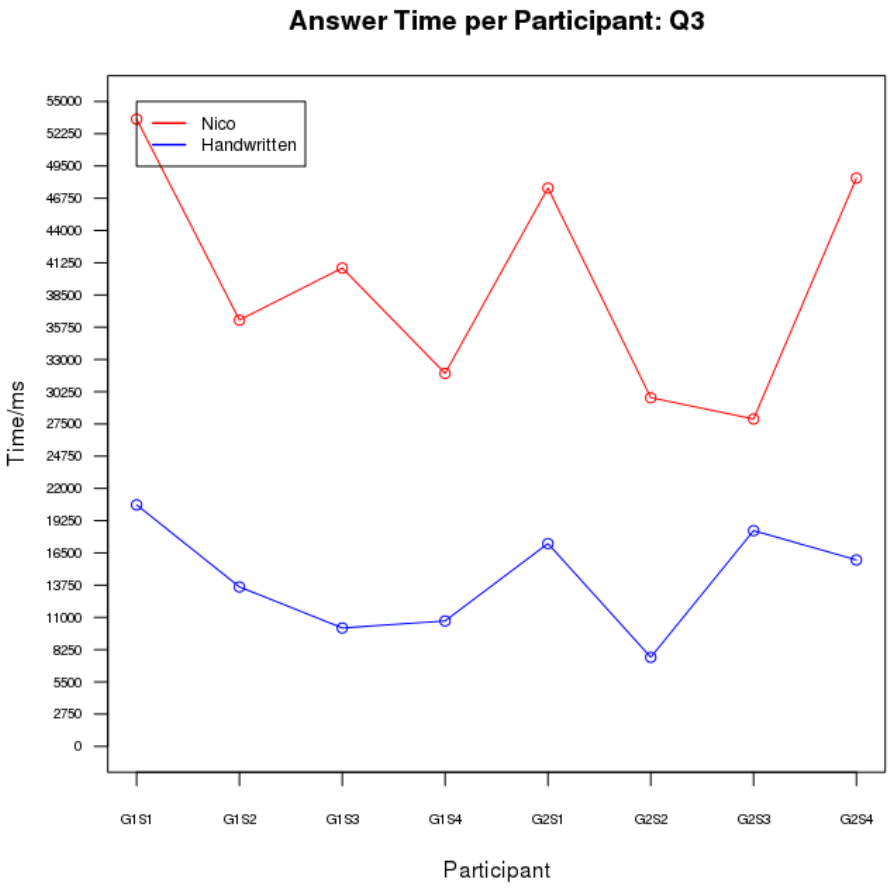


Figure 4.2: Time taken in milliseconds per participant to answer Question 3: 1+2+3+4+5.

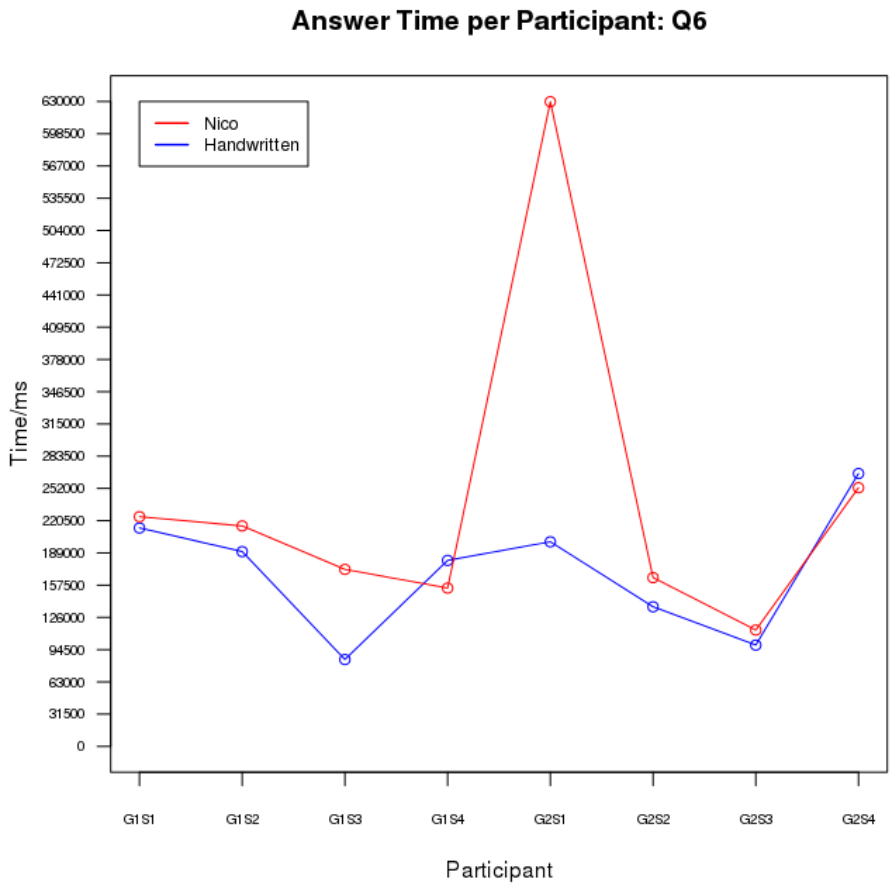


Figure 4.3: Time taken in milliseconds per participant to answer Question 6: ((1+2+3+4+5)+(2×4×6×8×10))×1×2×(1+2+3+4+5).

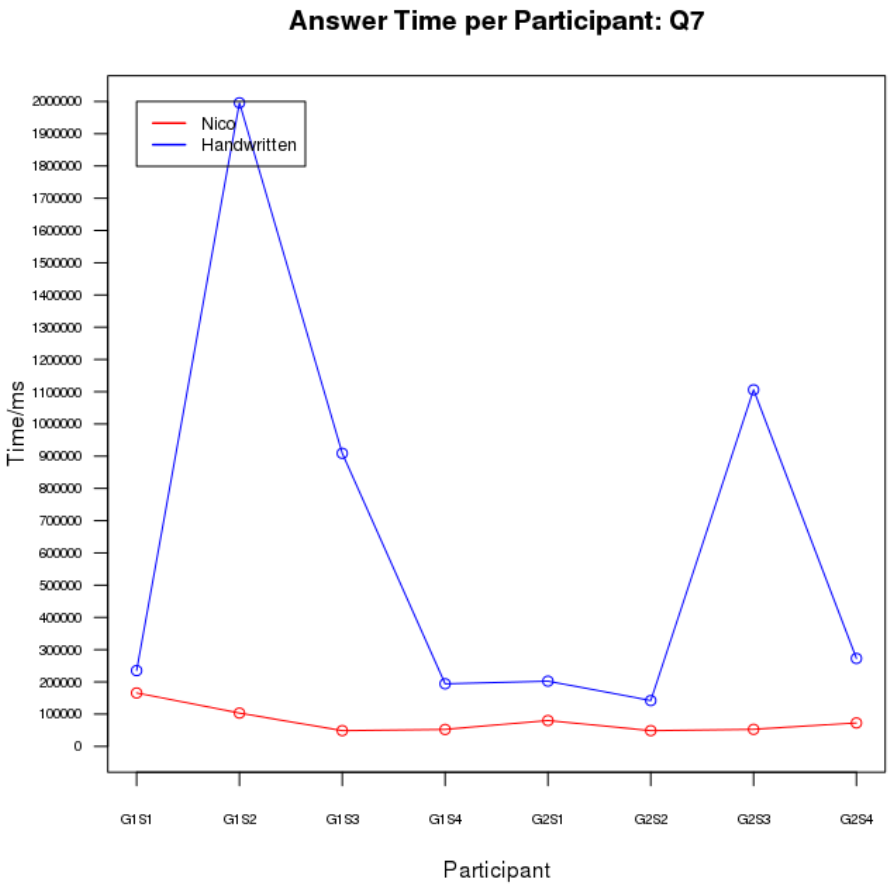


Figure 4.4: Time taken in milliseconds per participant to answer Question 7: 12+14.

