Neil Satra

Sketching Charts

Computer Science Tripos, Part II

Pembroke College

May 15, 2014

Proforma

Name:	Neil Satra
College:	Pembroke College
Project Title:	Sketching Charts
Examination:	Computer Science Tripos, Part II, 2014
Word Count:	11510
Project Originator:	Alan Blackwell (afb21), Neil Satra (ns532)
Supervisor:	Alistair Stead (ags46)

Original Aims

This project aimed to enable users to visualise information faster, and encourage them to explore and experiment with it more. This would be done by building a tablet application that draws on the familiar metaphor of sketching using pen and paper. Users must be able to make a rough sketch of the type of chart they'd like to see. An ink stroke recogniser must then complete the chart with user data. Such a tool should provide the benefits of direct manipulation and liveness, since user actions receive immediate feedback. These benefits need to be evaluated through a user study.

Work Completed

I have successfully designed and developed a charting application that meets all the functional and usability requirements, and includes three extensions. The sketch recognition component uses multiple data mining algorithms to recognise chart elements drawn on a tablet with a stylus. The custom charting widget plots user data into the chart type the user sketched. The interface was designed to be easy to learn - this was evaluated through a user study. The charting widget is ready to be open sourced and included in other ap-

plications. In the process, I also contributed two bug fixes to tools used in the project.

Special Difficulties

- Acquiring and fixing the source code of a component used for sketch recognition from the team that wrote it.
- Automating the testing of the highly visual parts of the project.

Declaration of Originality

I, Neil Satra of Pembroke 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

Contents

Chapter 1

Introduction

The aim of this project has been to design and develop Sketchography, an application for users to make a chart of their data by sketching a rough version onto a tablet screen, rather than using a complex Graphical User Interface or programming. The goal is to make an application that's easier for users to pick up than existing charting tools, and one that encourages exploration of the data and various visualisations. I have successfully met all the core requirements, implemented three extensions, and carried out an evaluation that provided evidence that the design goals were reached.

This chapter documents the motivation for such a project, including previous work in the field, and an overview of what functions the application performs from a user's point of view.

1.1 Motivation

As a society, we are generating increasing amounts of data, and need better ways to quickly visualise them through charts. This project is an exploration of Human Computer Interface concepts governing the interactions of users with tools that let them explore and visualise data.

The design of most charting tools is driven by the choice of interface: the mouse and keyboard. Thus, they usually allow graph generation through one of two means:

1. Configuring a chart template through a number of wizards and dialogue boxes.

2. Manually writing code to generate chart graphics.

Users need to familiarise themselves with how the choices in the wizard, or the commands in the language, translate to graphics.

A better solution is to use a metaphor to a system users have already learnt to use - drawing using pen and paper. Such a system would benefit from matching the users' mental model.

Additionally, there is a long lag between the users expressing their intention in current systems, and seeing the results of their changes after they close the configuration dialogue or compile and re-run the code. This discourages experimentation and exploration. A better solution would exhibit 'liveness' by immediately accommodating users' changes.

1.2 Background and Related Work

Sketching inputs have been studied since the 1960s (Sutherland, 1964) as more natural interfaces to computers for graphics-related tasks, compared to indirections like the mouse and keyboard. This has largely been motivated by the widely recognised importance of interactiveness to Information Visualisation (InfoVis) (Lee et al., 2012).

Meanwhile, there has been increasing adoption of touch-enabled phones and multi-touch slates amongst the general public, demonstrating people's predilection for what have been referred to as Natural User Interfaces (Lee et al., 2012). The wide availability of these tools also makes a sketch-based application more feasible to use.

Additionally, Norman and Draper (1986) describe the 'Gulf of Execution', or the gap between a person's intent and their ability to execute that intent. Existing charting tools require users to learn how to accomplish each task, whereas the familiar metaphor of sketching on paper can encourage exploratory work due to the ease of creating changes by visually expressing what sort of change one is trying to make.

There have been a couple of projects that attempt to apply sketch input to the problem domain of creating charts. Microsoft Research's SketchInsight (Walny et al., 2012) lets users make gestures to indicate a type of chart they want. However, by letting the user simply draw the chart they are imagining, rather than making them learn gestures that may not bear a visual resemblance to the end product, Sketchography aims to be easier to learn. Additionally, gestures are immediately converted to charts, which means any further interaction or modifications do not make use of the stylus' digital ink.

Chao et al. (2010) propose a system that lets the user compose basic elements to make visualisations of arbitrary complexity. However, this requires a lot of sketching even for the most basic charts that are commonly used.

In addition to an improved design, this project is also using better sketch recognition techniques, thus reducing the likelihood of frustration due to misclassification of ink strokes.

A larger body of related work is discussed in section 3.1, in the context of design decisions made during the implementation of this project.

1.3 Project Description

This paper describes an application that allows the user to sketch a subset of a chart on their computer touch screen as they would on paper. The hypotheses are that, when compared to other charting applications,

H1 This interface is more 'learnable' over time

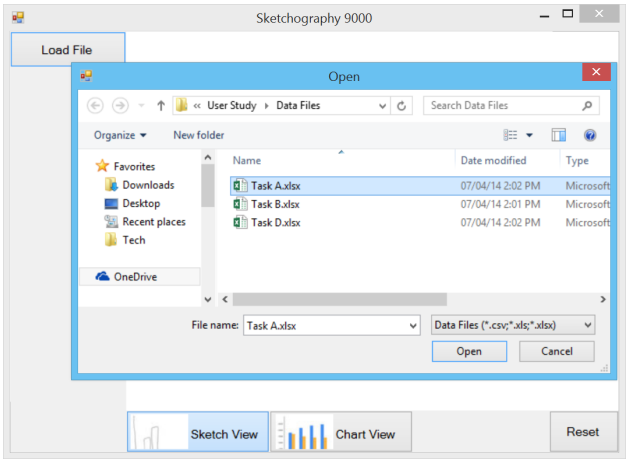
H2 It encourages exploratory data visualisation creation by making modification easier

H3 Its advantages hold independent of how fluent the user is in English

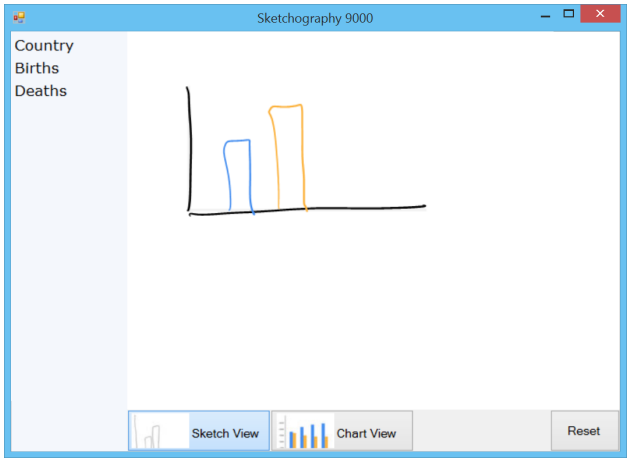
These hypotheses were investigated through a user study, which provided data that strongly supported the first two, and wasn't conclusive on the last one.

The end result is a charting application that works as below:

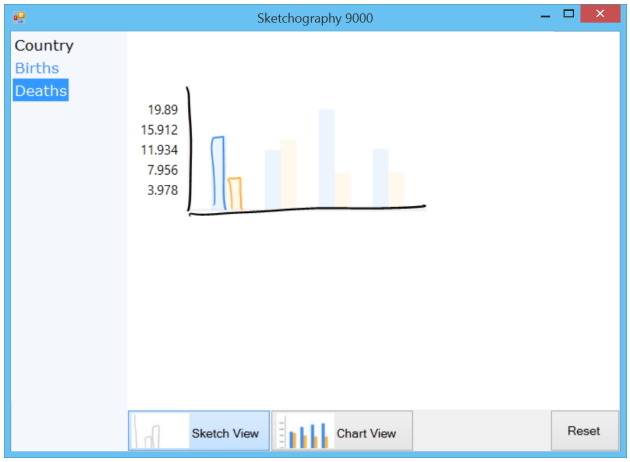
1. The user imports data from a Microsoft Excel file.



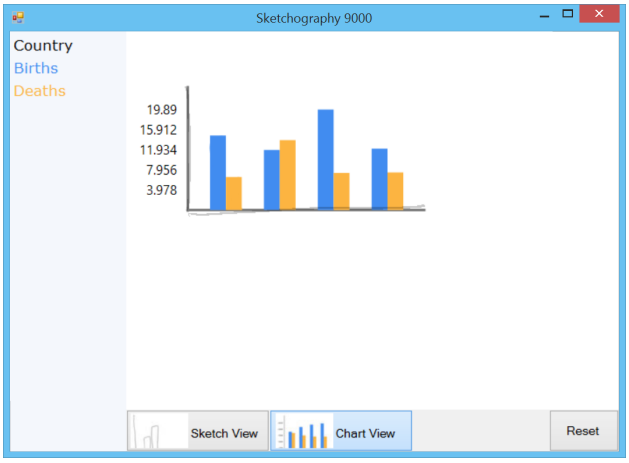
2. They sketch a rough indication of part of a chart.



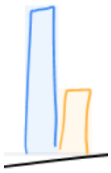
3. They drag the data onto elements of the chart to actually bind the data to the chart.



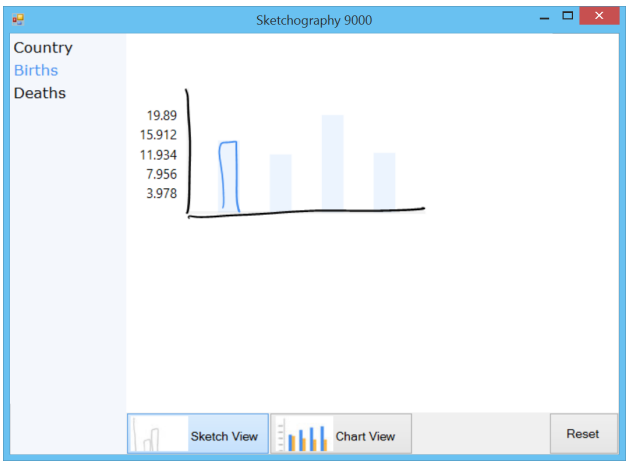
4. The tool then creates a completed, 'formal' chart with the user's data.



5. The tool transforms the user's original sketch to more closely match the formal chart, making the mapping between sketch and formal chart elements evident to the user.



6. Any changes on either the sketch or formal chart is fed through to the other view. For example, erasing the a sketched bar removes a data series from the formal bar.



1.4 Summary

Existing charting tools are very capable and full-featured, but aren't as easy to learn as they should be, in order to encourage users to explore the data. A new application was proposed that should be easy to learn, and make modifications of the data easy. It uses well-performing data mining algorithms to perform ink stroke recognition, and a charting component to display the data visually. A user study was carried out to confirm that it achieved its goals.

Chapter 2

Preparation

In order to keep the project organised and on schedule, some preparatory work was required. Specifically, the requirements needed to be analysed, and the goals and work items needed to be spelt out accordingly. This enabled me to use professional software engineering practices, including test driven, iterative development and version control. I had to read about various design principles and how they could be applied to maximise the usability of this application.

Additionally, the feasibility of performing the stroke recognition needed to be evaluated, by building a classifier that exhibited reasonable accuracy. The various approaches to sketch recognition undertaken in the past had to be researched. Once I chose the most appropriate approach, the data mining algorithms which it uses had to be understood.

2.1 Requirements

This project could have taken numerous different design directions. Since the functionality and its benefits over existing systems depended heavily on the design chosen, it was difficult to state technical requirements without first designing the system.

However, in order to focus the design exploration and plan realistic development schedules, a requirement analysis was necessary. Based on the project proposal, the following functional goals were identified. They focused on what basic tasks the system must help the user achieve, while allowing leeway in how the program exposed and implemented these features.

The system must allow the user to:

Core 1 Visualise data they have stored in common file formats.

Core 2 Specify the type of chart they want by drawing a likeness of it on screen using a stylus.

Core 3 Bind the data to the chart using an interface that makes it clear how the data is being used to generate the graphics.

Core 4 Specify visual properties of the chart, such as size and position, through the sketches.

Core 5 Manipulate the visual appearance of the created chart.

In addition, time permitting, the system may:

Extension 1 Transform user-drawn sketches to resemble and overlap the formal chart elements they generated, in order to show the link between the sketch and the formal elements visually.

Extension 2 Mirror any manipulations applied to the formal layer back to the user drawn sketches, in order to keep the visuals of the formal and sketch layers in synchronisation.

Extension 3 Allow users to undo actions by erasing sketches, and remove the corresponding formal elements without throwing errors.

Extension 4 Allow users to manipulate any property of chart elements, not just one, so that the domain of visualisations they can create is infinite. For example, allow them to bind not just the height of bars in a bar chart to data, but also their width and colour.

Extension 5 Analyse the data and infer properties that may allow it to automatically suggest properties of the chart, such as which field belongs on which axis, or whether a data series should be log scale or linear scale.

Extension 6 Support exporting the chart as a Microsoft Chart object that can be embedded as a dynamic object in Microsoft Office files, not just as a raster image.

The core of this project focusses on making more usable software, rather than providing additional functionality, compared to existing systems. Hence, some usability goals were also specified:

Usability 1 Users must be able to create charts at least as quickly as they can using current charting systems.

Usability 2 Users must be able to build a mental model of the software's behaviour within 2 uses of it. They should thus be able to accurately predict the consequences of any action taken within the software.

Usability 3 Changes to the visualisation must occur through directly manipulating the visual representation of the chart, rather than through disconnected User Interface widgets.

Usability 4 The user must be able to easily try out changes to the visualisation, see what the resultant chart would look like, and undo them if needed.

