Who Changed My Annotation? An Investigation into Refitting Freeform Ink Annotations

Craig Sutherland, Andrew Luxton-Reilly, Beryl Plimmer
Department of Computer Science
University of Auckland
Private Bag 92019, Auckland, New Zealand
Email: {cj.sutherland,a.luxton-reilly,b.plimmer}@auckland.ac.nz

***Abstract*—Pen-based computing allows users to seamlessly add freeform ink annotations to dynamic digital documents. When these documents change, the annotations should adapt, or "refit", in response to these changes. While previous studies have demonstrated ways of refitting, there have been few investigations in what the users expect to happen. In this paper, we report on an investigation into user expectations for refitting freeform ink annotations. We explored how the four main categories of annotations can be refitted and what user thought of each approach. Our findings indicate the approach depends on the user's overall preference and the category of annotation. Furthermore we suggest ways to implement freeform ink annotations on dynamic digital documents.**

I. Introduction

As pen-based devices become more popular there are new possibilities for using them. One area is incorporating freeform ink annotations onto dynamic digital documents. Freeform annotations offer benefits over text-based annotations in that they are better for comprehension and memory and have a lower cognitive load [1], [2]. But we do not know how users expect their annotations to change when the document changes.

Historically people are used to freeform annotations on paper where the layout is fixed. Adding freeform annotations to static documents is a simpler task as the implementation can assume there will be no changes to the base document. But pen-based devices afford the adding of freeform annotations to dynamic documents[1]. Thus, if the document changes any annotations must adapt to match the changes in the content, otherwise the meaning of the annotation will be lost. In this paper, the term refitting refers to the process of automatically adapting an annotation when the underlying content changes. How to refit annotations is a current gap in the research [3].

Refitting annotations is difficult for two reasons. First, there are the technical challenges to elegantly refit annotations. Second, user expectations are unknown. People are used to annotations on static documents and do not know how their annotations can change. Therefore it is difficult to understand how they would react to different changes. There have been very few studies that have investigated refitting [4]–[7] and only one of these studies queried user expectations [4]. None of these studies investigated connectors: annotations that connect two or more items together. Understanding how connectors can refit is important as they are typically associated with locations in the base document content.

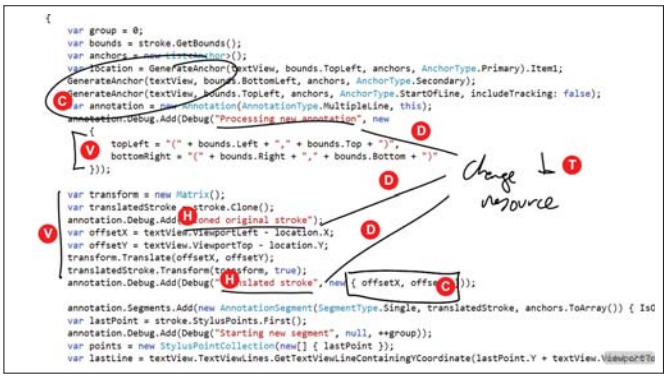

Fig. 1. The different categories of annotation. *H*: Horizontal lines; *V*: Vertical lines; *D*: Connectors; *C*: Enclosures; *T*: Text (not refitted).

In this paper we report on two iterations of our investigation into refitting annotations. Our investigation provides two main contributions. First, we provide insight on how users expect their annotations to refit when the underlying context changes. This includes several new algorithms for refitting annotations. Second, we expand the categories of annotations that can be refitted to include connectors. Taken together, these two contributions enhance our understanding of how to refit freeform annotations.

II. Related work

There are relatively few studies on how annotations can be refitted. A recent review [3] identified only four publications in this area [4]–[7]. The following subsection reviews these four studies and their implementations of refitting. Another gap in the literature is how connector annotations could be refitted. The second subsection reviews the literature on freeform ink connectors in the related domain of freeform ink sketching.

A. Refitting Annotations

In the earliest publication on refitting annotations, *XLibris* [6], annotations are classified as one of three "categories": single line (e.g. underlines), multiple line (e.g. margin bars and circled passages), or other. Single line annotations are split into

[1]A dynamic document is one that has the potential to change.

978-1-5090-0252-8/16/$31.00

segments: one per word. As the content changes the segments stay with the associated word. Multiple line annotations are scaled vertically. When a multiple line annotation is added, it is associated with two words: one each at the top and bottom of the annotation. The scale is then applied based on the vertical distance between these two words. Finally, other annotations are ignored [6].