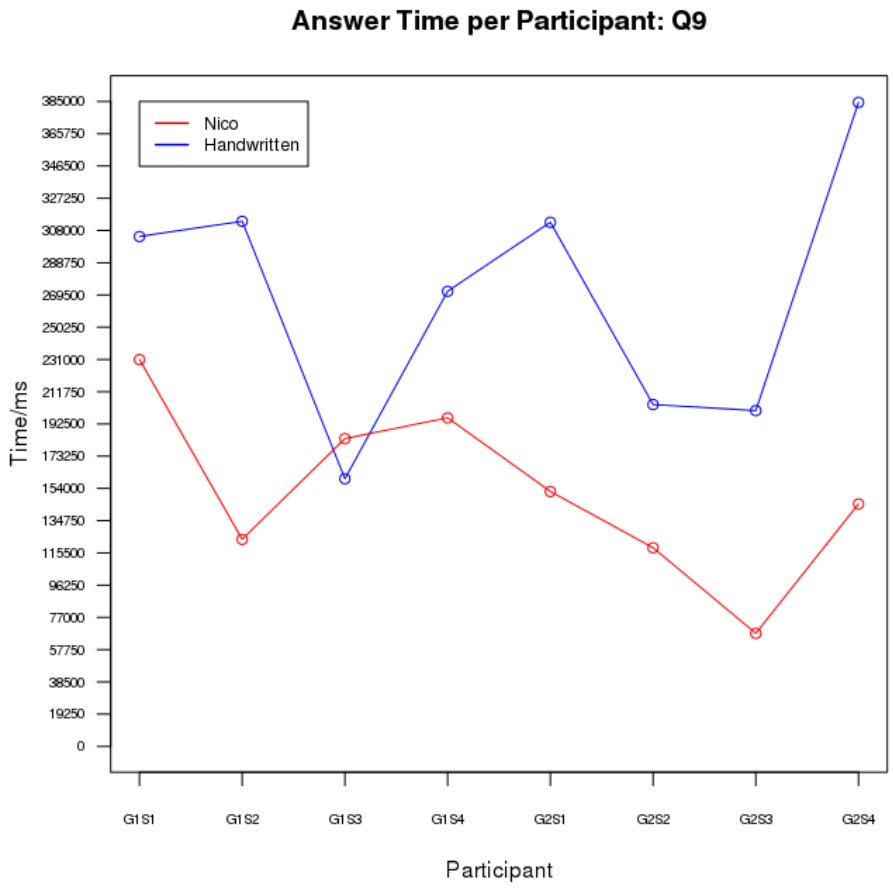


Figure 4.5: Time taken in milliseconds per participant to answer Question 9: 120÷((2×10)+5+5).

A two-sample, paired t-test using an α value of 0.05 was used to determine whether or not the difference between the mean time taken to answer each question by hand and using *Nico* was statistically-significant. The paired t-test was chosen as the samples are not independent. The results of the t-tests on each question's datasets are shown in *Table 4.2*. The data is assumed to fit a normal distribution. A Shapiro-Wilk test was considered to test each dataset for normality, but the sample size of eight data points per set (i.e. eight participants for each question) was deemed to be too small for such a test to have meaningful results.

Question	t	p	Significant?
Q1	-8.0204	0.00008968	Yes
Q2	-8.0493	0.00008764	Yes
Q3	-8.9296	0.00004489	Yes
Q4	-2.9592	0.02113	Yes
Q5	-5.288	0.001138	Yes
Q6	-1.3149	0.2300	No
Q7	2.3688	0.0497	Yes
Q8	3.1	0.01732	Yes
Q9	4.0293	0.005	Yes
Q10	2.1298	0.0707	No

Table 4.2: Table showing the results of the paired t-test comparing the distribution of times taken to answer each question by hand and using the software, using an α value of 0.05.

As the p-values in the above table indicate, there was a significant difference in performance between using *Nico* and completing the questions by hand.

For the first five questions, the time taken to complete the questions is markedly increased by using the software. This is somewhat to be expected; the first five questions were 'warm-up' questions, requiring a maximum of three separate calculations to solve and with all numbers involved in each calculation being less than ten. Such questions were included to give the user a chance to get used to solving simple problems using *Nico*, before moving on to more complex tasks.

By the sixth question, there is no longer a statistically-significant difference between using the software and completing the questions by hand. This particular question still uses numbers less than ten, but requires several more nested calculations to solve, suggesting that *Nico* offers some improvement where the handwritten method is lacking, but also that this is offset by the greater time required to input very simple expressions into the application.

The final four questions introduced numbers greater than ten, include many subcalculations and large numbers. For the seventh, eighth and ninth questions, *Nico* offers a statistically-significant improvement. These questions involve the manipulation of larger numbers, requiring strategies that go beyond simply recalling number bonds, such as partitioning or long multiplication. It is these types of questions that *Nico* is intended to make easier for the user, and the often-dramatic decrease in the time taken to solve such questions is shown to great effect here.

The tenth question, on reflection, was not well-chosen. It is comparable in difficulty to the fifth question, being a series of simple subcalculations with numbers less than ten. Though there are more calculations to perform, and one operation involving a large number at the end, this question is not especially challenging to solve by hand, and as such the lack of a significant difference between using the software and not doing so is not entirely unexpected. The simple subcalculations are easier to perform by hand, and combining them is faster using *Nico*.

This conclusion is borne out by the plots shown above: *Fig. 4.2* shows a clear advantage to calculating a simple question by hand. By Question 6, (*Fig. 4.3*), there seems to be no clear advantage to using either method, and for Questions 7 and 9 (*Fig. 4.4* and *Fig. 4.5*), which are 'complex' questions as described above, *Nico* is shown to be a faster means of calculation. This is especially the case with Question 7. Data were also collected regarding how well the questions were answered by each user, which follow.

	Group 1							
	Participant 1		Participant 2		Participant 3		Participant 4	
Question	Marks	Errors	Marks	Errors	Marks	Errors	Marks	Errors
Q1	1	0	1	0	1	0	1	0
Q2	1	0	1	0	1	0	1	0
Q3	1	0	1	0	1	0	0	0
Q4	1	0	1	0	1	0	1	0
Q5	1	0	1	0	1	0	1	0
Q6	0	0	0	0	0	0	0	0
Q7	1	0	1	0	1	0	1	0
Q8	0	0	1	1	1	0	1	0
Q9	0	0	1	0	1	1	1	1
Q10	1	0	0	0	1	0	1	0

	Group 2							
	Participant 1		Participant 2		Participant 3		Participant 4	
Question	Marks	Errors	Marks	Errors	Marks	Errors	Marks	Errors
Q1	1	0	1	0	1	0	1	0
Q2	1	0	1	0	1	0	1	0
Q3	1	0	1	0	1	0	1	0
Q4	1	0	1	0	1	0	1	2
Q5	0	1	1	0	1	0	1	0
Q6	0	1	1	0	0	0	0	0
Q7	1	0	1	0	1	0	1	0
Q8	0	0	1	0	1	0	1	1
Q9	1	0	1	0	1	0	1	0
Q10	0	0	1	0	0	5	1	0

Table 4.3: Table comparing the mark obtained for each question answered by hand (a correct answer scored 1, whereas an incorrect answer scored 0) to the number of attempts made to answer each question before submitting the correct answer in *Nico*.

The Pearson product-moment correlation coefficient r of this data was calculated, comparing the concatenated 'marks' data and the concatenated 'tries' data (two series of 80 elements each). The value obtained for r was -0.2278926, indicating a weak negative correlation between the mark obtained and the number of wrong submissions using *Nico*. To determine the significance of this result, the r value is compared to a critical value taken from a table, in this case from Fisher, who gives,

for a test at the 0.05 level and for 80 samples, a critical value of 0.2172. [10] Hence, the calculated r value shows a statistically-significant, negative correlation between the marks obtained for a handwritten answer and the number of wrong answers provided using the software.

4.3.3.1 Feedback

The final section of the questionnaire constituted a series of questions adapted from Blackwell and Green's paper. [4] The users' responses were graded based upon their tone, scoring -1 for negative, 0 for neutral and 1 for positive. The results follow.

	Group 1				Group 2				Totals		
Question	P1	P2	P3	P4	P1	P2	P3	P4	+	±	-
2.1.1	1	1	-1	0	1	1	1	1	6	1	1
2.1.2	-1	1	-1	-1	-1	-1	1	-1	2	0	6
2.1.3	1	1	1	1	1	1	1	1	8	0	0
2.2.1	1	1	1	0	1	1	1	1	7	1	0
2.2.2	-1	1	-1	-1	-1	1	1	1	4	0	4
2.3.1	-1	-1	-1	-1	-1	1	0	0	1	2	5
2.3.2	-1	1	0	-1	0	0	-1	1	2	3	3
2.4.1	1	1	-1	1	-1	1	1	1	6	0	2
2.4.2	-1	-1	-1	-1	-1	1	1	1	3	0	5
2.5.1	-1	1	-1	1	1	1	1	1	6	0	2
2.5.2	0	1	0	0	-1	1	1	1	4	3	1
2.5.3	0	1	1	1	0	1	1	1	6	2	0
2.6.1	1	1	0	0	1	1	1	1	6	2	0
2.6.2	-1	-1	-1	-1	-1	1	0	1	2	1	5
2.7.1	-1	1	-1	1	0	1	1	1	5	1	2
2.7.2	0	-1	-1	1	1	0	1	0	3	3	2
2.7.3	0	1	0	1	0	0	1	0	3	5	0
2.8.1	1	1	1	0	1	1	1	0	6	2	0
2.9.1	1	0	0	0	0	1	0	0	2	6	0
2.9.2	0	0	-1	0	0	1	0	0	1	6	1
2.9.3	0	0	0	1	0	0	0	0	1	7	0
3.1	0	0	1	0	0	0	1	1	3	5	0
3.2	0	1	-1	0	-1	1	-1	-1	2	2	4

Table 4.4: Table showing the tone of feedback (-1 indicates negative feedback, 0 indicates neutral feedback, 1 indicates positive feedback) by participant and question, with the totals for each type of feedback by question.