2.2 Design Goals

In order to meet the usability requirements, some design principles need to be followed. The main benefit of a sketch-based interface is that it is more likely to match the user's mental model, or their expectation of what every UI widget will do, since it draws upon familiar metaphors, and offers liveness and direct manipulation of the visualisation. This section explores the theories of direct manipulation and liveness, and how they might be applied to this project.

Direct Manipulation

> "Direct manipulation systems offer the satisfying experience of operating on visible objects. The computer becomes transparent, and users can concentrate on their tasks. [It] permits novice users access to powerful facilities without the burden of learning to use a complex syntax and lengthy list of commands. Direct manipulation involves three interrelated techniques:
>
> 1. Provide a physically direct way of moving a cursor or manipulating the objects of interest.

2. Present a concrete visual representation of the objects of interest and immediately change the view to reflect operations.
3. Avoid using a command language and depend on operations applied to the cognitive model which is shown on the display."

(Shneiderman, 1983)

The mouse and keyboard already provided more direct manipulation than the previous command line interfaces; stylus-based and touch interactions are the next step. A stylus provides a more direct way of moving the cursor and manipulating objects of interest than a mouse since it interacts directly with objects on screen. Besides the inherent benefits of stylus interaction, this application provides a concrete visual representation of the objects of interest, since all configuration settings for the chart are represented as visual parts of the chart rather than options in a configuration window far removed from the chart itself. Additionally, changes are made directly to the chart, resulting in instantaneous feedback, rather than being made to a settings window which then updates the chart when closed.

Live Programming

Blackwell (2002) suggests that classical definitions of programming need to be re-thought based on the evolving use of computers. Specifically, it is suggested that:

1. All computer users must now be regarded as programmers.
2. All software applications now include programming languages.
3. These tools only differ in their usability for the purpose of programming.

Ko et al. (2011) clarify by calling it end-user programming - "Most programs today are not written by professional software developers, but by people with expertise in other domains working towards goals for which they need computational support... Although these end-user programmers may not have the same goals as professional developers, they do face many of the same software design challenges..."

Since one of the alternatives to this project is to actually use programming frameworks like D3.js to generate charts, the task of instructing the computer on how to turn data into graphics is rightly considered programming. One of the properties that improves the usability of such a programming tool is liveness.

> "Liveness in programming environments generally refers to the ability to modify a running program. Liveness is one form of a more general class of behaviors by a programming environment that provide information to programmers about what they are constructing."
>
> (Tanimoto, 2013)

This application allows you to directly draw or erase lines in the chart, for example, rather than making changes to textual code and then re-running the code to verify that it does create the change you intended. Thus, it provides a live programming environment, whose benefits have been previously documented (Tanimoto, 2013).

2.3 Work Items

The following broad work items were identified as necessary to achieve the objectives above:

1. Get the requisite approvals for the human study from the Ethics Review Committee.

2. Assess the various methods to build a classifier for ink recognition.

3. Run an initial user study to see how people naturally draw graphs, and also use it to collect training examples for the classifier.

4. Build a UI that accepts strokes, runs them through the classifier, and shows the user feedback to indicate successful recognition.

5. Build the UI widget that lets users import their spreadsheets in Microsoft Excel (xlsx) or Comma Separated Value (csv) files. It must then expose the various fields detected.

6. Build the charting component to convert the recognised sketches and the ink into a finished visualisation.

7. Run a pilot study followed by a user study to evaluate the system.

An iterative development process, similar to the Spiral Model and Scrum, was adopted for this project. This allowed early experimentation and user testing of the various components and different interface designs. After each development iteration, design choices arose. Of the various options evaluated, one would be built. This would start a new iteration, and the cycle would repeat.

To manage technical debt and minimise bugs, Test-Driven Development (TDD) was adopted. In its pure form, TDD relies on a fully planned out functional specification and automated testing. However, since the design and functionality of this project were evolving over time, small specifications were made for each iteration, and changed as necessary after a sprint (to borrow Scrum terminology). Also, since the input needed to be sketched by hand, and the output was visual and changed in literal pixel values between trials without changing in semantic meaning, it was non-trivial to simulate and verify this interaction automatically. These parts were verified through repeated manual testing each time the code was changed or refactored. Parts of the code that were effectively in a functional programming style (deterministic calculations with no side effects) could, however, be checked with unit tests. In addition, to pick up design flaws early, the prototypes were constantly hallway tested (friends and family were asked to use the application and speak out what they were thinking as they did so).

Thanks to the constant refactoring of the code, the coupling between various components was minimised. This meant the project could be open sourced at the end and plugged in as a standard GUI component in other Windows applications, making it easier to integrate this component in real world applications.

2.4 Development Environment

For this project, the hardware available was a Microsoft Surface Pro (1st gen), which features the active digitizer screen and stylus required for precise inking. Since this machine runs Windows by default, I chose to develop the

system using the .NET framework, which has built-in support for Tablet PCs and Ink handling.

As a precaution, insurance was taken out on the machine to ensure quick replacement in case of damage or loss. Additionally, version control was used extensively in the project, to ensure no work was lost. The code for the RATA ink stroke recogniser, described below, was uploaded to a Git repository in collaboration with RATA's authors. The code for this project was then written in a fork of that repository, to allow updates to RATA to be pulled in. The dissertation itself was written in LaTeX, so that the text files could be versioned in another Git repository. Both repositories were backed up off-site on repository host Bitbucket. The dissertation was also backed up online using file synchronisation software Microsoft OneDrive.

2.5 Building the classifier

Core to the system is an ink recognition component that identifies the chart element (e.g. bar, axis) that the user has sketched. This must work above a certain accuracy threshold or the system will prove frustrating to users (Frankish et al., 1995). However, since the project scope included other components too, the time available to build this classifier was limited. Additionally, given the complexity of building a classifier, and my limited experience with building them, it would be difficult to get the same accuracy as those built using mature libraries. Hence, we decided to build a classifier using existing tools rather than implementing one from scratch.

2.5.1 Recognition Method

A number of different approaches have been taken in building systems that automatically interpret hand-drawn sketches. These approaches vary in the recognition accuracy they offer, and the robustness to generalise across multiple domains. Some of the approaches that have been attempted:

- Focus on **defining shapes structurally**. A base vocabulary of primitives like lines, ellipses and arcs is built by describing the properties of such shapes. Shilman et al. (2002) used a hand-coded grammar to describe shapes in a domain as a composition of such primitives. Alvarado and Davis (2004) used dynamically constructed Bayesian networks to

scale this process to multiple domains. Hammond and Davis (2006) developed a language to manually describe how diagrams in a domain are drawn, displayed and edited.

- Look at the **visual appearance** of shapes and symbols. Kara (2004) and Ouyang and Davis (2009) used image-based similarity metrics to perform template matching. Shilman et al. (2004) broke up the ink into connected subgraphs of nearby strokes, which were then compared to known symbols. Oltmans (2007) proposed a visual parts-based method that utilise a library of shape contexts to describe and identify symbols in a domain.

- **Compute features** of the ink. Patel et al. (2007) select these features and sets their thresholds statistically. Yu and Cai (2003) use heuristics for the same purpose. Chang et al. (2010); Rubine (1991); Willems et al. (2009) all use machine learning to automatically find relationships between features and choose an appropriate feature set accordingly.

Chang et al. (2010) showed that the first two approaches forego some accuracy, since they rely on the final pixel values of the sketch and so do not fully exploit the rich temporal data stored with digital ink. Within the last approach, they showed that using machine learning allows recognisers to be generated for new domains with less effort than statistical or heuristic methods. Since the domain for this charting application could grow over time, we wanted a recogniser that could be adapted easily over time. Using data mining, this can be done using training data rather than programming effort.

Having decided on using a feature-based approach that selects the feature set by data mining a training dataset, there were a number of alternatives available to us. While most projects, like (Rubine, 1991) and (Willems et al., 2009), rely on one or two data mining algorithms, (Chang et al., 2010)'s RATA.SSR combines the results from four well-performing algorithms in WEKA (Hall et al., 2009) tuned to their best configurations, to provide a more accurate recogniser. RATA.SSR thus outperformed all the other recognisers tested (PaleoSketch (Paulson and Hammond, 2008), CALI (Fonseca et al., 2002), $1 recogniser (Wobbrock et al., 2007)) on domains other than the one they were explicitly demonstrated on.

RATA.SSR also has the advantage of outputting an API that can be plugged into our system after we build the recogniser. Additionally, one of

the authors, Beryl Plimmer, previously worked with some members of the Graphics and Interaction Group at the Cambridge Computer Laboratory, and so could be reached for support and source code, which proved to be an invaluable resource.

2.5.2 Data collection

After acquiring RATA, I inspected the code and did manual testing, which revealed some blocking bugs. Since we were in contact with the authors of the software, I was able to confirm with them that these were indeed bugs. I implemented fixes for them and contributed them back to the authors, and am working towards getting the code ready for to be published Open Source.

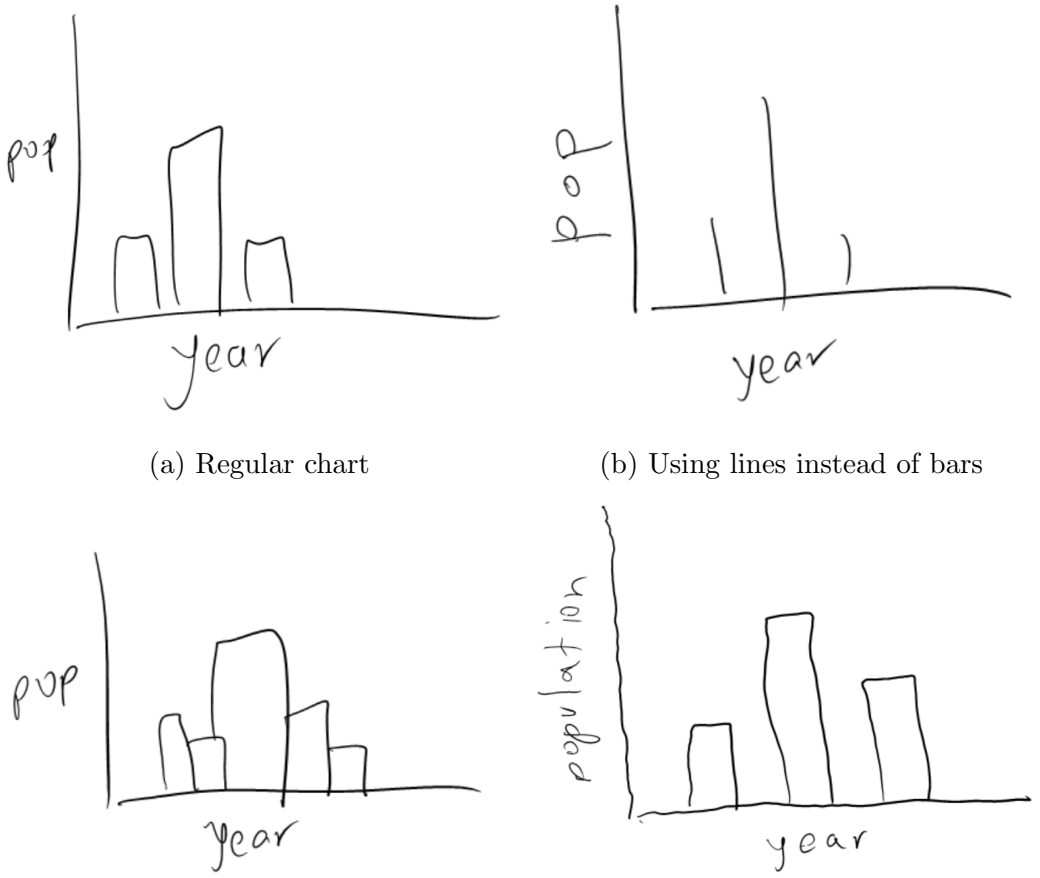
With a working version of RATA, an initial study was run to collect training data. We asked 10 participants (20-26 year old Cambridge students, studying a large variety of subjects) to draw a chart. I spoke out the following prompt:

> Imagine you are a government official trying to use a bar chart to visualise how the population has grown over time. Can you sketch out what this bar chart might look like? Just treat this screen like paper.

They were then shown a simple UI which contained a large white canvas and the following task description written in a panel:

> Draw 2 axes. Label the x axis 'Year' and the y 'Population'. Draw 3 bars of different heights. Each shape (axis, bar) should be drawn in one stroke.

They were asked to draw the same chart 2 times, in order to get 20 training samples in total, and to observe how much variation there is between multiple sketches by the same user. On the second drawing, they were encouraged to draw a less conventional chart, to make the system as robust in the face of variations as possible.



(a) Regular chart

(b) Using lines instead of bars

(c) Grouped bars

(d) Using a mouse instead of the stylus

Figure 2.1: Some of the more unusual chart sketches collected

Three elements were then defined: Axis, Bar and Text (an extra element, 'L Axis' was added later based on feedback from a pilot user study). We went through each figure and labelled the various elements.

Figure 2.2: Elements of the sketch labelled as Axis (cyan), Bar (brown) or Text (dark blue)

Once all the elements in all the charts were labelled, we could begin training. RATA includes a dataset generator tool that allowed easy extraction of various features of the strokes, such as 'distance from first to last point', 'absolute curve of largest segment' and 'pressure variation'. Data for 121 such attributes, about 270 ink strokes was compiled into a .csv file for use in training.

2.5.3 Training

The labelled data was then sent to a 'Vote' classifier in Weka, which combines the probability distributions derived from multiple classifiers (Kuncheva, 2004). Specifically, the types of classifiers combined were Logit Boost, Bayes Net, LMT (Logistic Model Trees) and Random Forest.

In order to assess how well each of these individual classifiers were performing, an experiment was set up using Weka Experimenter. The data collected in the initial study was shuffled, and then 66% was chosen randomly as training data, the rest as testing. A paired T test on this data showed that none of these top-performing algorithms had statistically significant improvements in accuracy over the others.