The next reported system is *Callisto* [4]. *Callisto* uses similar rules to *XLibris*, including the same annotation categories. The main change is circled text was treated as a single line annotation. Like *XLibris*, single line annotations were associated with the underlying words and stayed in position relative to them; multiple line annotations were vertically scaled and all other annotations ignored. In addition to refitting annotations *Callisto* also introduced a "cleaning" mode where annotations were beautified and converted to a form that allowed for easier refitting.

Ramachandran and Kashi also reported a system that allowed for refitting [7]. Unfortunately their publication does not provide any details on how annotations are refitted.

These three implementations all used tablet PC devices. These devices collect ink directly on the screen. The fourth implementation, *Proofrite*, uses a pen-and-paper UI approach where the document is printed to paper, the ink collected with an Anoto pen and transmitted to a computer [5]. *Proofrite* uses the same rules as *XLibris* for refitting.

Only one publication [4] reports any evaluations of user expectations. In this study, users only wanted their annotations refitted if it was done correctly. Otherwise they preferred that the underlying content is locked so it cannot be changed. While the authors proposed a potential cut-off for detecting annotations that could be correctly changed they were unable to find one. They also found people were generally happy with cleaned annotations; however it increased their expectations of what the system would automatically do for them [4].

One limitation with all these studies is they only implemented a single change algorithm for each annotation category. For example, with *Callisto* the users compared modified and non-modified annotations [4]. There are many different ways of refitting annotations and potentially some are more acceptable than others to users. This then provides the premise for our study.

B. Connectors

A connector is an annotation that connects two (or more) items together. This could be two locations in the text, an annotation to a location in the text (see Figure 1), or two annotations together. While previous freeform annotation studies have not investigated connectors, connectors have been investigated in other domains [8]. One such domain is freeform ink sketching [9]–[11]. These studies suggested different approaches for manipulating the connectors on graph diagrams.

One approach is to scale the connector on an axis between the start and end points [10]. However when the connector is shortened it can look distorted. Otherwise it is a simple approach. A more complex approach was proposed by Arvo and Novins [11]. This approach uses the baseline between the two endpoints. If the baseline is shorter than the original the connector is condensed; otherwise the stroke is stretched. They compare the connector to a piece of string and interpolate the points with an ellipse the same length as the original baseline. The final study [9] investigated connectors that need to flow around another intermediary node. The authors implemented node avoidance using a cosine function (they called their approach *Context Reflow*.)

III. Our approach

The primary focus in our study is the users' opinions of refitting. To investigate this we implemented several different refitting algorithms. These algorithms were evaluated by users over two rounds of studies. The refitting algorithms were based on previous research [4], [6], ideas from a focus group of several researchers in the area of ink sketching, and feedback from the participants.

The algorithms all use the graphical representation of the strokes, rather than trying to infer any semantic representation. It is very difficult to understand the semantics of a stroke as its meaning is often ambiguous [12]. In contrast, it is much simpler to understand the graphical properties of a stroke. Thus, using only the graphical representation removed a potentially confounding factor from the study.

In the first round of user studies, we investigated twelve different algorithms. These algorithms were suggested by previous research and the focus group. The participants were asked to read and annotate a short section of text, as prior research indicates users react better to modifications of their own annotations [4]. After reading and annotating was finished, the underlying text was changed and the participants evaluated the results of each algorithm.

In the second study, we investigated fourteen different algorithms. These included preferred algorithms from the first round and new algorithms based on user feedback. This round used the same protocol as the first round.

The hypotheses tested were:

1) People would want their annotations automatically refitted so the meaning is preserved.
2) There would be specific algorithms that best preserve meaning for each category of annotations.
3) While participants would prefer different algorithms there would be a small subset preferred overall.

TABLE I
Categories of annotation types.