This data can also be visualised using a diagram, as follows.

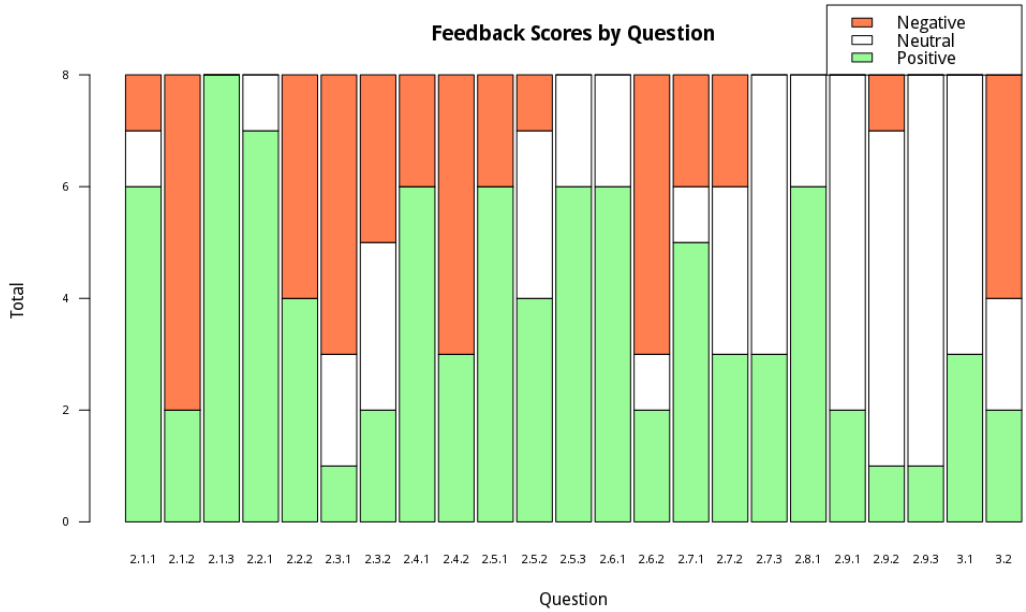


Figure 4.6: Diagram showing the proportions of positive, neutral and negative feedback per question.

The mean feedback scores both by question and by topic are shown below.

Topic	Question	Mean Score
Visibility & Juxtaposability, μ = 0.3750000	2.1.1	0.625
	2.1.2	-0.500
	2.1.3	1.000
Viscosity, μ = 0.4375000	2.2.1	0.875
	2.2.2	0.000
Error Proneness, μ = -0.3125000	2.3.1	-0.500
	2.3.2	-0.125
Closeness of Mapping, μ = 0.1250000	2.4.1	0.500
	2.4.2	-0.250
Role Expressiveness, μ = 0.5416667	2.5.1	0.500
	2.5.2	0.375
	2.5.3	0.750
Hidden Dependencies, μ = 0.1875000	2.6.1	0.750
	2.6.2	-0.375
Progressive Evaluation, μ = 0.2916667	2.7.1	0.375
	2.7.2	0.125
	2.7.3	0.375
Provisionality, μ = 0.7500000	2.8.1	0.750
Secondary Notation, μ = 0.1250000	2.9.1	0.250
	2.9.2	0.000
	2.9.3	0.125
Feedback, μ = 0.0625000	3.1	0.375
	3.2	-0.250

Table 4.5: Table showing the mean feedback values (using the same scoring system as in *Table 4.4*) by question and by groups of questions.

By looking at the mean values by topic, it is possible to determine roughly how well-received the software was, with regard to each aspect that the test participants were questioned on.

The mean feedback score across all questions was also taken, giving a value of 0.25the feedback was broadly neutral-to-positive.

4.3.3.1.1 Visibility & Juxtaposability

μ = 0.3750000

Feedback with regard to the software's visibility and juxtaposability was broadly netural-to-positive. One participant described the notation as "very intuitive to use", although another noted that "circles +[sic] operators could be larger". Questions 2.1.1 and 2.1.3 received mostly positive feedback; the mean feedback score for this section is lowered somewhat by the mean score of -0.5 for Question 2.2.2, in which several users noted that the connecting lines between circles could occasionally become confusing when there were particularly many.

4.3.3.1.2 Viscosity

μ = 0.4375000

Feedback on the perceived level of viscosity in the software was also neutral-to-positive, verging upon positive. Question 2.2.1 received mostly positive feedback, whereas Question 2.2.2 received neutral feedback. Users noted that it was "very easy" to make changes to previous work, and that it was "easy to do and obvious how to do so". However, they also felt that it was "more difficult to get rid of arguments [using Ctrl]". Another user concurred with this: "deleting additional numbers [...] is the most difficult, but still simple to remember".

4.3.3.1.3 Error Proneness

μ = -0.3125000

Users responded to the questions in this section with neutral-to-negative feedback, verging on negative for Question 2.3.1. Several users felt that the drag-and-drop mechanism for joining circles was easily misunderstood, having tried to terminate lines in invalid locations, although one user notes (for both questions in this section) that "I didn't make any mistakes!".

4.3.3.1.4 Closeness of Mapping

μ = 0.1250000

This section was met with neutral-to-positive feedback, with Question 2.4.1 receiving more positive feedback and Question 2.4.2 receiving neutral-to-negative reponses. Participants felt that the notation was "very close[ly]" related to the result that they were describing, with one user going so far as to say that it was "exactly so".

Several users found that the limitation of available numbers to the range -10 to 10 to be a strange way of working: this limitation was not mentioned in the tutorial video, with the expectation that users would appreciate the need for such a feature, to prevent the software from becoming equivalent to a calculator, rather than a means of expressing method. As it was particularly difficult to explain this without imposing a subjective method of calculation upon the user, some participants concluded that it was a flaw in the software, rather than a conscious design choice.

4.3.3.1.5 Role Expressiveness

μ = 0.5416667

Feedback on this section was broadly positive, with Question 2.5.3 receiving particularly positive responses. User opinions were varied with regard to the clarity of the notation. A small majority felt that it was "well laid out", and several users found the colour-coding "useful". Some, however, found that they were liable to forget the meaning of some parts of the notation. One user felt that

> "if I were looking at a completed diagram it might be confusing but when you work through a question and break it down it is easy to see what's what because you create each part of it as you get there."

Most users felt that there were no parts of the notation that were especially difficult to interpret, although a few note that the screen can become busy for more complex calculations. One user agrees with the feedback from the 'Visibility & Juxtaposability' section from that connecting lines between circles can become "like a spiderweb, difficult to take in altogether" when numerous. All users responded to Question 2.5.3 with "no", "none" or equivalent answers, indicating that there were no parts of the notation that were unclear but included as a result of a feeling of obligation to do so.

4.3.3.1.6 Hidden Dependencies

μ = 0.1875000

Feedback on this section was neutral-to-positive, though closer to neutral. Question 2.6.1 received a mostly positive response, whereas the feedback on Question 2.6.2 was mildly negative. Most users found that dependencies between units of the notation were made very clear, apart from one user who found the dependencies clear, but found it difficult to infer the results of their calculations whilst constructing them, and one user who neglected to answer the question. Once again, several users found that the notation became harder to read, and dependencies

harder to follow, when the number of circles on-screen was especially large. This suggests that either circles should be made smaller, or the canvas made larger, or that an alternative to lines, which can become "tangled", should be sought as a means of linking circles.

4.3.3.1.7 Progressive Evaluation

μ = 0.2916667

Once again, feedback was neutral-to-positive. Users found the running total in the information panel to be a useful means of tracking their progress, but noted that it was difficult to keep track of the value of partially-completed calculations where several trees of circles were kept separate until the end, due to the single answer display in the information panel.