Tester: Paired Corrected T Tester
Analysing: Percent correct
Confidence: 0.05 (two tailed)

Algorithm	**Accuracy**
LogitBoost	97.05
BayesNet	98.15
LMT	98.80
RandomForest	96.07

Table 2.1: Comparison of accuracy of the algorithms used

2.6 Summary

This chapter detailed the work done before implementation was begun. A thorough requirements analysis covered functional goals (core and extension) as well as usability goals, while being open-ended about how exactly the system would look and operate. The work items required to meet these requirements were assessed. In order to assess the feasibility of building such a system, an initial study with 10 participants was run. The data thus generated was manually labelled and used to generate and train a classifier. Finally, the algorithms used within the classifier were individually checked for their accuracy on this domain of chart element identification, on which they all performed very well, getting accuracies of upwards of 96.07 despite the small training set.

Chapter 3

Implementation

With the classifier working, a major milestone was reached as the project was deemed viable to complete. To build the application, the interface needed to be designed, and the underlying functionality needed to be developed. While the design and the development were highly interdependent and hence interleaved in the iterative development process, they have been described separately below for clarity. Constant hallway testing was carried out to minimise the probability of bugs or usability issues going uncaught.

3.1 Design

I identified the following 6 design decisions at the start of the implementation phase.

1. The application must have a large canvas to sketch on.
2. The chart must be shown somewhere - either on the same canvas as the sketches, or in a separate viewing panel. The limited screen size on the Surface Pro makes the latter a less desirable option.
3. There must be a panel where data can be imported, and the imported column headers can be viewed.
4. The user must be able to drag and drop this data onto where it needed to be bound, as this would be direct manipulation.
5. A status area may be needed to inform the user about what state the program is in.

6. There must be a 'Reset' button for users to start over if needed.

These specifications were enough to start building a high level prototype, but the final interface would need much more thought and research about the details of the design. A number of tough choices were faced, whose tradeoffs are outlined in the remainder of this section.

3.1.1 Sketches or gestures

So far, I have talked only about sketches, but in fact, there are two ways one can use a stylus to send input to a computer: sketches, or gestures. Sketches share some visual semblance with the chart they're trying to indicate, whereas gestures are just movements chosen from a library of easy-to-distinguish movements, that behave like commands instructing the software to choose a certain chart type. SketchInsight (Walny et al., 2012) uses the latter approach, requiring, for example, just an 'L' shape to indicate a bar chart, or a 'V' shape to indicate a line chart.

Advantages of Gestures

- Gestures provide higher recognition accuracy, since the domain of shapes to distinguish between is limited, and can be designed to maximise differences between them.

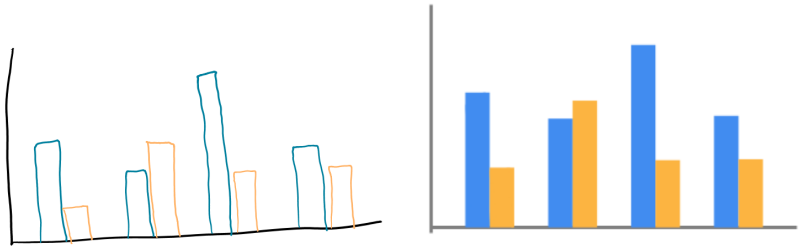
Advantages of Sketches

- A gesture is transient, unlike a sketch. Users wouldn't be able to modify their input, they would have to redo their actions.
- One of the principles of direct manipulation is to avoid using a command language, but a gesture library is exactly that, requiring users to remember which gesture corresponds to which chart type or element.

To satisfy the design goal of direct manipulation, the sketch-based approach was chosen.

3.1.2 'Sketchy' or Formal Appearance

Once the user sketches, say, one bar, the application needs to complete the chart by generating more bars for the rest of the data points. It could do this in one of two styles: 'sketchy' or formal.



(a) Sketchy look (b) Formal look

A 'sketchy' look encourages exploration and modification. This is due to the usability dimension described by Bresciani et al. (2008) as 'Perceived Finishedness'. Yeung et al. (2008) showed that a doodle-like appearance of low-fidelity prototypes encourages early-stage experimentation and discussion, whereas a formal appearance looks finished and professional. Blackwell et al. (2008) note how the rapid replacement of pen and paper by computers has made more of our processes formal and precise, potentially discouraging us from exploration and iteration in creative processes that would benefit from them. Thus, generating the rest of the bars with the same look and feel as the one sketched by the user has clear benefits in terms of reflecting malleability.

However, automatically generating more bars with a sketchy look can be non-trivial (Plimmer et al., 2010). Wang et al. (2011) proposed an approach that uses a library of hand-drawn examples to generate shapes that look human drawn. However, besides being laborious, this would be inappropriate in this case since one bar would be drawn in the user's style, while the others would distinctly be in the style of another author.

On the other hand, Walny et al. (2012) proposed SketchInsight, which simply discards the user's strokes, and replaces them with a completely formal chart. This finished look, while discouraging modification, exhibits what Blackwell et al. (2008) describes as a 'Formal Connotation', which is expected in many professional settings. Users may be more inclined to present a formal-looking chart to colleagues than a sketched one.

Thus, I designed an interface that offered the benefits of both looks. The user's input is stored in sketch view, and the user can keep modifying it. The computer generated chart is in formal view, available at any time to preview or present.

3.1.3 Modes or modeless

Since the application now has both sketch and formal content, the user must have an easy way to switch between the two views. One approach is to give the user an explicit UI widget to toggle between the two modes. This way, the user explicitly indicates what they want to see, and thus should have a better understanding of what state the system is in. This also allows the system a chance to change the controls available to the user.

The other approach is to avoid modes, requiring lesser cognitive effort from the user since they don't need to keep track of what state the system is in. Norman and Draper (1986) describe this as the Gulf of Evaluation - the difficulty of assessing the state of the system, and the Gulf of Execution - the difficulty of knowing what actions are possible in the current state.

In this project, this could have been done by showing the sketch view when the user wants to edit the chart, and automatically showing the formal preview when they stop to look at it. Their intent can be predicted because the active digitizer hardware can detect when the stylus is hovering over the screen, before it actually touches the screen. The application can then replace the formal view with the sketch view just in time as the user's stylus touches the screen, and switch back to formal view when the cursor is out of range. Hallway testing revealed users were delighted by this feature, since most touch systems cannot give feedback to hover events.

At first, the modeless version was chosen for its lower cognitive overhead. However, when the extension to allow edits not just to the sketch view, but also the formal view, was undertaken, we had to switch over to a mode-based system to allow the user to interact with the graph in both views with their stylus.

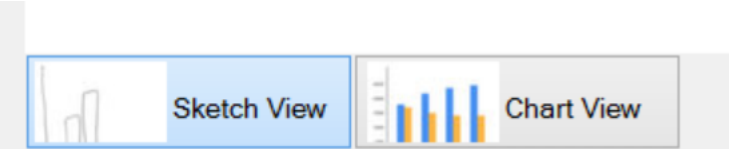


Figure 3.2: Users can click visual toggle buttons to switch between views

3.1.4 Stock or bespoke charting widget

Since the system is generating a formal version of the chart, a charting component is required to render this visualisation. The .NET framework comes

with built-in chart controls that offer basic functionality with relatively low implementation effort. They also allow easy export as dynamic chart objects into Microsoft Office files. However, customising their appearance beyond a certain point is extremely difficult, making it easier to just make one's own charting component from scratch and control all aspects of the rendering. This means having to re-implement a lot of core functionality though, such as scaling shapes correctly, choosing labels that are round figures when possible, and generating colours than work well together for different data series. This also means that the chart can only be exported as a raster image rather than as a chart object.

To enable rapid prototyping, we chose to utilise the standard charting component at first. As our needs to customise the chart grew, we were able to make our own chart class that implemented the same interface as the standard component, and so could be slotted in to replace it.

3.1.5 Finite or infinite domain

Some tools, such as Microsoft Excel, let the user make one of a limited set of charts, such as bar or pie charts, quickly. Others, (which usually involve coding), such as D3.js, let the user make a vast variety of visualisations by creatively combining basic elements like lines, boxes and wedges. However, these require expert knowledge of the tools, and take longer to create basic visualisations.

	Library of charts	**Modular charts**
Description	Draw basic gestures or elements to indicate which chart type is desired	Draw any one of 7 basic components and bind data to attributes of theirs such as width, height, colour or radius
Domain	Finite domain	Infinite domain
Speed	Quick	Slower
Complexity	Simple interface, just drop data on an element to bind data	Complex interface to expose all attributes and manage data binding

Chao et al. (2010) uses pen gestures to indicate 'proto-objects' that can be combined. While an infinite domain system that uses pen sketches would have been intriguing to explore, it would contradict the project's primary usability goal that the system should be faster than users' current systems. Additionally, the 80/20 rule indicated that while a few power users may want to generate custom visualisations, the majority would just want to make simple charts. Thus, the additional functionality did not justify the additional complexity for the average user.

3.2 Development

Interleaved with the design process above was the development process detailed in this section. The C# code supports a stand-alone Sketch Chart component that can be used in any Windows Forms application that requires charting functionality, as well as a reference implementation of such an application, Sketchography, which lets users import their Microsoft Excel or Comma Separated Values data and generate charts. This satisfies all the core requirements described in section 2.1, as well as 3 of the 6 extensions outlined.

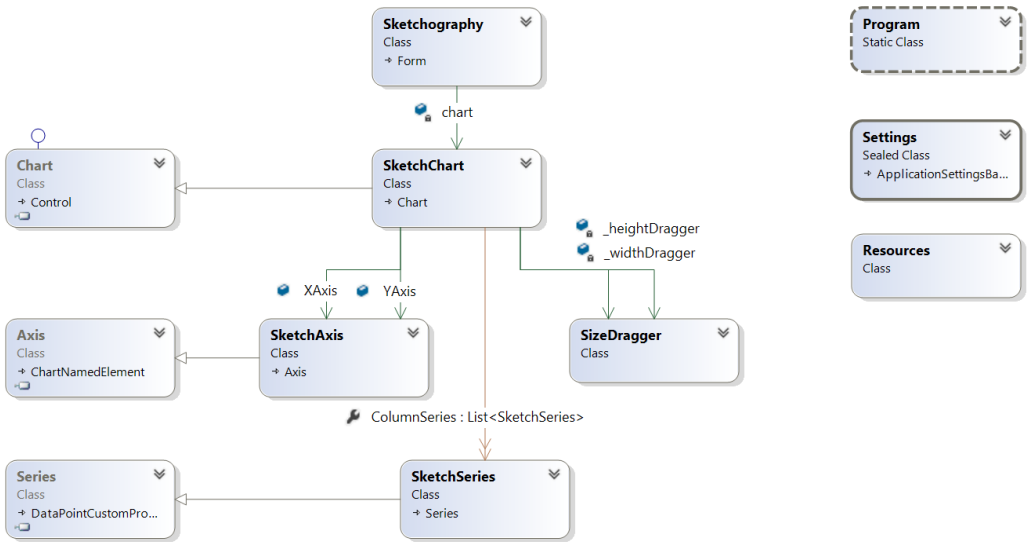


Figure 3.3: Interactions between classes

As seen above, the `Sketchography` form is the main application run when `Program` starts. `Sketchography` contains a `SketchChart` component, which

contains the canvas for users to sketch on, as well as the formal chart generated. `SketchChart` in turn uses `SketchAxis` and `SketchSeries` objects. `SketchChart`, `SketchAxis` and `SketchSeries` inherit from their non-sketch counterparts `Chart`, `Axis` and `Series` (which are a part of the .NET framework), in order to reuse the built-in functionality and members, and complement them with functions and members specific to sketching. `SketchChart` also references two instances of `SizeDragger`, which is a custom UI widget to support dragging ends of columns in a column chart to scale them. Settings and Resources are two additional classes used to store properties of the project.

No class is more than 400 lines of code, indicating that functionality has been split up at a reasonably fine grain to not concentrate too much responsibility in any one class. Visual Studio, the IDE used for Windows Forms development, has a built-in code metrics tool with a maintainability index.

Index value	Maintainability
20 - 100	Good
10 - 19	Moderate
0 - 9	Low

Table 3.1: Microsoft Visual Studio maintainability rankings (Microsoft Corporation, 2013)

The project got an overall index of 74/100, putting it squarely within the highest maintainability segment.

At a high level, the functionality of the program can be divided into 3 responsibilities - data handling, sketch processing and charting (in increasing order of complexity). It is written in an Object Oriented fashion, with separation between the views (Windows Forms) and controllers (C# classes), to allow for easy testing.

3.2.1 Data import and management

The application is targeted at the average consumer, whose data is most likely to be stored in spreadsheet files. Thus, it is important to allow them to import data from .xlsx and .csv files. For the sake of simplicity, the code

assumes that the data is well-formed. Specifically, it works on the following assumptions:

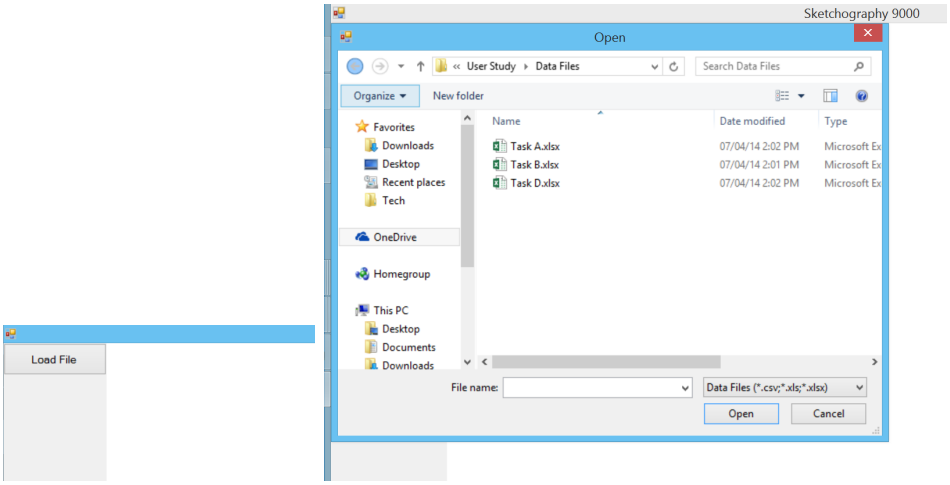
1. The data is arranged as records in the rows of the spreadsheet.
2. The first row contains the names of the various fields.
3. No data is missing (if there are m columns and n rows, there are $m \cdot n$ data values.