Category	Annotation Types
Horizontal Line	Underlines, highlights and scratch-outs
Vertical Line	Margin bars and braces
Enclosures	Circular and rectangular bounding boxes
Connectors	Call-outs and arrows

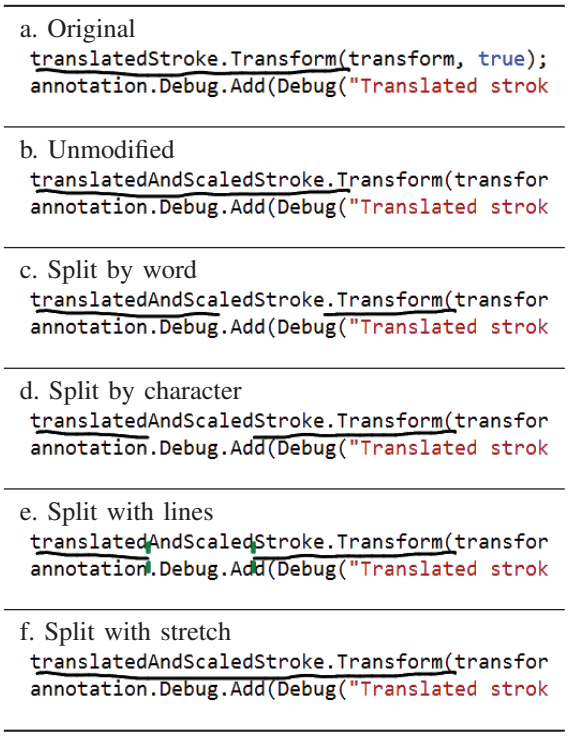

Fig. 2. Examples of Horizontal Line Refitting.

IV. Implementation

Following the strategy used by previous studies, we grouped annotation types into four categories. We hypothesised users would prefer the same adaptation process for each annotation type in a category. Table I shows our categories, based on prior research [4], [6].

We implemented the algorithms in a plug-in for *Visual Studio 2013* called *vsInk* [13] (see Figure 1). To facilitate the investigation we extended the software in three ways. First, the software now stores a read-only copy of the original annotation. All adaptations clone the original and modify the clone rather than the original. Otherwise, as pointed out previously [6], the annotation is corrupted over time. Second, we added a simple interface that allowed changing the adaptation process for each category. Finally, when the adaptation process is changed the display is immediately refreshed. This allows the user to quickly see the difference each algorithm makes to the annotation.

For this study we used a recogniser implemented using RATA.Gesture [14]. RATA.Gesture provides a toolset for generating ink classifiers using machine learning. The recogniser for this study identified horizontal lines (highlights and underlines), vertical lines (braces and margin bars), enclosures (circles), connectors and text. While text annotations were anchored in the document we did not attempt any refitting. We provided the ability to reclassify strokes if the classifier did not correctly classify them.

A. Horizontal lines

We implemented five ways of refitting horizontal line annotations (see Figure 2). The first three ways are based on where the annotation is split. In 'Unmodified' the annotation is not split at all (see Figure 2b.). The annotation is associated with the first underlying character. In 'Word Boundary' the annotation is segmented against each word (see Figure 2c.). In 'Character Boundary' the annotation is segmented against each character (see Figure 2d.). The annotations are then split based on where characters are added between the boundaries; with the annotation staying relative to the first associated character. If the text is split between the segments the annotation will stay with the previous anchor. Previously the only algorithm for refitting horizontal line annotations was to split them using word boundaries [4], [6].

The remaining two algorithms use character boundary splitting as the starting point. For 'Split with lines', vertical dashed lines are added at the split locations (see Figure 2e.) This dashed line is semi-transparent and extends above and below the annotation for half the character height. The final algorithm, 'Split with stretch', stretches the previous segment so it fills the gap (see Figure 2f.). This gives the illusion that the annotation itself has been stretched to fill the gap.

B. Vertical lines and enclosures

We implemented eight algorithms for refitting vertical line and circle annotations. The first two have been implemented previously and the remaining six are new. Figure 3 shows these algorithms. We grouped both vertical lines and enclosures together as both categories span multiple lines and we hypothesized that the users would prefer similar algorithms. Figure 3a. shows the original unmodified annotation and the underlying context.

In 'Unmodified' the annotation was left as it was originally (see Figure 3b.). This is the algorithm used in the majority of implementations (e.g. [13]) but no one has reported on the desirability of this algorithm.

In 'Whole stretch' the entire annotation is scaled (see Figure 3c.). This is the algorithm used in *XLibris* [6] and *Callisto* [4]. In this algorithm the annotation is associated with the lines at the top and bottom of the annotation. The annotation is scaled vertically using the distance between the top and bottom of the annotation.

In 'Simple split' the annotation is split into segments without any indicators (see Figure 3d.). The annotation is segmented so each segment is associated with a single line of text. When a new line is added the annotation is split and the gap is left unfilled.

In 'Split with vertical line' the annotation is split into segments with a vertical line between segments (see Figure 3e.). This line is dashed and semi-transparent so it is different from the rest of the annotation. The line is always vertical: it directly joins the last point of one section with the first point of the next section.