4.3.3.1.8 Provisionality

μ = 0.7500000

Response to this question was very positive. Users found that the running total displayed in the information panel, coupled with the prominent bin icon to faciliate easy deletion, allowed them to explore potential solutions before submitting their final answer. One user comments:

> "I could create notations [i.e. circles] and throw them in the bin if they didn't help."

This is in accordance with the intention of the project, which is to allow users to explore potential solutions, and to be able to determine more easily why a given method may or may not work.

4.3.3.1.9 Secondary Notation

μ = 0.1250000

Feedback here was positive, though only mildly so. A means of making notes or comments is not provided in the software, though this was neither well- nor poorly-received by the users. Several users mentioned that, given to opportunity to annotate their diagrams in paper form, they would make notes keeping track of the totals of subcalculations: "[I would] keep a track of the different sums I had done so far". Users commented that they did not add any extra marks in the software to clarify or emphasise the information displayed onscreen.

4.3.3.1.10 Feedback

μ = 0.0625000

Feedback in this section was largely neutral. Question 3.1 received slightly positive responses, whereas Question 3.2 received slightly negative feedback. Most users did not feel that they were using the software in a way which was 'wrong' or 'unusual', though one user did feel that having to create larger numbers from the limited range available constituted such behaviour, once again suggesting problems with the tutorial. Furthermore, several users suggested that an improvement to the software would be to allow numbers outside of the range of -10 to 10 to be used. Others felt that the circles should be larger, or that the canvas should be more capacious.

4.4 Summary

The criterion for the success of this project, as quoted at the beginning of Chapter 3, was

> "[to be] able to generate an abstract syntax tree in Clojure from the graphical language and evaluate such a tree, passing the results back to the graphical application and displaying this to user in less than 300ms."

In this regard, *Nico* has met and far exceeded expectations. A functioning application and accompanying visual metaphor have been developed that allows the user to express a calculation graphically. From this, the application is able to generate a valid Clojure S-expression – the tree – that is able to be passed around and evaluated. The application takes considerably less than 300ms to interpret the notation and update the user-facing display accordingly.

As an extension, a user study was conducted to assess the suitability of the software to complement traditional methods of mathematical education in the target audience of Year 5 pupils. The results of this study can be summarised thus:

- There is a statistically-significant difference between the time taken to solve questions by hand and by using the software
  - For simple questions that are easily solved mentally, *Nico* increases the solution time
  - For more complex questions involving large numbers and nested calculations, *Nico* decreases the solution time

- There is a statistically-significant negative correlation between whether a question is answered correctly or incorrectly by hand and the number of wrong attempts at an answer before succeeding using the software
- Feedback on the software was broadly neutral-to-positive, with users concluding that:
  - The notation was clear, with few hidden dependencies
  - For a large number of circles, the canvas could sometimes become overcrowded
  - The software encourages exploration, and it is easy to try out many potential solutions
  - The limitation of numbers to the range -10 to 10 was confusing, especially as it was explained poorly

The final chapter summarises what has been achieved and what has been learnt over the course of this project, and draws some final conclusions.

Chapter 5

Conclusions

Overall, this project has been very successful. Not only has an application meeting the requirements specified for success been developed, with a suitable new metaphor for calculation, it has also been shown to have a statistically-significant effect on the calculation speed and correctness of a group of users.

The software itself comprises a three-layer structure, with a back-end that interprets and manipulates user-input data, an interaction handler that provides an iterface between the back-end and the user interface, and the user interface itself, the environment in which the user performs actions, interacting with the notation to express calculations.

The user study conducted as an extension to the main project concluded that *Nico* made a statistically-significant difference both to the users' speed of answering questions and to the users' likelihood to answer questions correctly. It was observed that for simple questions involving small numbers and few steps in the calculation, the software increased the time taken to answer. For more complex questions, with many subcalculations and larger numbers, the software was shown to decrease the time taken for users to answer.

There was also a statistically-significant negative correlation between the number of marks obtained in handwritten tests and the number of incorrect answers submitted to the software.

User feedback was mildly positive, with users finding that the software made it easy to explore many possible solutions, but also that the notation could become confusing where a large number of circles occupied much of the canvas' available

area. However, it was also noted that the notation had few *hidden dependencies*, and was easy to read in most cases.

The project suggests a number of potential improvements and extensions:

- To improve the notation in accordance with the feedback received from this user study
- To consider other metaphors for calculation as an alternative to the handwritten method
- To conduct a larger-scale user study with a similar premise, with participants from the target audience
- To conduct a study into the long-term effects of using such software as an aid to learning in a real school environment

The application of Petre and Green's 'Cognitive Dimensions' framework to metaphors in educational software might yield further worthwhile research, extending Blackwell and Green's research into visual programming languages into the domain of educational software. [12] [3]

This project will be further developed and maintained in the future, with a view to its introduction into actual learning environments.

Bibliography

[1] R. Abraham. The Trouble with Math. *Educating the Whole Child for the Whole World: The Ross School Model and Education for the Global Era*, pages 125–137, 2010.

[2] A. F. Blackwell and T. R. G. Green. Cognitive dimensions of information artefacts: a tutorial, 1998.

[3] A. F. Blackwell and T. R. G. Green. Does metaphor increase visual language usability?, 1999.

[4] A. F. Blackwell and T. R. G. Green. A cognitive dimensions questionnaire optimised for users, 2000.

[5] A. F. Blackwell and T. R. G. Green. Notational systems – the cognitive dimensions of notations framework. *HCI Models, Theories, and Frameworks: Toward a Multidisciplinary Science*, 2003.

[6] L. Bragg. Students' impressions of the value of games for the learning of mathematics. In *Proceedings of the 30th conference of the international group for the psychology of mathematics education*, pages 217–224, Cape Town, South Africa, July 2006. International Group for the Psychology of Mathematics Education.

[7] L. Bragg. Students' conflicting attitudes towards games as a vehicle for learning mathematics: a methodological dilemma. *Mathematics education research journal*, 19:29–44, 2007.

[8] Oracle Corporation. Mouseevent (java platform se 7). `http://docs.oracle.com/javase/7/docs/api/java/awt/event/MouseEvent.html`, 2012.

[9] G. C. Delacruz, G. K. .W. K. Chung, and E. L. Baker. Cresst report 773: Validity evidence for games as assessment environments, 2010.

[10] R. A. Fisher. *Statistical Methods, Experimental Design and Scientific Inference*. Oxford University Press, 1990.

[11] M. Fogus and C. Houser. *The Joy of Clojure: Thinking the Clojure Way*. Manning Publications, 2011.

[12] T. R. G. Green and M. Petre. Usability analysis of visual programming environments: a 'cognitive dimensions' framework, 1996.

[13] S. Halloway. *Programming Clojure*. Pragmatic Bookshelf, 2009.

[14] Michael Hugill. *Advanced Statistics*. CollinsEducational, 1991.

[15] J. H. Kanner. *The instructional effectiveness of color in television: A review of the evidence*. Department of the Army, Office of the Assistant Chief of Staff for Communications-Electronics, Washington, D.C., 1968.

[16] R. Lamberski and F. Dwyer. The instructional effect of coding (color and black and white) on information acquisition and retrieval. *Educational Technology Research and Development*, 31:9–21, 1983. 10.1007/BF02765207.

[17] Sunset Lake Software LLC. Pi Cubed | Sunset Lake Software. `http://www.sunsetlakesoftware.com/picubed`, 2012.

[18] Acqualia Software Pty. Ltd. Soulver | Acqualia. `http://www.acqualia.com/soulver/`, 2011.

[19] B. A. Myers. Taxonomies of visual programming and program visualization. *Journal of Visual Languages and Computing*, 1:97–123, 1990.

[20] D. Ray. daveray/seesaw. `https://github.com/daveray/seesaw`, 2011.

[21] A. Roca and F. Rousseau. Interactive multimedia and next generation networks. In *MIPS 2004: Second International Workshop on Multimedia Interactive Protocols and Systems (LNCS 3311)*, Grenoble, France, November 2004. ACM.

[22] samaaron and jlr. overtone/live-coding-emacs. `https://github.com/overtone/live-coding-emacs`, 2011.

[23] P. Swan and L. Marshall. Mathematics games as a pedagogical tool. In *CoSMEd 2009 3rd International Conference on Science and Mathematics Education Proceedings*, pages 402–406, Penang, Malaysia, November 2009. SEAMEO RECSAM.

[24] P. Swan and L. Marshall. Mathematics games: Time wasters or time well spent? In *Proceedings of the 10th International Conference "Models in*

Developing Mathematics Education", pages 540–544, Dresden, Saxony, Germany, September 2009. University of Applied Sciences, Dresden.