Under these assumptions, importing tabular data is a common use case, so I studied a number of existing libraries and methods to do this in C#. At first the LinqToExcel package from the NuGet package manager was used to easily import data from the file. However, this package had limited documentation, making it hard to achieve more complex tasks. Additionally, it added an external dependency, and carried a license that allow free reuse of the software as long as the original copyright message was included. The package was helpful for rapid prototyping at first, but after the code was more mature, we removed this dependency and implemented the file import ourselves using the OLE DB provider.

The interface shows a "Load File" button, which when clicked reveals a standard system filepicker dialog with a filter to show *.csv, *.xls and *.xlsx files. Selecting a file causes the "Load File" button to disappear and be replaced by a list of column headers (the names of the various fields). This helps minimise interface clutter and only expose the most relevant information at all times. However, it comes at the cost of the user needing to hit 'Reset' if they choose the wrong file by mistake. To reassure the user that data has been imported correctly, and for them to check which column header corresponds to which data, hovering over any header reveals the first few entries for that field as a sample.

3.2.2 Sketch Processing Workflow

RATA.SSR provides a sample that calls the classifier API. For this project, that reference was followed, with a lot of extraneous code and unnecessary branches removed, until a basic prototype existed consisting of nothing but a blank canvas which receives user ink, passes it off to a classifier, and outputs the recognition result in a log-like text box. This was then integrated into the application described above, with its data import facility.



(a) Initial state (b) When "Load File" is clicked

(c) Column headers are shown with data previews in tooltips

When the user draws a stroke on the SketchChart element within the Sketchography window, a number of steps occur:

1. The `SketchChart` is covered by an `InkOverlay` object, which receives the stroke. An `inkOverlay.Stroke` event is fired, which allows our event handler to run custom code.

2. The `InkOverlay.Stroke` event handler calculates additional features of the stroke that are not included in the stroke's properties by default, and stores them in its extended properties.

3. The stroke is then sent to the classifier, which is loaded from a file on program initialization.

4. The `classifierClassify` function returns a string from a set of previously defined strings representing the various recognition results. This string is passed off to the `ConvertToFormal` method, which, as the name suggests, creates the corresponding formal chart elements. The

`ConvertToFormal` method is also sent the stroke object itself, so that it can use its properties such as the location to place the formal object accordingly.

After the core of the project was finished, **Extension 3** was implemented that allows erasing of strokes. The stylus available has a button simulating an eraser at the back. When this eraser comes into range, a `CursorInRange` event is thrown, wherein we can check whether the stylus is inverted or not.

```
_inkOverlay.EditingMode = e.Cursor.Inverted ?
                    InkOverlayEditingMode.Delete :
                    InkOverlayEditingMode.Ink;
```

Then, when a stroke is received (the same event is thrown whether the stroke is ink or the eraser), if the editing mode is Ink, the steps above are run. If it is Delete, the corresponding chart element is detected and reset. On the next re-paint of the interface, this element will be removed from the canvas. To minimize lag between the action and the feedback, this repainting is forced immediately with an `Invalidate()` call.

Below is a description of how `ConvertToFormal` responds to user actions it receives, via the stroke classifier.

User Action	Application Response
Load File	Bind chart's data source to the in-memory data table created from the imported file.
Draw an axis	1. Calculate the bounding box (the smallest rectangle that entirely encloses the stroke) of the sketched axis, and use a heuristic comparing the height to the width to determine whether it is a vertical or horizontal axis. 2. The line running through the center of the bounding box is taken as the intended coordinates of the formal axis. This must be done since sketched axes are rarely perfectly straight lines, so using the actual endpoints of the line might result in a non-aligned axis. 3. The endpoint coordinates of the straightened line are used to set the vertical or horizontal position and size of the formal chart. They are also saved in an Axis object, which can then be drawn onto the formal chart. 4. If it a horizontal axis, a drop target is below the axis. This is a rectangle of the same length as the axis, and a preconfigured height deemed big enough to make it hard for the user to miss, while minimising the screen space used up.
Draw a bar	1. A bar is the indicator that the user wants a new data series added on a bar/column chart. Thus, a new data series is added to the chart's SeriesCollection in the form of a Series object. The name of this series is set as "SeriesX", where X is incremented for each new series. This will be useful for the drop target system described later. 2. When the new series is added, the Chart object automatically assigns it a new series colour from the colour palette. The application changes the colour of the stroke the user drew to this series colour, to give feedback that it has been recognised as a series. 3. A drop target the same size and position as the bounding box of the bar is created, corresponding to that series.

In response to strokes that create chart elements, drop targets are generated so that the user may then drag and drop data directly onto the chart element to bind it. This provides direct manipulation, since it does not force the user through a menu and dialogue system to configure which data corresponds to which axis. In order to do this, a dictionary is maintained in SketchChart, storing the mapping between Rectangles encoding the location of a drop target, and the string representing the chart element it corresponds to. For the x axis, this would be "X Axis"; for the data series it would be the name of the series.

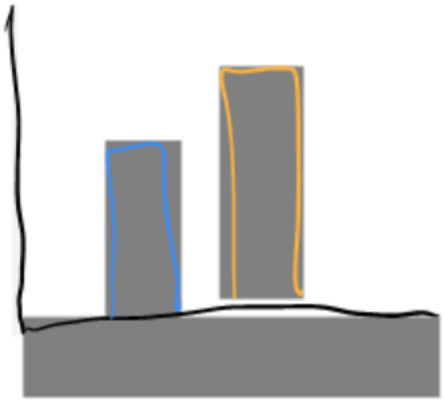
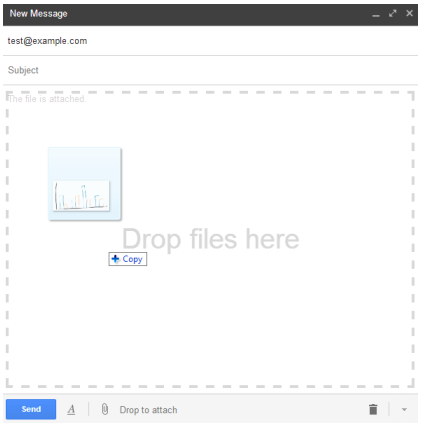
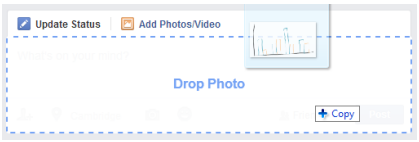


Figure 3.5: Drop Targets for 2 data series and the X axis, revealed when data is being dropped onto the chart

When the user begins dragging a column header from the list on the left, this calls a number of event handlers in Sketchography. These configure the drag and drop action so that the column name saved within the list item is the data sent to the drop location. They also flip a `ShowDropTargets` flag, so that the chart rendering system draws grey rectangles indicating where the drop areas lie. This UI pattern is based on one users may be familiar from a number of drag and drop interfaces they may have previously interacted with, thus making it easier for them to understand what's happening. We also tried adding 'Drop here' text in the targets like in the examples below, but this proved too cluttered in the small drop targets being used. Additionally, hallway testing revealed that once users began dropping, they were already aware that they must drop it in targets, and the change in colour was conspicuous enough to draw their attention to them.



(a) Gmail.com compose window, taken in May, 2014



(b) Facebook.com status update widget, taken in May, 2014

When the data is dropped, this calls another event handler. In Sketchography, the event handler simply verifies that it's a valid drag and drop, and then passes on the data to the `DropData` function on the SketchChart. For every rectangle in the dictionary of drop targets, `DropData` checks if the drop event occured within it. When it finds the drop target the user picked, it looks up the corresponding string, and calls `BindDataSeries` on it. `BindDataSeries` adds the binding appropriately. If the data was dropped on a data series rather than the X axis, it also renames that series and the corresponding drop target, so that the user can reassign that bar to some other data if they change their mind. From this point, the charting functionality takes over.

3.2.3 Charting

This part of the functionality, unexpectedly, proved the most challenging and time-consuming. In the first few iterations, a built-in .NET chart component was used, which allowed rapid prototyping. However, converting from the coordinate system used within the charting component to that used in the rest of the window was proving non-trivial and inaccurate. In addition, the component did not allow very fine-grained positioning of its various elements, or much control over their sizing and other attributes. Hence, we had to implement our own chart control. To make the most of the functionality already implemented in the built-in control, we inherited from it. This also meant our custom control could easily replace the built-in one since it implemented

the same interface but with a few extra features.

The painting of the chart graphics is done in a largely procedural fashion unfortunately, as this is how it is done for Windows Forms controls. Most of the work is done in the `PostPaint` event handler, which is called after the built-in chart is done painting (even though the built-in chart control is not shown to the user at all, it still needs to be painted in a hidden background for some of its functionality to work correctly). This painting occurs whenever changes are made that might affect its output. Within the event handler, there are distinct sections that:

- Draw a background to cover up the built-in chart.
- Draw the formal axes and bars.
- Draw a height drag handle if needed.
- Draw a width drag handle if needed.
- Move the user's strokes to fit the bars.
- Draw the drop targets if needed.
- Change the colour of ink strokes to match their series colour if needed.
- Update the legend.

Each of these steps involves a lot of complicated computation, maintaining of state, and conversion between coordinate systems. The particularly interesting parts are described below.

Positioning and scaling bars

To allow maximum flexibility, constants are introduced that control various parameters of the appearance of the bars, such as the ratio of the gap between them, and the fraction of the height of the chart they should cover by default. Then, formulae are used to account for the number of data series, the relative width scale of each series, the relative scale of the gap between data points, and the length of the X axis, to calculate the position each bar should be drawn at.

The application also has to calculate the values to show for labels on the Y axis. This is something that the custom charting control does not handle

as elegantly as the built-in control, since the values are not usually rounded, whole numbers. This is because the range of values in the data series is simply spread across a set number of labels to determine the individual label values.

Transforming strokes to match formal bars

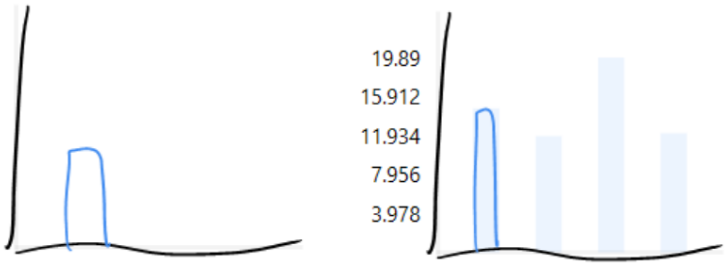


Figure 3.7: Before and after the transform

This implements **Extension 1**, to better express the connection between the sketches and the formal bars to the user. It is done by maintaining a mapping between the sketch and the first bar of the series. When the bar is drawn, its coordinates and dimensions are saved. These are then applied as a scale and translation transform to the associated sketch.

Height and width dragging

Extension 2 was to allow users to make changes to the formal chart, and have those changes feed back to the sketches. We decided to offer the ability to change the width and height of the bars. When in the formal view, if users hover over the first bar in a series, they see a drag handle each for the height and width, in the form of a grey rectangle. We decided on the size of these handles by experimenting with various sizes to see which one formed a target big enough to grasp with a stylus. Since these rectangles are simply graphics drawn onto a picture box rather than explicit controls, they do not offer native drag functionality. Instead, this has to be approximated by toggling a flag when mouse/stylus is down within the rectangle of the drag handle indicating that a drag has started. When the mouse/stylus is lifted, if the flag was on and the cursor position has changed, this is viewed as a drag, and the change in position is taken to be the size change desired. This

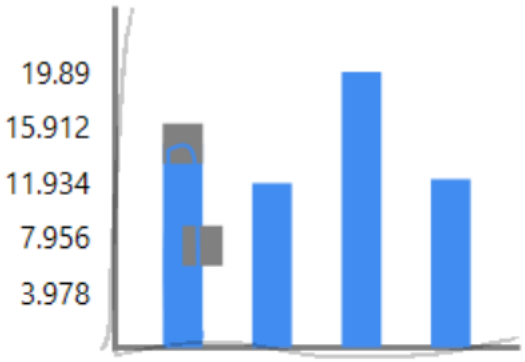


Figure 3.8: Drag handles

scaling is then applied to the variables previously introduced for positioning and scaling bars, so the changes take effect in the next paint cycle, which seems fairly instantaneous to users, if not perfectly so.

To help users discover what these drag handles do, the cursor changes to a sideways drag cursor in the direction corresponding to whether it is a height or width handle, when it hovers over these handles.

At first, the naive implementation of the scaling broke when users tried particularly extreme drags, which they tended to play with even if it wasn't the scale they finally intended to use. The calculations had to be refined over time.

Maintaining the legend

To keep the user aware of the state of the system at all times, it is important to have a legend showing which data is connected to which element of the visualisation. While most charting tools have a separate legend, we decided, in the theme of direct manipulation, that the colour key should be right where the data is - in the list of column headers.