[25] B. Victor. Kill Math. http://worrydream.com/KillMath/, 2011.

[26] B. Victor. Scrubbing Calculator. http://worrydream.com/ScrubbingCalculator/, 2011.

[27] S. Witamborski. santamon/guiftw. https://github.com/santamon/GUIFTW, 2011.

Appendix A

User Response Times

The plots of the times taken for each user to answer each question follow.

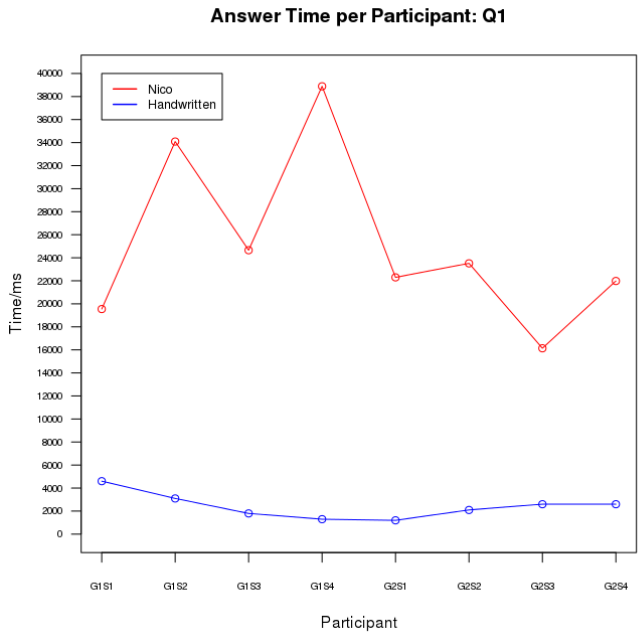


Figure A.1: Time taken in milliseconds per participant to answer Question 1.

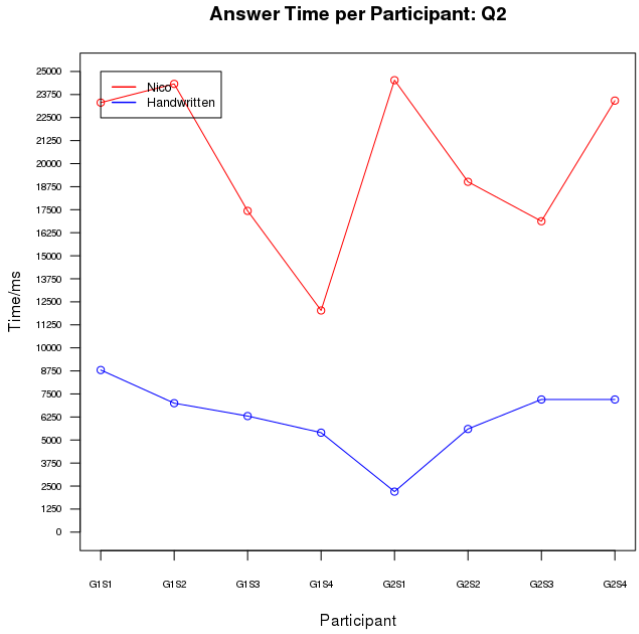


Figure A.2: Time taken in milliseconds per participant to answer Question 2.

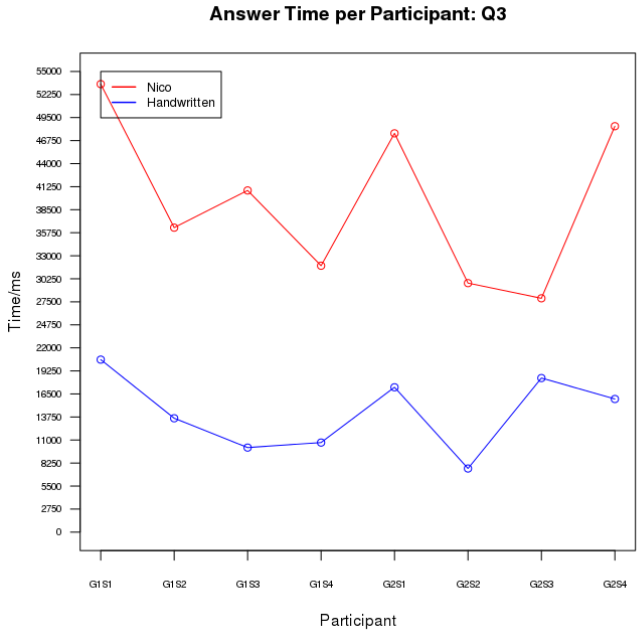


Figure A.3: Time taken in milliseconds per participant to answer Question 3.

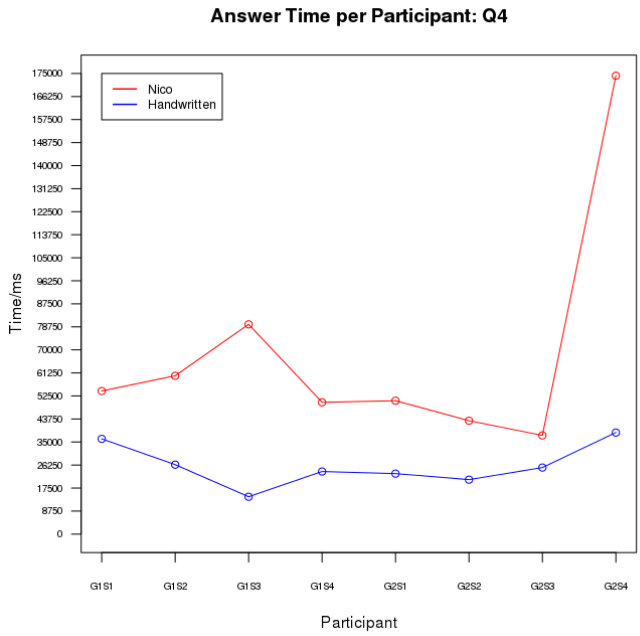


Figure A.4: Time taken in milliseconds per participant to answer Question 4.

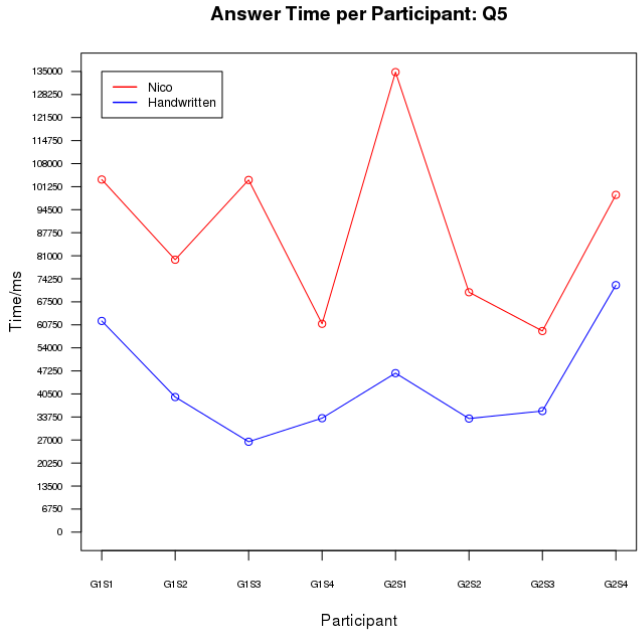


Figure A.5: Time taken in milliseconds per participant to answer Question 5.

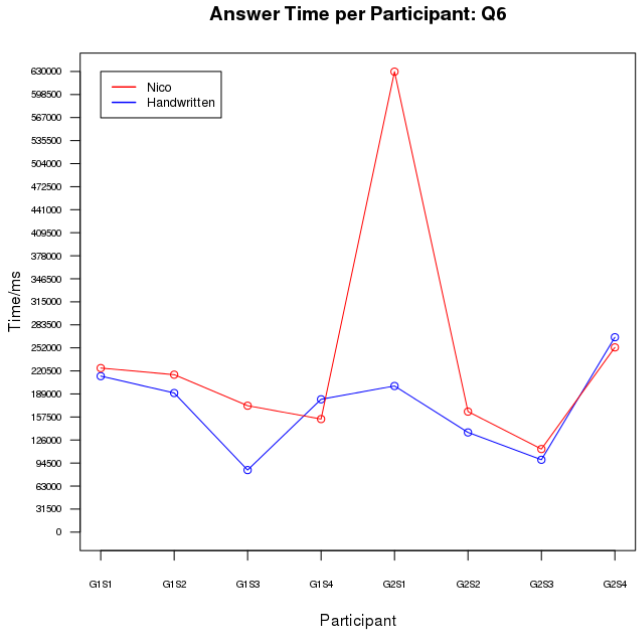


Figure A.6: Time taken in milliseconds per participant to answer Question 6.

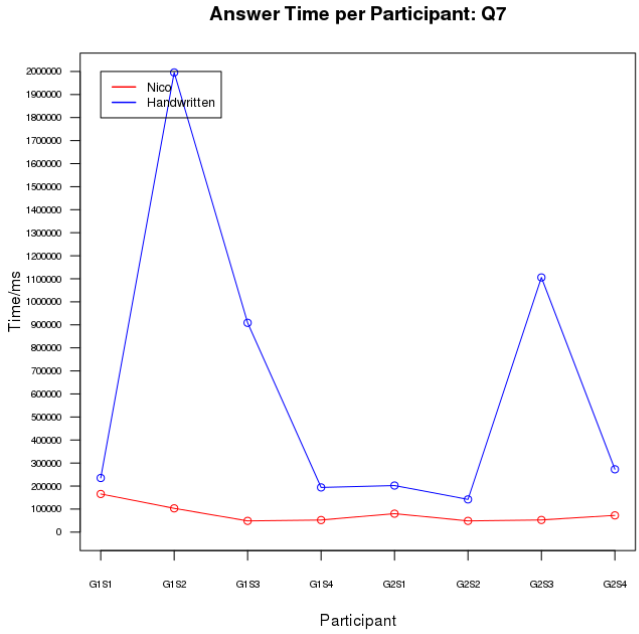


Figure A.7: Time taken in milliseconds per participant to answer Question 7.

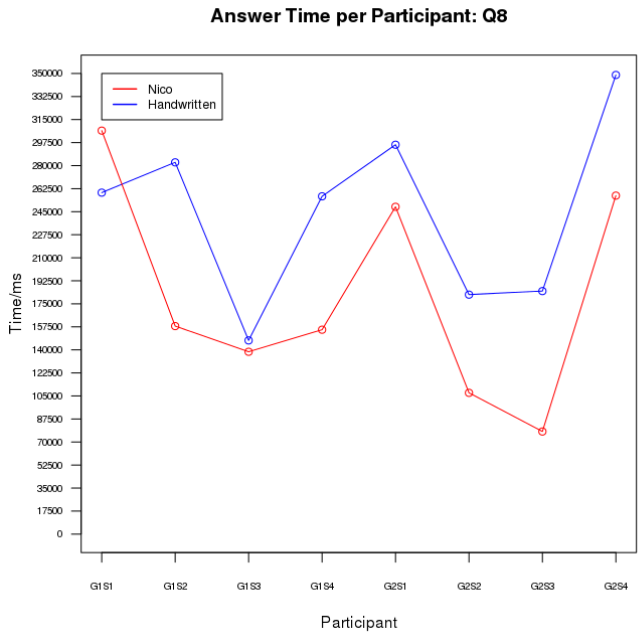


Figure A.8: Time taken in milliseconds per participant to answer Question 8.

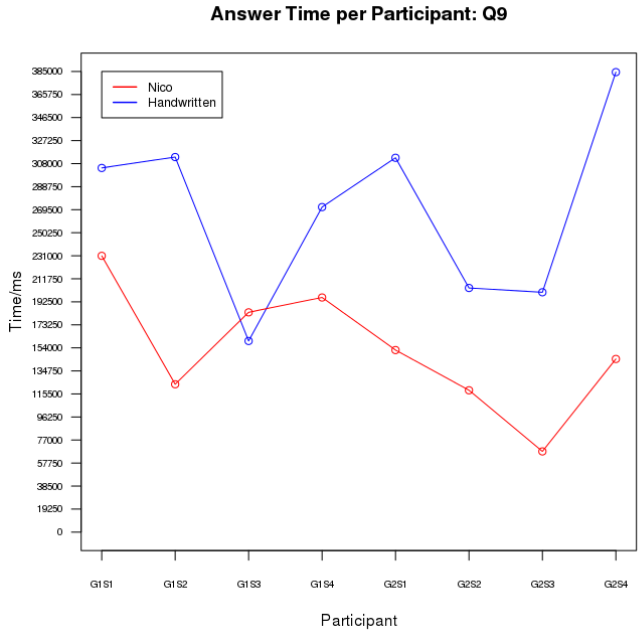


Figure A.9: Time taken in milliseconds per participant to answer Question 9.

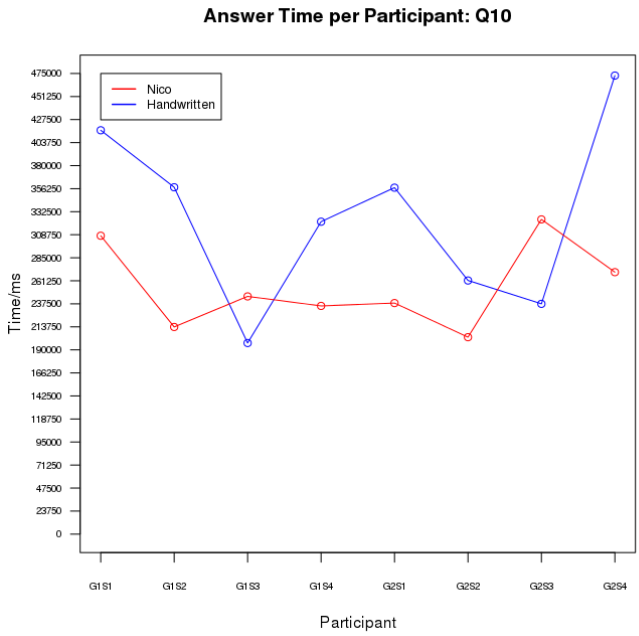


Figure A.10: Time taken in milliseconds per participant to answer Question 10.

Appendix B

Questionnaire

The questionnaire used in the user study follows.

Nico: An Environment for Mathematical Expression in Schools

Thank you for agreeing to participate in the user study for my Part II Project. *Nico* is a piece of educational software designed to aid learners in the visualisation of mathematical problems, by separating out the constituent parts of a calculation into distinct visual units on-screen.

The purpose of this study is to ascertain how well *Nico* achieves its goal of providing a clear, accessible, interactive means of calculation, and to gather feedback on how the application could be improved. The study also aims to compare *Nico* to traditional mathematical methods.

Please review and sign the attached Statement of Informed Consent (Section 1) and please feel free to ask any questions you may have about it.

1 Statement of Informed Consent

Statement of Informed Consent

I state that I am over 18 years of age and wish to participate in a program of research being conducted by Philip Yeeles at the University of Cambridge. I acknowledge that this study has been approved by the University of Cambridge Computer Laboratory Ethics Committee.

The purpose of this research is to assess the usability of a graphical notation and software application for representing mathematical calculations in a graphical manner.

The study involves the use of the application whilst being supervised. I will be asked to complete certain tasks both with and without the application, and I will also be asked open-ended questions about the application and my experience as a user thereof.

All information collected in the study is confidential, and my name will not be identified at any time. I understand that I may ask questions or terminate my involvement in the study freely and at any time without consequence.

I acknowledge that my (anonymised) responses may be published in the final report, and that this report will be made publicly available from the University of Cambridge Computer Laboratory Library and from GitHub.

Signed:

Name:

Date:

THIS PAGE INTENTIONALLY LEFT BLANK

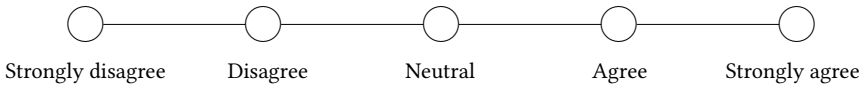
2 Study

Before we begin, please answer the following questions.

- Have you used *Nico* before (circle as appropriate)?

  Yes No

- How well do you agree with the following statement (cross as appropriate)?
  I am confident in my ability to calculate answers to simple mathematical problems.



Thank you. You will now be issued a group number.

- If you are in Group 1, please proceed to Section 2.1.
- If you are in Group 2, please proceed to Section 2.2.

If you would like to stop at any point, please don't hesitate to let me know.

2.1 *Nico*

In this section, you will use *Nico* to solve some simple mathematical problems. The purpose of this section is to evaluate how well *Nico* performs in comparison with manual calculation. We will be keeping a record of the time taken to complete each problem, but please do not let this make you feel rushed. Work at a pace that is normal and comfortable for you.

Before we begin the tasks, please watch the instructional video `tut.ogm` for a briefing on how to use *Nico* and an explanation of its controls.

Now that you have done this, please spend 5 minutes experimenting with *Nico*. Open the application and load the file `qs/blank.nqs` using the file chooser. As you explore, please tell me your thoughts about the application.