However, the list of column headers is in the Sketchography form, whereas the colours are stored within the SketchChart, which is referenced in the form. We followed a publish-subscribe pattern here, by creating an event in SketchChart that notifies about changes to the legend. SketchChart also exposes a public Legend dictionary, mapping colours to headers. Sketchography subscribes to those notifications, and its event handler regenerates the

Figure 3.9: Legend

legend by checking the list in SketchChart.

Switching between the formal and informal views

We initially did this by offering a simple toggle button that read 'Toggle View', and a text box that showed the current view the user was in 'Sketch View' or 'Chart View'. However, during hallway testing, users repeatedly clicked on the text box, or were just generally confused about which view they were in. This confirmed a previous worry that the disconnect between the action trigger and the reflection of its actions would be confusing, so we implemented radio buttons as shown below. These contain pictures of the two views, so users can spot the difference visually without needing to internalise what the two phrases represent. Another concept of showing visual layers with live previews was not tested because of time and technical constraints.

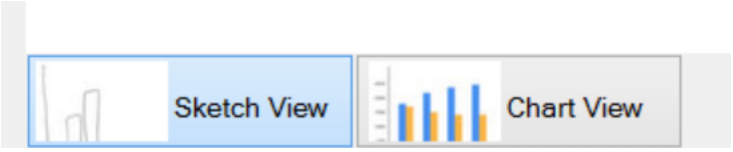


Figure 3.10: Switching between formal and informal views

Behind the scenes, all the chart elements are drawn in both views, with the unselected view dimmed or ghosted out. Hence, when users are sketching, they can see faint feedback of the recognition in the background and thus notice early if there has been a misclassification. This is done by setting two alpha or transparency values - one which is high when the formal view

is selected, and one which is high when in the informal mode. Each of the elements then uses one of these alpha values to draw themselves according to which mode they should be prominent in.

3.3 Summary

This chapter has detailed how various obstacles were overcome to implement all the core requirements, as well as three extensions. It describes how various design tradeoffs were evaluated to painstakingly make a design that would maximise usability for users by being the easiest to learn. It also describes how the code components fit together, with one main window which contains the sketch-based charting component. The charting component in turn has instances of various chart element objects.

Chapter 4

Evaluation

Since design usability was the salient point of this project, a user study was a good way to assess its performance. After gaining approval from the Ethics board, a pilot study was run, followed by the actual user study. The results confirm the hypotheses - users learnt how to use the system after one use, and they were able to modify their chart faster in Sketchography compared to Excel. A description of the study and a summary of the results follows; for the entire questionnaire and full data analysis, see appendix blah.

4.1 Study Goals

In the Introduction (chapter 1), it was hypothesized that the direct manipulation and liveness of this system would offer two advantages:

H1 This interface is more 'learnable' over time

H2 It encourages exploratory data visualisation creation by making modification easier

H3 Its advantages hold independent of how fluent the user is in English.

Hence, the study was designed to evaluate whether these three properties were achieved.

Learnability

To see whether the interface is learnable over time, users were asked to carry out a very similar task twice, and their times across tasks were compared.

If the user shows a statistically significant improvement in the time taken to carry out the task, then they are learning how to use the interface, and becoming better at using it. The chosen task is to import (mock) data and create a bar chart.

To make this experiment fair, the tasks must be phrased very carefully. The tasks must be very similar, to make sure one isn't systematically harder or easier, but not so similar that users can mechanically follow the same steps without even understanding the problem.

In addition, the tasks must tell the user what output is expected, without explicitly listing every single step required to get there. This didn't hold true for the first draft of the task descriptions, an example of which is below.

> Task A:
>
> 1. Load the 'Task A.xlsx' file.
> 2. Create a bar chart of 'Births' and 'Deaths', with 'Year' on the X Axis.

Clearly, this would leave little room for users to demonstrate that they have learnt how to use the application, since the task guides them through exactly what they need to do. Thus, the task was rephrased to focus on the problem (note, the data schema of the file was also changed to focus on differences between countries rather than years).

> Task A:
>
> You work in a government agency and are trying to see how population is growing in 4 different countries. The 'Task A' file contains information regarding how many births and deaths occurred in 1 year in each country (the numbers are scaled to account for population).
>
> Use the Sketchography app to create a column bar chart that will allow someone to compare birth and death rates across the countries.
>
> Speak out aloud the name of the country where the population is declining (where the death rate is larger than the birth rate).

This is better, since the user now needs to understand the context of the problem, grasp which data is relevant and which fields need to go on each axis for them to notice the trend required.

Modification

To see whether the interface encourages exploration, the time required to make a specific modification in Sketchography needed to be compared to the time required for the same modification in users' usual chart editor. The only other chart editor we compared was Microsoft Excel, which proved a sensible decision since all the final study participants indicated that Excel was their tool of choice for making charts, and only 2 out of 10 had actually used anything else to create a chart.

To make the comparison fair, we needed a modification that was a reasonable change to make quickly, and was supported by both editors. Two such changes are to change the width of the bars, and to change the height of the bars, without changing the dimensions of the chart itself. In the pilot study, some participants were asked to change the height, while others were asked to change the width. However, in Excel, the user has no direct control over the width of the bars. Instead, they have to do this indirectly by changing a 'Gap Width' setting, which controls the space between the bars. This indirection was not a perfect analogue to the method to change the bar width in Sketchography, so some users never identified how to make this change. Thus, in the final user study, we focused instead just on changing the height of the bar.

The users weren't shown how to make this change, or that the option was only available in the formal view (since stylus interaction with the chart in sketch view would be interpreted as ink). They had to explore the User Interface and discover the way to do this themselves. The method involved hovering over a bar in the data series, which exposes a drag handle on the top and the right of the bar. The top one can be dragged to manipulate the height (more specifically, the Y axis scale). In Excel, it involved opening an 'Action Pane' by double clicking on a bar, or right clicking on it and then selecting the right option, or via a button in the ribbon interface on the top of the screen. In this 'Action Pane', there was a setting to change the Y axis scale.

Language Independence

A secondary benefit of the direct manipulation and visual metaphors is that Sketchography isn't as language dependent as a system with configuration dialogues in English. Users should be able to learn Sketchography just as

quickly whether they speak English as their primary language or not.

India is a nation of multiple languages, but a lot of businesses conduct their activities in English. By carrying out the study there, participants with varied levels of fluency in English can be recruited.

To study the effect of language on the two measures above, users were asked what their primary language was, on their questionnaire. This was then divided into a binary classification: English or Non-English.

Other factors

A number of factors could affect people's performance on these tasks. For example, users who regularly create charts in Excel might be faster. To investigate how these factors affect the measures above, a questionnaire was given to users following their completion of the tasks, asking:

1. How often do you make charts for work or studies?
2. What tool do use for making these charts?
3. How many years have you been using that tool for?
4. Have you used Microsoft Excel to make charts?

Additionally, they were asked some open-ended questions to gain qualitative observations on their thoughts on Excel and Sketchography.

The exact phrasing of the questionnaire and the task descriptions is in the Appendix TODOexc.

4.2 Pilot Study

The pilot study was conducted on 3 participants, all grad students at the University of Cambridge Computer Lab, who were not paid for their time. It was carried out in a controlled lab environment, with a video recording set up that focussed on their interactions with the application, and also recorded what they were saying as they interacted with it.

The study proceeded as follows:

1. The participant signs a form indicating his consent to participating, and being recorded for the purposes of the study. They are also informed

that they are being timed not to assess their own performance, but that of the system, in order to put them at rest.

2. I demonstrate the system to them by importing a data file, creating a bar chart, switching between formal and informal views and resetting the application to its initial state.

3. I explain that they will be given a task description on a slip of paper, and can take as long as they like to read it. When they are ready, I will start timing. I will not answer any questions they may have about the system.

4. The Surface tablet is presented to them, with the application open in its initial state. The default folder when the 'Open File' dialogue is opened contains only the three Excel files they need for their tasks, and are named 'Task A.xlsx' and so on.

5. They are given the 2 creation tasks, followed by the two modification tasks. Comments they make out loud during this time are noted down.

6. After they finish all 4 tasks, they are given the questionnaire.

7. After they finish the questionnaire, they are free to leave. Most stay to make comments about the software, which are also noted down.

The key things we learnt were:

1. The task descriptions gave the steps in too much detail, as described above.

2. The width modification task isn't appropriate, as described above.

3. Despite it being subtly mentioned that each chart element must be drawn with an individual stroke, some users draw both the X and Y axis in one stroke as an 'L' shape. The program is modified before the final study to recognise these axes correctly and instantiate the two Axis objects accordingly.

There were no structural issues with the way the study itself was carried out, but the timing techniques and the script used during the demonstration did get refined with practice, making for a more valid final user study.

Age group	Number of participants
20-30	4
30-40	1
40-50	4
50-60	1

Table 4.1: Distribution of participant ages

4.3 User Study

The final user study was conducted with the same structure as the pilot study since no issues were faced. To get a fair mix of native and non-native English speakers, it was carried out in Mumbai, India. 10 participants with various levels of experience with Excel, and aged between 20 and 60 years old, volunteered. They were all white collar works in various industries, some of whom used computers on a daily basis.

One issue that our pilot study had not caught that surfaced in the user study was to do with the recognition of the bars. Some of the Indian participants drew bars as two strokes, which none of the English participants had done. Thus, one of the strokes making up the bar would get recognised as an axis rather than a bar. This led to some interesting observations. The program has the ability to erase and redraw strokes that have been misclassified. However, for the purposes of the study, this feature was not demonstrated to users for 2 reasons:

1. The eraser is one of the less discoverable aspects of the application. Any of the participants could have noticed the eraser like shape at the end of the stylus and tried pressing it. I wanted to see if any users would discover this on their own, without being shown. None of the 10 did so, presumably because users have previously been familiar only with capacitative styluses that don't have such a feature.

2. No matter how robust the stroke recogniser is made, there will be classification errors during usage. It was important to note how users react to this, understand what has happened, and correct the problem.

In practice, each of the users who made these split bars found a way around this. Some took the less desirable option of hitting 'Reset' and starting over, but the majority just drew over their previous sketches. When a

new axis was drawn, the old, misinterpreted one was forgotten, and the chart got corrected. Then, the user would draw the bar with one stroke as required.

4.3.1 Learnability

Our hypothesis was that the time taken for the creation task the second time around would be lower, and this was confirmed by a Paired T Test as shown below.

```
> t.test(Task1, Task2, paired = TRUE)

Paired t-test

data:  Task1 and Task2
t = 5.2186, df = 9, p-value = 0.0005502
alternative hypothesis: true difference in means
is not equal to 0
95 percent confidence interval:
 23.34058 59.05942
sample estimates:
mean of the differences
                   41.2
```

The test is valid if the data follows a normal distribution, which a Q-Q plot and Shapiro-Wilk test proved to be true for this data.

Dataset	W	p-value
Task 1	0.8435	0.04854
Task 2	0.8317	0.03511

Table 4.2: Creation task data was normally distributed

4.3.2 Modification

The hypothesis was confirmed that people took significantly less time to modify the chart in Sketchography than they did in Excel.

A Paired T Test show this to be true (p-value = 0.001389), but was inconclusive since the data was not normal in this case (p-value = 0.3966 for Sketch and 0.1896 for Excel).

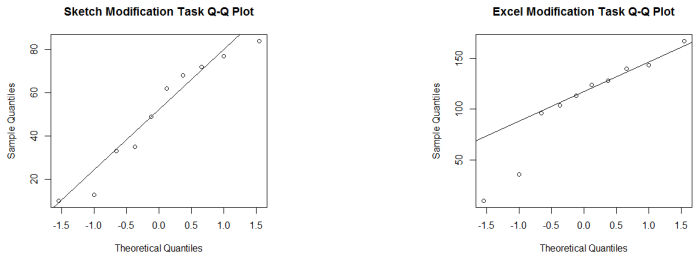


Figure 4.1: Modification task data wasn't normally distributed

A Wilcox test, which doesn't rely on the same assumptions of normality, confirmed the hypothesis true.

```
Wilcoxon rank sum test with continuity correction

data:  Sketch and Excel
W = 15.5, p-value = 0.01014
alternative hypothesis: true location shift is not equal to 0
```

Qualitatively, a lot of participants were observed trying to drag the bounding box handles that appear when you select a bar in the Excel chart. They were frustrated when they realised that these handles can't actually be manipulated. One participant seemed to have prior experience with this - on reading the task, they said out loud, "Oh yeah this is a pain, I know this is a pain".

Even when users did find relevant options in the Excel configuration panels, three of them hovered over the button, hesitated, and then tried something else instead. One actually said "But this is different, no?", before abandoning it. They weren't able to instantly preview what result they would get if they picked the button, so they avoided exploring that possibility.

On the other hand, they felt comfortable dragging the height handle in Sketchography despite it not having a label confirming its purpose.

One user even tried using the interface with a mouse, and by pinching her fingers apart on screen in a 'zoom in' gesture. This suggests a willingness to experiment. It also suggested some future work for this project, to integrate finger gestures to manipulate the chart while using the pen purely for inking.

4.3.3 Language Independence

Half the participants spoke English as their primary language with native fluency, while the other half used it during their job but not as a primary language. Tests to show that the learnability for English speakers was more than that for non-English speakers failed. While not statistically conclusive, this suggests that there wasn't a significant difference in learnability depending on language, which supports the hypothesis.

4.3.4 Other observations

Given that some participants might have a lot of experience with Excel and none with Sketchography, making it an uneven playing field, data was collected about how frequently they use Excel. As might be expected, the less frequently the participant used Excel, in general, the bigger the gain in speed they got from using Sketchography for the Modification task. There aren't enough samples per frequency category (Daily, Weekly, ...) to run a conclusive statistical analysis, but the trend is visible in Figure 4.2.