Now, let us move on to the problems. Please close the application and open it again, this time loading the file `qs/user-study.nqs`. You will be presented with a series of problems to solve using *Nico*; please solve them.

Thank you very much.

- If you are in Group 1, please continue to Section 2.2
- If you are in Group 2, please continue to Section 3

2.2 Manual Calculation

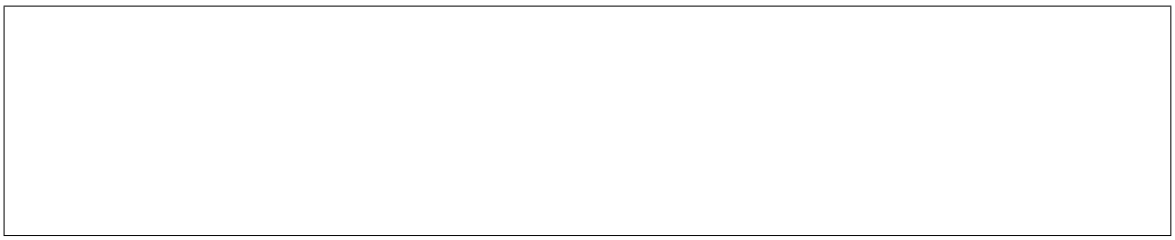
In this section, you will solve some simple mathematical problems using pen and paper. The purpose of this section is as a control, to compare to your results using *Nico*. Once again, we will be keeping a record of the time you take to complete each question, but please do not let this make you feel rushed. Work at a pace that is normal and comfortable for you, and don't forget to show your working.

Let us begin.

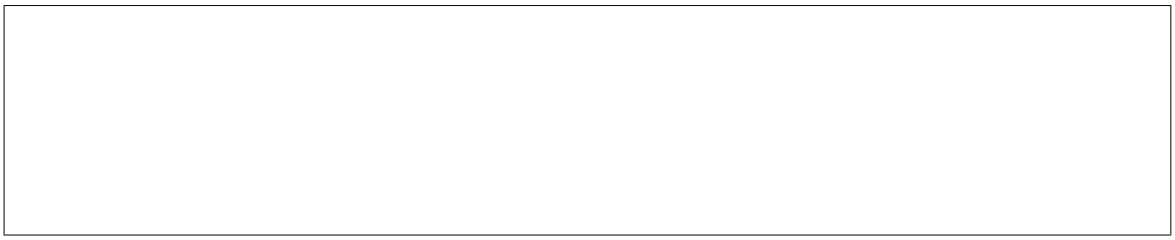
1. *2+3*

2. *9÷3*

3. *1+2+3+4+5*



4. *(2×4)+(3-5)*



5. *((3×4)÷(3+3))×8*

6. *((1+2+3+4+5)+(2×4×6×8×10))×1×2×(1+2+3+4+5)*

7. *12+14*

8. *247×35*

9. *120÷((2×10)+5+5)*

10. *((2+5)×(6÷2)×(9-8))+((3+4)-(5×6))+120*

Thank you very much.

- If you are in Group 1, please continue to Section 3
- If you are in Group 2, please continue to Section 2.1

3 Questions

3.1 Notation

When using the system, what proportion of your time (as a rough percentage) do you spend:

1. Searching for information within the notation ☐ %
2. Translating substantial amounts of information from some other source into the system ☐ %
3. Adding small bits of information to a description that you have previously created ☐ %
4. Reorganising and restructuring descriptions that you have previously created ☐ %
5. Playing around with new ideas in the notation, without being sure what will result ☐ %

3.2 Cognitive Dimensions

3.2.1 Visibility and Juxtaposability

1. How easy is it to see or find the various parts of the notation while it is being created or changed? Why?

2. What kind of things are more difficult to see or find?

3. If you need to compare or combine different parts, can you see them at the same time? If not, why not?

3.2.2 Viscosity

1. When you need to make changes to previous work, how easy is it to make the change? Why?

2. Are there particular changes that are more difficult or especially difficult to make? Which ones?

3.2.3 Error Proneness

1. Do some kinds of mistake seem particularly common or easy to make? Which ones?

2. Do you often find yourself making small slips that irritate you or make you feel stupid? What are some examples?

3.2.4 Closeness of Mapping

1. How closely related is the notation to the result that you are describing? Why?

2. Which parts seem to be a particularly strange way of doing or describing something?

3.2.5 Role Expressiveness

1. When reading the notation, is it easy to tell what each part is for in the overall scheme? Why?

2. Are there some parts that are particularly difficult to interpret? Which ones?

3. Are there parts that you really don't know what they mean, but you put them in just because it's always been that way? What are they?

3.2.6 Hidden Dependencies

1. If the structure of the calculation means that some parts are closely related to other parts, and changes to one may affect the other, are those dependencies visible? What kind of dependencies are hidden?

2. In what ways can it get worse when you are creating a particularly large description?

3.2.7 Progressive Evaluation

1. How easy is it to stop in the middle of creating some notation, and check your work so far? Can you do this any time you like? If not, why not?

2. Can you find out how much progress you have made, or check what stage in your work you are up to? If not, why not?

3. Can you try out partially-completed versions of the calculation? If not, why not?

3.2.8 Provisionality

1. Is it possible to sketch things out when you are playing around with ideas, or when you aren't sure which way to proceed? What features of the notation help you to do this?

3.2.9 Secondary Notation

1. Is it possible to make notes to yourself, or express information that is not really recognised as part of the notation?

2. If it was printed on a piece of paper that you could annotate or scribble on, what would you write or draw?

3. Do you ever add extra marks (or colours or format choices) to clarify, emphasise or repeat what is there already?

3.3 Feedback

1. Do you find yourself using this notation in ways that are unusual, or ways that the designer might not have intended? If so, what are some examples?

2. After completing this questionnaire, can you think of obvious ways that the design of the system could be improved? What are they? Could it be improved specifically for your own requirements?

Thank you very much for your help with my project! Your responses will remain confidential, although they are liable to appear in an anonymised form in the final report, of which a copy will be retained by the University of Cambridge Computer Laboratory Library (`http://www.cl.cam.ac.uk/library/`). The final report will also be available via my repository at GitHub (`http://github.com/loomcore/nico`).

Appendix C

Project Proposal

The original project proposal follows.

Computer Science Part II Project Proposal

Nico: An Environment for Mathematical Expression in Schools

P. M. Yeeles, Selwyn College
Originator: P. M. Yeeles
18 October 2011

Special Resources Required

- PWF account
- SRCF account
- GitHub account
- Toshiba Satellite L500-19X (Intel Pentium T4300 2.0GHz, 4GB RAM, 500GB disk)
- Samsung NC10 Plus (Intel Atom 1.66GHz, 1GB RAM, 250GB disk)

Project Supervisors: Dr S. J. Aaron & A. G. Stead
Director of Studies: Dr R. R. Watts
Project Overseers: Dr J. A. Crowcroft & Dr S. Clark

Introduction

Discussions with local teachers have led me to hypothesise that educational software for mathematics could be used to reinforce learning by focussing on method, rather than on a numerical answer. My aim is to develop a problem-solving system aimed at pupils in year 5 in which the solution to a problem can be represented as a tree of operations – a block-based graphical language to describe mathematical method. The correctness of the solution is then assessed with respect to the structure of the tree. The application will be written in Clojure, using JavaFX 2 for the graphical elements, though if this becomes infeasible I will use either the Eclipse SWT or Swing with GUIFTW. This dissertation will determine whether Nico offers an improvement regarding pupils' ability to recall the correct method for answering mathematical problems. The success of the project will be gauged by whether or not the software is able to generate an abstract syntax tree in Clojure from the graphical language and evaluate such a tree, passing the results back to the graphical application and displaying this to user in less than 300ms[1]. As an extension, I will distribute Nico with anonymous feedback forms to local schools, to determine if the software is actually of use in the classroom.