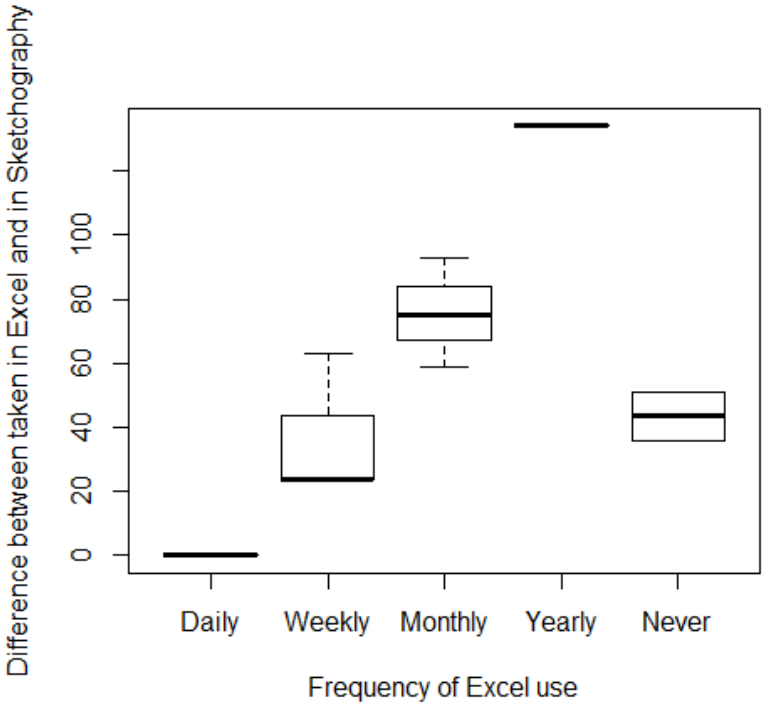


Figure 4.2: Less frequent users of Excel gain a bigger speedup from using Sketchography

No correlation was observed between the creation time task difference and the number of years the participant had used computers (Pearson's product-moment correlation p-value = 0.1978). Similarly, there was no correlation between creation time difference and the years of Excel use(p-value = 0.1386).

Interestingly, there is no correlation between the modification tasks time difference and whether the participant has used Excel for charting (p-value = 0.5753). This suggest that users might still not have learnt how to use Excel after having used it before, unlike Sketchography where they showed a measurable speedup after having used it once.

4.3.5 Participant comments

Some of the interesting responses to the open-ended questions are included below.

Question: How do you think the software works?

(Note: This question intended to elicit whether the participants had formed a correct mental model of how the software functions, but some of the participants less averse with English misinterpreted it as soliciting feedback, which led to even more interesting answers)

"Easy and real fast."

"Works pretty well. It is much easier to understand this software as compared to Excel."

Question: What is the best thing about Excel?

"Can easily plot the data and could be understood easily."

"All pervading."

Question: What is the worst thing about Excel?

"Certain times it is time consuming."

"Too vast to remember everything."

"Flexibility in charts is fairly low - it's templatized for the most part and not customize-able."

"Cumbersome."

Question: What is the best thing about Sketchography?

"Very efficient software. Easy to understand. Charts can be prepared in a matter of seconds."

"Intuitive."

Question: What is the worst thing about Sketchography?

"While making bars, not everybody will have the same style, so the Y Axis can get misjudged. Rest was easy and fun. Less time consuming. User friendly. Mistakes should get highlighted."

"The image recognition could be better. Apart from that, the software is easy to use, extremely intuitive, versatile and has great potential."

"A few kinks to work out, but on the right track."

4.4 Summary

In the introduction, this project outlined some hypotheses that it believed would be supported by the merits of the application. This chapter details a user study carried out on 10 participants that confirmed the two primary hypotheses with statistically significant results, and suggested that the third is supported too. Namely, users learn how to use the software on their first try and are already becoming skilled, and thus faster on their second try. They are also able to determine how to make a specific change quicker in

Sketchography than they are in Microsoft Excel. It is likely that they have been able to learn how to use the application and make the modification regardless of how conversant they are in English.

Chapter 5

Conclusion

Overall, this project went better than expected, resulting in three extensions being implemented. It was shown to be a statistically significant improvement over existing tools in terms of learning curve. This was the result of numerous design iterations, and a lot of code to develop the custom charting system that best matched the project needs.

5.1 Comparison with Requirements

The requirements originally listed in the preparation chapter(section 2.1) are restated below for easy comparison, to show that they were met and exceeded.

The system must allow the user to:

Core 1 Visualise data they have stored in common file formats.

Core 2 Specify the type of chart they want by drawing a likeness of it on screen using a stylus.

Core 3 Bind the data to the chart using an interface that makes it clear how the data is being used to generate the graphics.

Core 4 Specify visual properties of the chart, such as size and position, through the sketches.

Core 5 Manipulate the visual appearance of the created chart.

These core requirements were all met. In addition, time permitting, the system may:

Extension 1 Transform user-drawn sketches to resemble and overlap the formal chart elements they generated, in order to show the link between the sketch and the formal elements visually.

Extension 2 Mirror any manipulations applied to the formal layer back to the user drawn sketches, in order to keep the visuals of the formal and sketch layers in synchronisation.

Extension 3 Allow users to undo actions by erasing sketches, and remove the corresponding formal elements without throwing errors.

Extension 4 Allow users to manipulate any property of chart elements, not just one, so that the domain of visualisations they can create is infinite. For example, allow them to bind not just the height of bars in a bar chart to data, but also their width and colour.

Extension 5 Analyse the data and infer properties that may allow it to automatically suggest properties of the chart, such as which field belongs on which axis, or whether a data series should be log scale or linear scale.

Extension 6 Support exporting the chart as a Microsoft Chart object that can be embedded as a dynamic object in Microsoft Office files, not just as a raster image.

Extensions 1, 2 and 3 were implemented, resulting in an even more easily learnable design. Extension 4 could have been built since most of the technical platform required already existed, but was decided against in order to keep the application simple for users to understand, and make it possible to make charts quickly. The remaining two extensions were of sufficient technical complexity to not be feasible to complete within the time constraints.

Some usability goals were also specified:

Usability 1 Users must be able to create charts at least as quickly as they can using current charting systems.

Usability 2 Users must be able to build a mental model of the software's behaviour within 2 uses of it. They should thus be able to accurately predict the consequences of any action taken within the software.

Usability 3 Changes to the visualisation must occur through directly manipulating the visual representation of the chart, rather than through disconnected User Interface widgets.

Usability 4 The user must be able to easily try out changes to the visualisation, see what the resultant chart would look like, and undo them if needed.

Anecdotally, users were able to create charts within minutes, and made positive comments about the perceived speed. The user study demonstrated that they had built up a mental model by correctly applying the understanding they had gained from one task, to perform a second, similar task quicker. All changes to the chart are made directly on the chart, there is minimal other UI. The second half of the user study demonstrated that the application could encourage exploration, since users made the same modification to their chart significantly faster in Sketchography than the leading charting tool, Microsoft Excel.

5.2 Future Work

While the goals of the project were met, the exploratory design phase as well as the user testing generated a lot of ideas that weren't feasible to implement within the time scale.

The primary improvement that would make this tool more practical would be to enable exporting the chart, as an image, or better yet, as a Microsoft Office Chart object that can be embedded as a vector object in office applications.

Additionally, the technical platform built could be extended to support many more types of charts. Given the small amount of data collected to train the classifier, adding recognition support for many more chart elements could have lowered accuracy, which would be detrimental to the usability. However, with more time, more data could be collected, for a variety of chart types.

The charting component could also be made a bit smarter, to apply heuristics such as shifting the axes slightly to make them align perfectly.

Bibliography

Alvarado, C. and Davis, R. (2004). SketchREAD: a multi-domain sketch recognition engine. In *Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology*, UIST '04, page 23–32, New York, NY, USA. ACM.

Blackwell, A. (2002). What is programming. In *14th workshop of the Psychology of Programming Interest Group*, page 204–218.

Blackwell, A. F., Church, L., Plimmer, B., and Gray, D. (2008). Formality in sketches and visual representation: some informal reflections. *Proceedings of the VL/HCC Workshoph, Herrsching am Ammersee, Germany*, page 11–18.

Bresciani, S., Blackwell, A., and Eppler, M. (2008). A collaborative dimensions framework: Understanding the mediating role of conceptual visualizations in collaborative knowledge work. In *Hawaii International Conference on System Sciences, Proceedings of the 41st Annual*, pages 364–364.

Chang, S. H.-H., Plimmer, B., and Blagojevic, R. (2010). Rata. ssr: Data mining for pertinent stroke recognizers. In *Proceedings of the Seventh Sketch-Based Interfaces and Modeling Symposium*, page 95–102. Eurographics Association.

Chao, W. O., Munzner, T., and van de Panne, M. (2010). Poster: Rapid pen-centric authoring of improvisational visualizations with napkinvis. *Posters Compendium InfoVis*.

Fonseca, M. J., Pimentel, C., and Jorge, J. A. (2002). CALI: an online scribble recognizer for calligraphic interfaces. In *Sketch Understanding, Papers from the 2002 AAAI Spring Symposium*, page 51–58.

Frankish, C., Hull, R., and Morgan, P. (1995). Recognition accuracy and user acceptance of pen interfaces. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, page 503–510. ACM Press/Addison-Wesley Publishing Co.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The WEKA data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18.

Hammond, T. and Davis, R. (2006). LADDER: a language to describe drawing, display, and editing in sketch recognition. In *ACM SIGGRAPH 2006 Courses*, SIGGRAPH '06, New York, NY, USA. ACM.

Kara, L. B. (2004). An image-based trainable symbol recognizer for sketch-based interfaces. In *in AAAI Fall Symposium Series 2004: Making Pen-Based Interaction Intelligent and Natural*, page 99–105. AAAI Press.

Ko, A. J., Abraham, R., Beckwith, L., Blackwell, A., Burnett, M., Erwig, M., Scaffidi, C., Lawrance, J., Lieberman, H., Myers, B., Rosson, M. B., Rothermel, G., Shaw, M., and Wiedenbeck, S. (2011). The state of the art in end-user software engineering. *ACM Comput. Surv.*, 43(3):21:1–21:44.

Kuncheva, L. I. (2004). *Combining Pattern Classifiers: Methods and Algorithms*. John Wiley and Sons, Inc.

Lee, B., Isenberg, P., Riche, N. H., and Carpendale, S. (2012). Beyond mouse and keyboard: Expanding design considerations for information visualization interactions. *Visualization and Computer Graphics, IEEE Transactions on*, 18(12):2689–2698.

Microsoft Corporation (2013). Code metrics values.

Norman, D. A. and Draper, S. W. (1986). *User Centered System Design; New Perspectives on Human-Computer Interaction*. L. Erlbaum Associates Inc., Hillsdale, NJ, USA.

Oltmans, M. (2007). *Envisioning Sketch Recognition: A Local Feature Based Approach to Recognizing Informal Sketches*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA. AAI0819449.

Ouyang, T. Y. and Davis, R. (2009). A visual approach to sketched symbol recognition. In *IJCAI*, volume 9, page 1463–1468.

Patel, R., Plimmer, B., Grundy, J., and Ihaka, R. (2007). Ink features for diagram recognition. In *Proceedings of the 4th Eurographics workshop on Sketch-based interfaces and modeling*, page 131–138. ACM.

Paulson, B. and Hammond, T. (2008). PaleoSketch: accurate primitive sketch recognition and beautification. In *Proceedings of the 13th International Conference on Intelligent User Interfaces*, IUI '08, page 1–10, New York, NY, USA. ACM.

Plimmer, B., Purchase, H. C., and Yang, H. Y. (2010). SketchNode: intelligent sketching support and formal diagramming. In *Proceedings of the 22nd Conference of the Computer-Human Interaction Special Interest Group of Australia on Computer-Human Interaction*, page 136–143. ACM.

Rubine, D. (1991). Specifying gestures by example. In *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '91, page 329–337, New York, NY, USA. ACM.

Shilman, M., Pasula, H., Russell, S., and Newton, R. (2002). Statistical visual language models for ink parsing. In *In AAAI Spring Symposium on Sketch Understanding*, page 126–132. AAAI Press.

Shilman, M., Viola, P., and Chellapilla, K. (2004). Recognition and grouping of handwritten text in diagrams and equations. In *Frontiers in Handwriting Recognition, 2004. IWFHR-9 2004. Ninth International Workshop on*, page 569–574. IEEE.

Shneiderman, B. (1983). Direct manipulation: A step beyond programming languages. *Computer*, 16(8):57–69.

Sutherland, I. E. (1964). Sketch pad a man-machine graphical communication system. In *Proceedings of the SHARE Design Automation Workshop*, DAC '64, page 6.329–6.346, New York, NY, USA. ACM.

Tanimoto, S. (2013). A perspective on the evolution of live programming. In *2013 1st International Workshop on Live Programming (LIVE)*, pages 31–34.

Walny, J., Lee, B., Johns, P., Riche, N. H., and Carpendale, S. (2012). Understanding pen and touch interaction for data exploration on interactive whiteboards. *Visualization and Computer Graphics, IEEE Transactions on*, 18(12):2779–2788.

Wang, M., Plimmer, B., Schmieder, P., Stapleton, G., Rodgers, P., and Delaney, A. (2011). SketchSet: creating euler diagrams using pen or mouse. In *Visual Languages and Human-Centric Computing (VL/HCC), 2011 IEEE Symposium on*, page 75–82. IEEE.

Willems, D., Niels, R., van Gerven, M., and Vuurpijl, L. (2009). Iconic and multi-stroke gesture recognition. *Pattern Recogn.*, 42(12):3303–3312.

Wobbrock, J. O., Wilson, A. D., and Li, Y. (2007). Gestures without libraries, toolkits or training: A $1 recognizer for user interface prototypes. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology*, UIST '07, page 159–168, New York, NY, USA. ACM.