Work that has to be done

The project breaks down into the following sections:–

1. Core system
    a. A syntax for questions and a means of loading them
    b. A set of basic functions available to the student
    c. A means of inputting an answer that can be evaluated on-the-fly
    d. A means of re-expressing the question to reflect how the student works (e.g. $12 \times 34 \Rightarrow (10 \times 34) + (2 \times 34)$)
    e. A method of validating the answer
    f. A means of tracking the current result of evaluating the method input so far
    g. A system of hints for students who may not know where to start
2. GUI
    a. A collection of drag-and-drop elements that can be used to construct a diagram representing how to solve the question

[1] *Interactive multimedia and next generation networks: Second International Workshop on Multimedia Interactive Protocols and Systems, MIPS 2004 Grenoble, France, November 2004, Proceedings (LNCS 3311)* by Roca and Rousseau has this to say on interactivity: "An abundance of studies into user tolerance of round-trip latency [...] has been conducted and generally agrees upon the following levels of tolerance: excellent, 0-300ms; good, 300-600ms; poor, 600-700ms; and quality becomes unacceptable [...] in excess of 700ms."

   b. A means of validating combinations of the drag-and-drop elements
   c. A means of defining functions
   d. A means of viewing documentation
3. Evaluation
   a. Test software on non-technical but mathematically-able subjects
   b. Evaluate the correctness of Nico's translations between diagram and code
4. Extensions
   a. Create and distribute questionnaires to test classes
   b. Collect and interpret data
   c. Create a tutorial mode for new users

Difficulties to Overcome

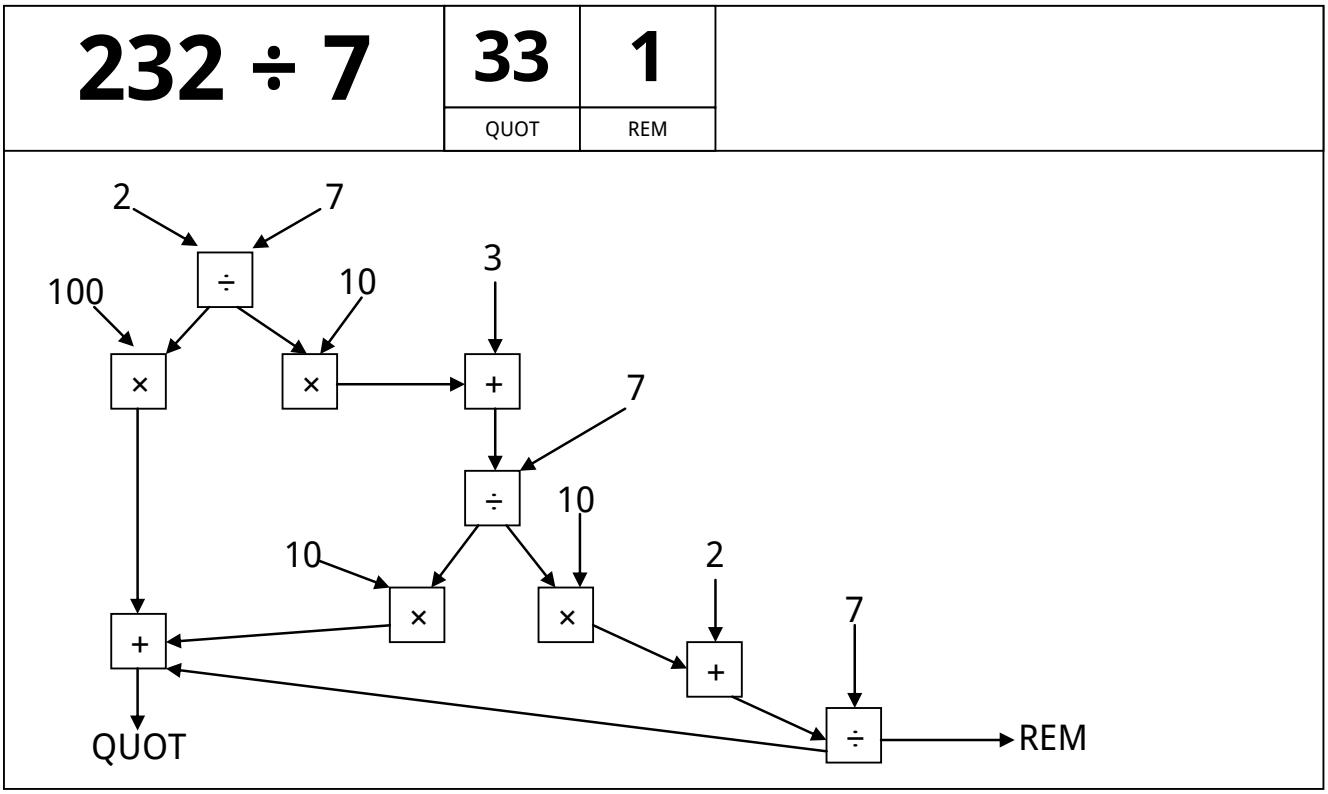
The following main learning tasks will have to be undertaken before the project can be started:

- Learn Clojure
- Become familiar with JavaFX 2
- Spend time designing the GUI and language

Starting Point

I have spent some months learning Clojure, and continue to do so. I have a good working knowledge of Java and experience teaching Mathematics and IT in Years 4 to 6. The first two years of the undergraduate course have familiarised me with Java and its libraries, and thanks to Clojure's interoperability I will be able to leverage these skills for this dissertation. My experience in schools has allowed me to develop the idea for this dissertation, and has given me an insight into what resources are useful in the classroom.

Below is a mockup of what I aim for Nico to look like. Notice how the method is expressed in the form of a flowchart, with outputs (in this case two) for the answer. Arrows show the direction of input and output, and the QUOT and REM boxes show the result of evaluating the functions being passed to them. Ideally, the question "232 ÷ 7" would also change to reflect how the student breaks down the question.



The above solution would auto-generate the following abstract syntax tree represented in Clojure code:–

QUOT is the output of:–

```
(+
  (*
    100
    (:quot
      (div
        2
        7)))
  (*
    10
    (:quot
      (div
        (+
          (*
            (:rem
              (div
                2
                7))
            10)
          3)
        7)))
  (:quot
    (div
      (+
        (*
          (:rem
            (div
              (+
                (*
                  (:quot
                    (div
                      2
                      7))
                  10)
                3)
              7))
            10)
          2)
        7)))
```

REM is the output of:–

```
(:rem
  (div
    (+
      (*
        (:rem
          (div
            (+
              (*
                (:rem
                  (div
                    2
                    7))
                10)
              3)
            7))
          10)
        2)
      7))
```

This assumes that we have a function `div` that takes two arguments `x` and `y` and returns an associative map `{:quot q :rem r}` such that `q` is the quotient of $x \div y$ and `r` is the remainder. Such a function will be included in the basic functions available to the user. Other functions of use would be addition, multiplication, subtraction, exponentiation, function definition and commenting (i.e. labels that are not evaluated), with options available in the question syntax (e.g. `:inhibit+ true`) to restrict arguments to a value of less than or equal to 10 (useful, for example, in questions on long multiplication, to prevent the student from simply giving `(* a b)` as the answer to $a \times b$). Hence a possible means of representing the question above could be:–

```
{:title “232 ÷ 7”
 :topic “arithmetic”
 :answer {:quot 33
          :rem 1}
 :inhibit+ false
 :inhibit- false
 :inhibit* false
 :inhibitdiv true}
```

Resources

This project requires little file space so my Toshiba PC’s disk should be sufficient. I plan to use the same PC as well as my Samsung PC to work on the project, and to back my files up to the PWF, the SRCF and GitHub. I will be using Git for version control.

Work Plan

Planned starting date is 27/10/2011.

October 2011

27/10/2011 - 10/11/2011
Work begins. Start covering the problems outlined in *Difficulties to Overcome*. Design the look and feel of the language and application.

November 2011

10/11/2011 - 24/11/2011
Design the question syntax. Implement the question interpreter. Implement the tree evaluator.

24/11/2011 - 08/12/2011
Implement the hints system. Begin work on the GUI.

December 2011

08/12/2011 - 22/12/2011
Finish the non-language section of the GUI. Begin implementing the graphical language.

29/12/2011 - 12/01/2012
Finish implementing the graphical language and its interpreter.

January 2012

12/01/2012 - 26/01/2012
Finish coding the core project. Begin extension work and evaluation.

26/01/2012 - 09/02/2012
Progress report written to be handed in by 03/02/2012. Preparation for presentation on 09/02/2012.

February 2012

09/02/2012 - 23/02/2012
Finish evaluation. Begin drafting the dissertation.

23/02/2012 - 08/03/2012
Finish extension work. Continue drafting the dissertation and evaluate extension work.

March 2012

08/03/2012 - 22/03/2012
Submit first draft of dissertation to supervisors by 16/03/2012. Begin redrafting on receipt of feedback.

22/03/2012 - 05/04/2012
Continuing redraft and resubmission of dissertation.

April 2012

05/04/2012 - 19/04/2012
Continuing redraft and resubmission of dissertation.

19/04/2012 - 03/05/2012
Dissertation complete 01/05/2012.

May 2012

03/05/2012 - 18/05/2012

Dissertation complete. Final edits, corrections. Binding and submission.