Yeung, L., Plimmer, B., Lobb, B., and Elliffe, D. (2008). Effect of fidelity in diagram presentation. In *Proceedings of the 22nd British HCI Group Annual Conference on People and Computers: Culture, Creativity, Interaction-Volume 1*, page 35–44. British Computer Society.

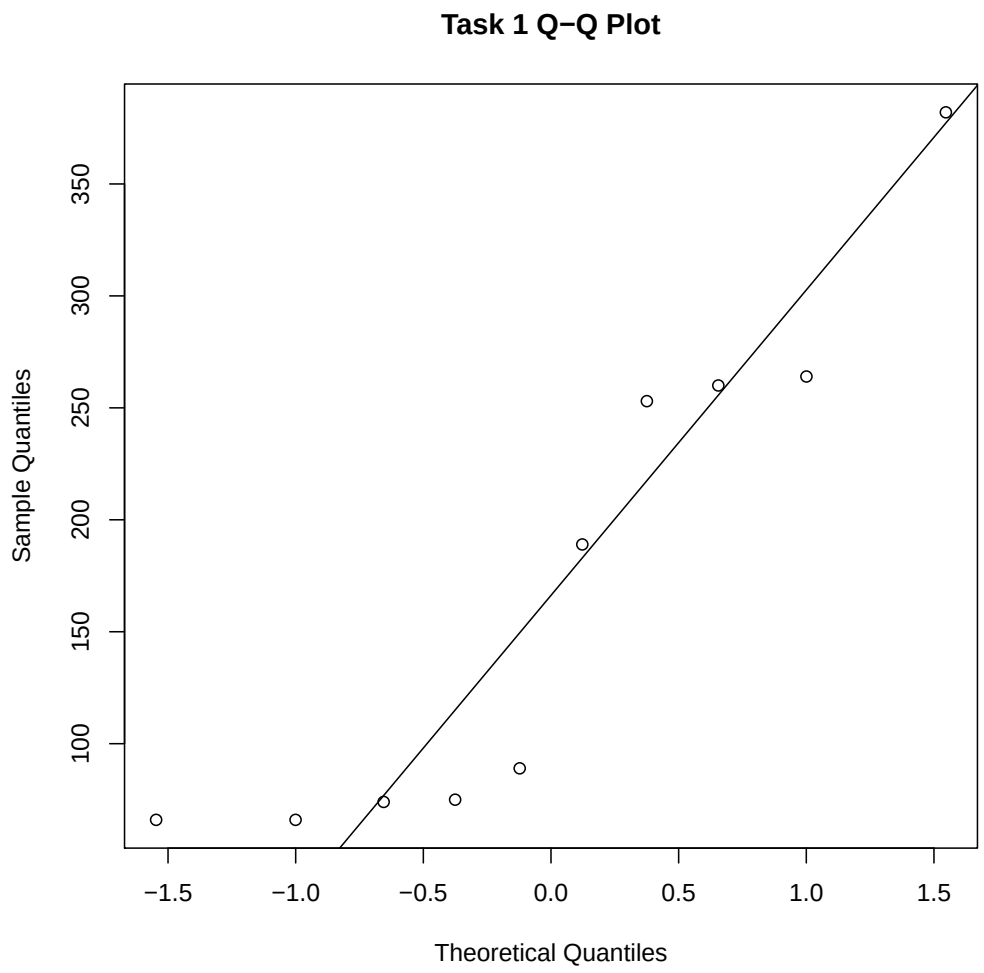
Yu, B. and Cai, S. (2003). A domain-independent system for sketch recognition. In *Proceedings of the 1st International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia*, GRAPHITE '03, page 141–146, New York, NY, USA. ACM.

Appendix A

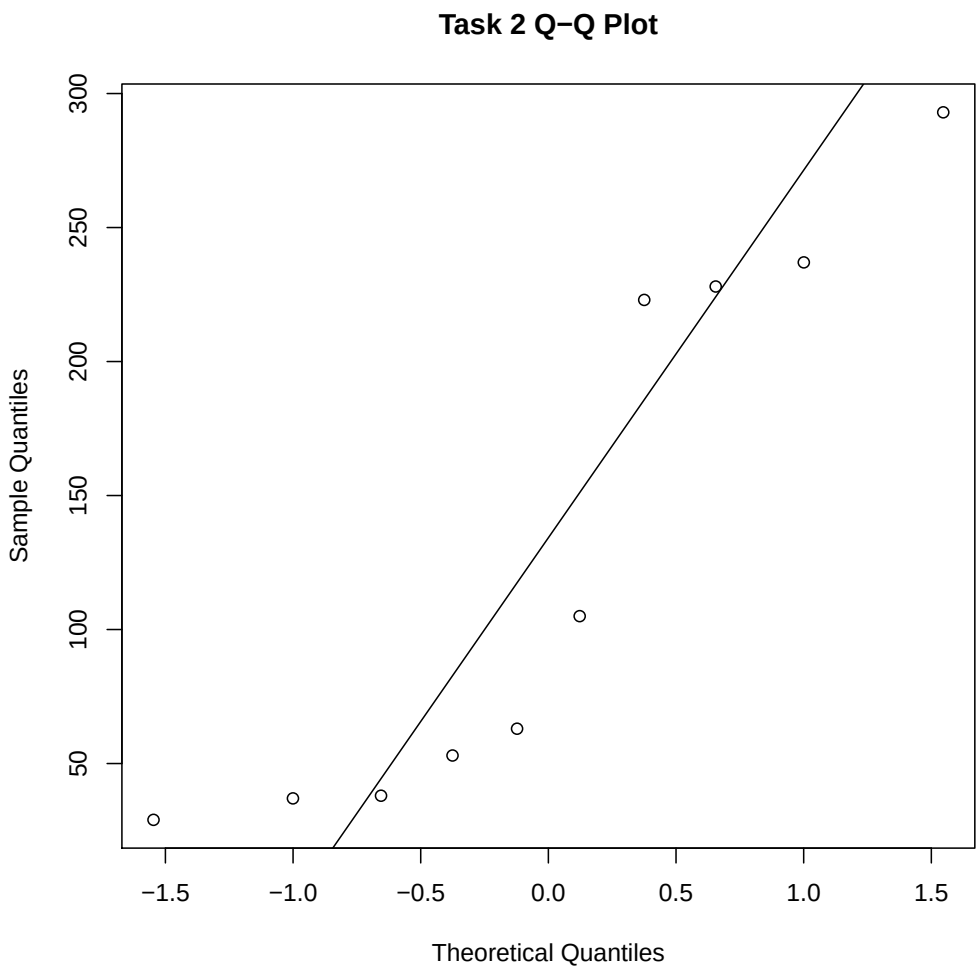
Study Analysis

Irrefutable evidence that users learnt how to use the software and got faster after one try.

```
t.test(Task1, Task2, paired = TRUE)

##
##  Paired t-test
##
## data:  Task1 and Task2
## t = 5.219, df = 9, p-value = 0.0005502
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  23.34 59.06
## sample estimates:
## mean of the differences
##                    41.2

qqnorm(Task1, main = "Task 1 Q-Q Plot")
qqline(Task1)
```




```
shapiro.test(Task1)

##
##  Shapiro-Wilk normality test
##
## data:  Task1
## W = 0.8435, p-value = 0.04854

qqnorm(Task2, main = "Task 2 Q-Q Plot")
qqline(Task2)
```



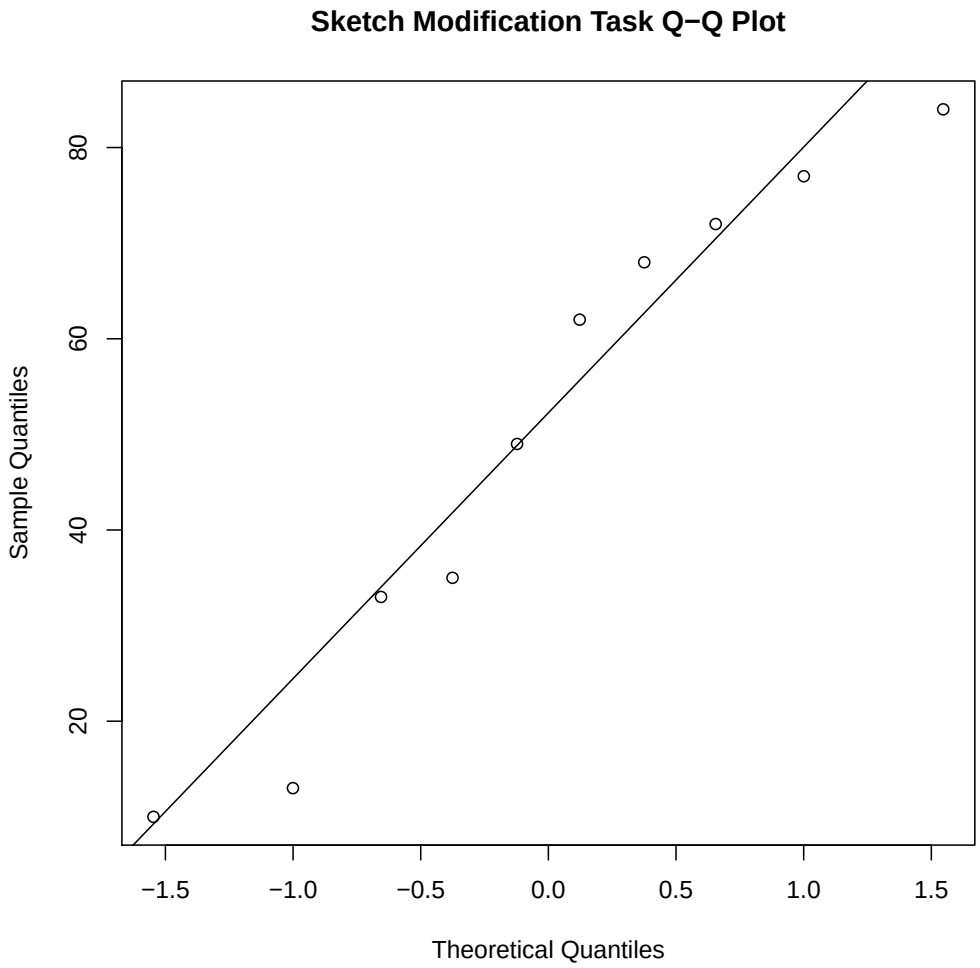

```
shapiro.test(Task2)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  Task2
## W = 0.8317, p-value = 0.03511
```

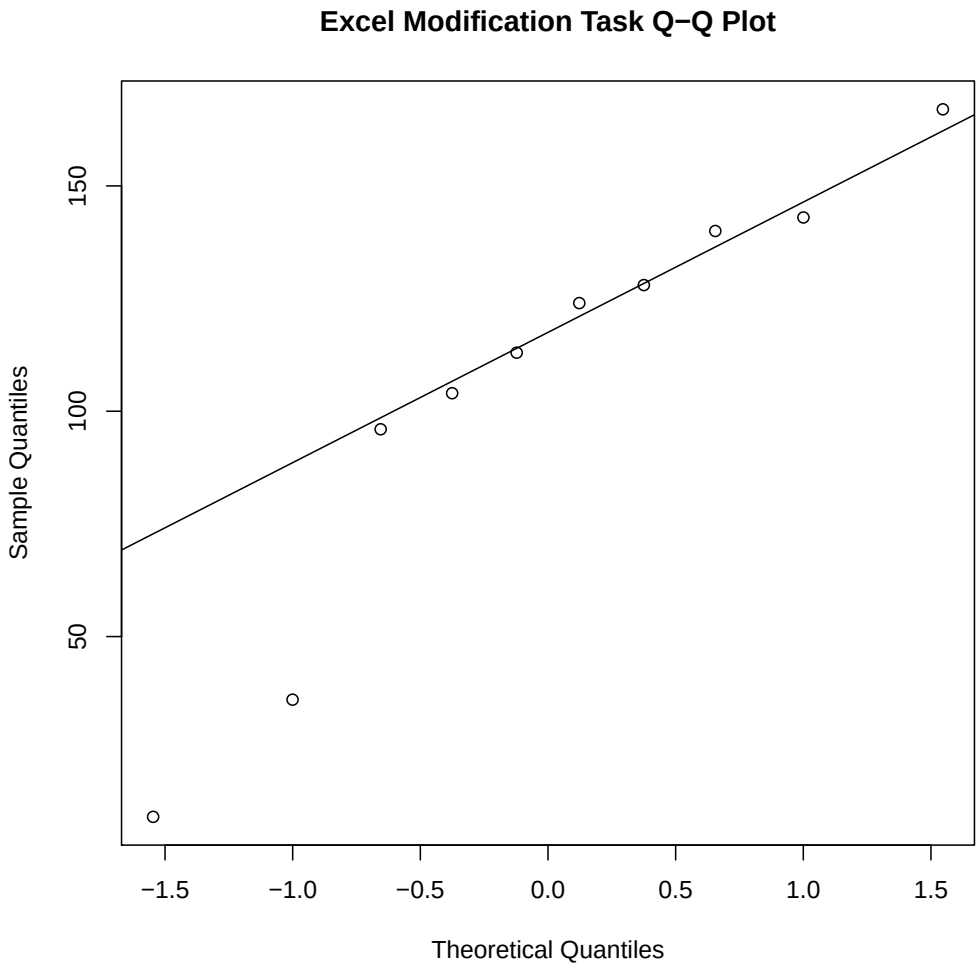
```
summary(aov(TaskDiff ~ LanguageBinary, data = data))
```

```
##                Df Sum Sq Mean Sq F value Pr(>F)
## LanguageBinary  1    518     518    0.81   0.39
## Residuals       8   5091     636
```

```
t.test(Sketch, Excel, paired = TRUE)

##
##  Paired t-test
##
## data:  Sketch and Excel
## t = -4.548, df = 9, p-value = 0.001389
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -83.55 -28.05
## sample estimates:
## mean of the differences
##                   -55.8

qqnorm(Sketch, main = "Sketch Modification Task Q-Q Plot")
qqline(Sketch)
```




```
shapiro.test(Sketch)

##
##  Shapiro-Wilk normality test
##
## data:  Sketch
## W = 0.9246, p-value = 0.3966

qqnorm(Excel, main = "Excel Modification Task Q-Q Plot")
qqline(Excel)
```




```
shapiro.test(Excel)

##
##  Shapiro-Wilk normality test
##
## data:  Excel
## W = 0.8943, p-value = 0.1896

wilcox.test(Sketch, Excel, Paired = T, exact = F)

##
##  Wilcoxon rank sum test with continuity correction
```

```
##
## data:  Sketch and Excel
## W = 15.5, p-value = 0.01014
## alternative hypothesis: true location shift is not equal to 0
```

```
summary(aov(ModDiff ~ LanguageBinary, data = data))
```

```
##                Df Sum Sq Mean Sq F value Pr(>F)
## LanguageBinary  1   1392    1392    0.92   0.37
## Residuals       8  12153    1519
```

```
cor.test(TaskDiff, ComputerYears)
```

```
##
##  Pearson's product-moment correlation
##
## data:  TaskDiff and ComputerYears
## t = 1.405, df = 8, p-value = 0.1978
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.2568  0.8393
## sample estimates:
##    cor
## 0.4448
```

```
cor.test(TaskDiff, ExcelYears)
```

```
##
##  Pearson's product-moment correlation
##
## data:  TaskDiff and ExcelYears
## t = 1.365, df = 8, p-value = 0.2094
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.2684  0.8356
## sample estimates:
##    cor
## 0.4346
```

```
cor.test(ModDiff, ExcelYears)

##
##  Pearson's product-moment correlation
##
## data:  ModDiff and ExcelYears
## t = -0.5841, df = 8, p-value = 0.5753
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.7379  0.4898
## sample estimates:
##     cor
## -0.2022
```

Appendix B

Project Proposal

N. Satra
Pembroke College
ns532

Part II Project Proposal

Creating Custom Visualisations using Sketch Recognition

16 October 2013

Project Originator: Neil Satra
Project Supervisor: Alistair Stead
Director of Studies: Chris Hadley
Overseers: Lawrence Paulson and Richard Gibbens

Aim

My aim is to build a system that allows users to visualise data in novel ways. They will achieve this by sketching an example of the visualisation for a small sample of their data, and demonstrating how the two are linked. The software must then recognise the sketch, understand the way it relates to the data, and generate the entire visualisation for the full dataset accordingly.

Introduction and Description of the Work

The amount of data being generated is constantly increasing, but the ways to easily visualise this data are still limited by the input mechanisms and tools of yesteryear. Through this project I want to enable the user to build more appropriate representations than they can using traditional graph creation tools like Microsoft Excel, and do it faster too.

A sample workflow for an end user might be as follows:

1. Import the data file into the application
2. The user will use a stylus with the touch screen to draw the axes. The software uses sketch recognition to recognise that they are lines.
3. The user will use a stylus to draw a bar. Once the tool has recognised it as a rectangle, indicate that you want to link the height attribute of the rectangles to a field from the data. The software will automatically generate bars for the rest of the data.
4. Additionally, the user could write the name of the fields they want on the axis, and the software will populate the chart with the relevant data (extension)

This tool will still allow users to generate simple representations like bar graphs. However, it will also allow them to create variants of these simple representations, such as a bar graph where the horizontal position of the bars conveys data. Additionally, they could create complex representations such as an x-y plot with circles where the area conveys data. Rather than defining an extensive domain of possible graphs, this tool shall provide them with basic elements such as points, lines, rectangles and circles that they can then combine to form complex visualisations. Using sketches, users can easily

input what they are picturing, instead of having to translate their mental picture into rectangle, circle and line tools offered by traditional drawing interfaces.

Resources Required

As well as use of the MCS computers and networked storage space, the following resources will be needed:

1. **Surface Pro**(Intel Core i5 - 1.70 GHz, 4GB RAM, 256 GB SSD), which has a pressure-sensitive touchscreen with digitizer and a compatible stylus. This is needed as an input mechanism for sketches. Insurance will be purchased for the device to ensure quick replacement in case of damage/loss/theft. The lab also has several Android tablets that could be used as substitutes if my Surface fails, since Rata works on Android as well. Documents will be backed up to Dropbox and WriteLatex, and code to Dropbox as well as Github.

2. **Rata sketch recognition framework**, which is available for download for non-commercial use. [1] Rata is one of the leading tools to build and train analysers for specific domains, and is written by Beryl Plimmer. Beryl was working in Cambridge until recently and is still in contact with us.

Starting Point

I have no prior knowledge or experience in this field. However, I'm taking courses in AI and Computer Vision over the coming terms which should teach me concepts I might need to draw on, for example to understand how Rata works in detail, and to create simple shape classifiers if Rata doesn't fit this project's needs. I shall also supplement these courses with online resources. There are sources to draw inspiration from in terms of design decisions, including Microsoft's SketchInsight[2] and Bret Victor's research[3] which itself is based on previous HCI research in the fields of Direct Manipulation, Programming by Example and Programming by demonstration. Some research on 'liveness' has also been conducted by Steve Tanimoto, who will be visiting

[1] Rata Framework - `http://www.cs.auckland.ac.nz/research/hci/digital_ink/ink_recognition/rata_recognizers.shtml`

[2] Microsoft Research, SketchInsight - `http://research.microsoft.com/en-us/um/redmond/groups/cue/sketchinsight/`

[3] Bret Victor, "Drawing Dynamic Visualisations" - `http://vimeo.com/66085662`

the Computer Lab Rainbow Research Group this term. Some of my design principles seem to match the philosophy of the now-discontinued Protovis.[4]

I will also be learning how to use Rata from scratch, but my supervisor has some experience with it, and I am in contact with the team that developed it, including the lead on the project who was at the Computer Laboratory until recently. Rata has some of the best precision and recall rates of sketch recognition software, and hasn't been used much in the past, so it should lead to a fairly successful recognition tool.

Substance and Structure of the Project

The project would involve building a dataset and training a classifier using it in conjunction with Rata. It would also require writing software that can store sketch input from the user and analyse it using the classifier. It would also have to take data as input and extract field names. It would then have to present a user-friendly interface for users to tag attributes of shapes in the sketch to fields in the data, and have the software generate similar shapes for the rest of the data accordingly.

The project has the following main sections:

1. Studying algorithms and techniques required for recognising sketches of basic shapes, and correlating these shapes to common elements of visualisations.

2. Studying trends in data visualisations and infographics to identify the common building blocks such as bars and wedges. This may be achieved by surveying students who regularly create visualisations, as well as studying award-winning infographics and visualisations online.

3. Evaluating the various technology stacks available to decide the right tools for the job. It must be decided whether to write a stand-alone app or a Microsoft Excel add-in, and whether to use Rata-generated recognisers or write my own.

4. Designing the interface and interactions needed for users to input shapes, define bindings between data and attributes of shapes (eg correlating

[4]Protovis: `http://mbostock.github.io/protovis/`

the 'population' field to the radius of the circle) and instruct the tool on how to recreate the shape for the rest of the data points.

5. Developing and testing the code for the algorithms referred to in (1) and the interface in (2). A more detailed breakdown of this development is presented in the Timetable.

6. Evaluation and the preparation of examples to demonstrate that the implementation has been successful. A working demo will be presented to my supervisor. There might be a quantitative evaluation of how accurately the tool-generated graph matches what the user's envisioned. There may also be a user study wherein users could be asked to present the same data, using this tool as well as other visualisation tools including Microsoft Excel and D3.js. Then they will compare the speed and learning curves of the different solutions as well as the appropriateness of the resulting visualisations.

7. Writing the dissertation.

If time allows, there might be scope to build extensions to the project such as the ones below:

- Write my own simple classifier, since if the number of shapes to identified are small, it may be just as much effort to write my own classifier as to learn and integrate Rata.
- Rather than limiting links from data to specific attributes of the shapes, an extension could be written that let's the user specify custom axes through gestures. eg. Diagonal of a rectangle or 'radius' of a star.
- Build more combinations of ways to link data to shape attributes, so that for example the user could link the 'Country' data field to the shape's colour to have it generate a colour scheme for the legend.
- Build simple heuristics to automatically calculate which attribute specified data fields may be best tied to. For example, if a field is numerical and a line is drawn, the height of the line is the most likely attribute to be linked. This could make it possible for the user to just drag the data field to the axis and have it generate the graph without further instruction. Heuristics could get very technically challenging to build, but the complexity of heuristics implemented could be decided based on time available.
- See if there are any opportunities for improvement in recognition by the Rata-generated classifier, perhaps by pre-processing the strokes in some manner.

Success Criteria

The following should be achieved:

- Read data from files and successfully extract fields and associated data.
- Accept and temporarily store sketch input.
- Recognise at least 4 basic shapes such as point, line, rectangle and circle.

- Let user successfully link data to compatible attributes of shape. The interface should make it evident how to create the link, and should work well with stylus input.
- Generate shapes for rest of data successfully.

Timetable and Milestones

Summary

In Michaelmas I will study past research in the field, study the tools available, and iterate on the design of the tool. I will also carry out some initial training of a classifier. During the winter holidays I will cover a lot of ground on implementation. The Rata classifier will be completed and the user interface and data import functionality shall be coded. During Lent term I will prepare the progress report and presentation, and continue work on implementation. Easter holidays will be used to write the dissertation, and design the evaluation. Finally, in Easter term, I will conduct the evaluation and finalise the dissertation. Work will be kept light to allow time for exam preparation.

Weeks 1 - 2

28 Oct - 10 Nov, 2013

Research past work in the areas of direct manipulation, programming by demonstration and by example, and liveness. Study existing visualisation tools. Research visualisation building blocks by studying various visualisation tools. Build corpus of common and unique visualisations from internet sources.

Milestones: Written paper summaries of the major works in this field. Prepared list of shapes to recognise (eg. bars, lines, wedges)

Weeks 3 - 4

11 Nov - 24 Nov, 2013

Continue studying visualisation tools. Research sketch recognition techniques and compare with what Rata provides. Assess costs and benefits of

implementing my own classifier. Study various possible technology stacks such as a stand-alone C# app or an MS Office plug-in.

Milestones: Have decided whether Rata can be used, and what stack to implement the tool on.

Weeks 5 - 6

25 Nov - 8 Dec, 2013

Based on the technology stack decided, design the flow of the app. Think about the interactions required to import data, sketch visualisations for a couple of points, link data to shap attributes, and then instruct the tool to generate the rest.

Milestones: Having examined the ideal workflow and those made possible by each of the technology stacks, have finalised the technology stack. Have a rough design ready.

Weeks 7 - 8

9 Dec - 22 Dec, 2013

Study the Rata framework and train it to recognise the set of shapes identified earlier. Understand entry points that can be used to integrate it with rest of tool.

Milestones: Have a basic analyser for shapes that works reasonably well. Have general understanding of Rata's capabilities.

Weeks 9 - 10

23 Dec - 5 Jan, 2014

Build C# app or 'app for Office' to present UI. Implement data import and extraction of headers. Design and build architecture of shapes and their attributes.

Milestones: Have a partially implemented front-end ready. Demonstration of user successfully importing a csv file and the data being show inside the app.

Weeks 11 - 12

6 Jan - 19 Jan, 2014

Lent term starts 14 Jan. Flesh out the app to accept strokes, store them, and send them through the classifier. Get results, create appropriate objects accordingly, and expose their attributes for linking data.

Milestones: Demonstration of the user drawing strokes on screen and the tool recognising the shape and exposing relevant attributes.

Weeks 13 - 14

20 Jan - 2 Feb, 2014

Buffer time for unfinished work. Progress report due on 31st January. Review remainder of project plan in view of program development to date and adjust as necessary. Write the Progress Report drawing attention to the code already written, incorporating some examples, and recording any augmentations which at this stage seem reasonably likely to be incorporated.

Milestones: Basic code now working, but probably not elegant. Progress Report submitted and entire project reviewed both personally and with Overseers.

Weeks 15 - 16

3 Feb - 16 Feb, 2014

Build functionality to link data fields to shape attributes. Also build functionality to automatically generate similar shapes for each of the other data points, and lay them out sensibly.

Weeks 17 - 18

17 Feb - 2 Mar, 2014

Polish the workflow. Explore options for exporting the created graph. Conduct hallway testing. Design user study or alternative evaluation methods. Recruit volunteers.

Milestones: A smooth workflow for users, established through hallway testing and demonstration to supervisor. Have a design for the user study and recruited volunteers if needed.

Weeks 19 - 20

03 Mar - 16 Mar, 2014

Conduct user studies. Easter holidays start 15 Mar.

Milestone: User study data collected.

Weeks 21 - 26

17 Mar - 27 Apr, 2014

Work on the dissertation. Easter term begins 22 Apr.

Milestones: Nearly final draft of Introduction, Preparation and Implementation chapters of Dissertation complete. Implementation chapter 90% complete.

Weeks 27 - 28

28 Apr - 11 May, 2014

Work on the project will be kept ticking over during this period but undoubtedly the Easter Term lectures and examination revision will take priority. Refine implementation, Evaluation and Conclusion chapters as needed.

Milestones: Code performs well. Dissertation essentially complete, and proof-read by Supervisor and possibly friends and/or Director of Studies. Buffer period in case anything falls behind schedule.

Week 29

12 May - 18 May, 2014

Dissertation due on 16th May. Submit Dissertation.

Milestone: Submission of Dissertation.