In 'Split with horizontal line' the annotation is split with a horizontal line added (see Figure 3f.). For enclosures, the line

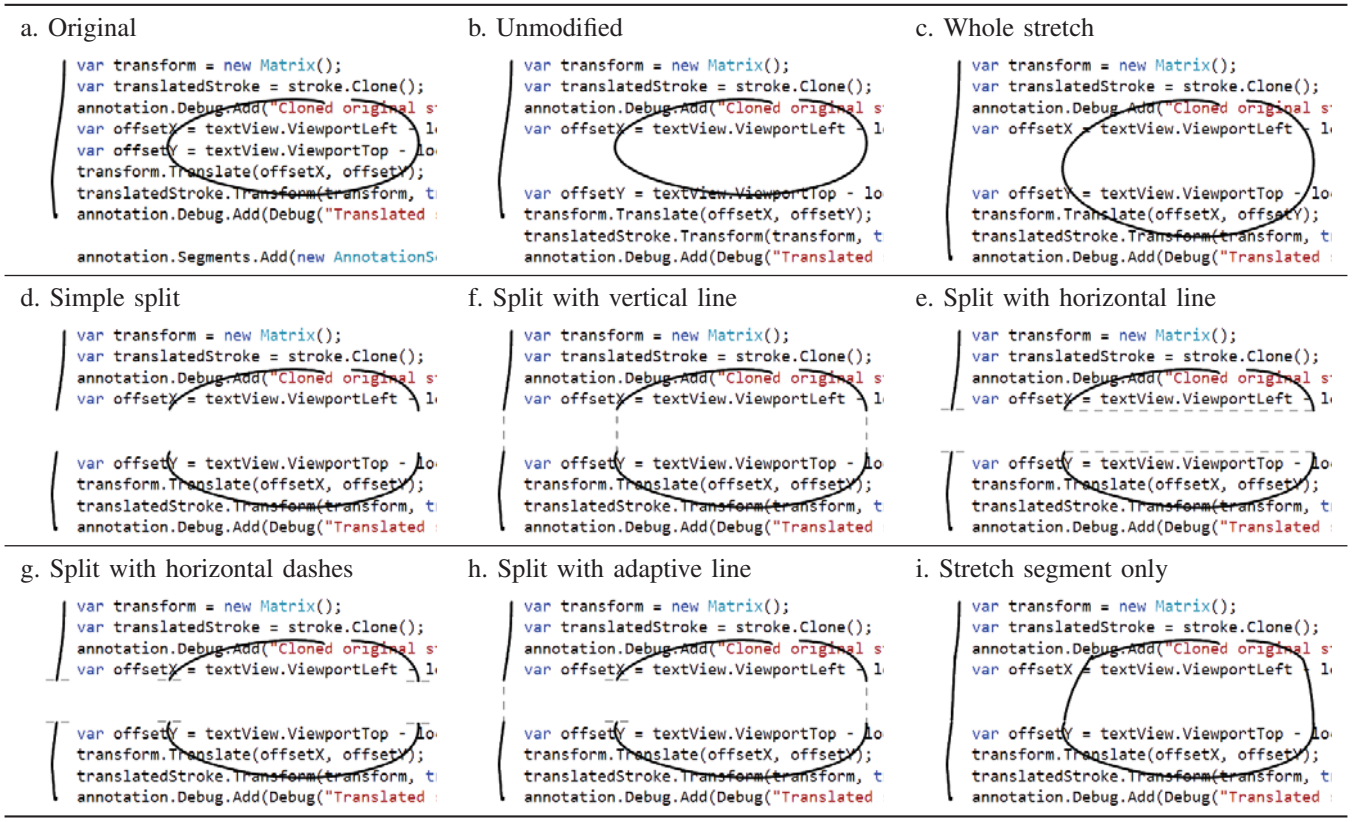

Fig. 3. Examples of Vertical Line and Enclosure Refitting.

goes from the outer-most point on the left to the outer-most point on the right. For vertical lines, the line extends to both sides of the line by the equivalent of two em spaces. For both variations the line is dashed and semi-transparent.

'Split with horizontal dashes' is similar to the previous algorithm with both circles and vertical lines having a short line extending to the sides of the stroke (see Figure 3g.).

'Split with adaptative line' combines 'Split with vertical line' and 'Split with horizontal line' (see Figure 3h.). If the segments surrounding the split is nearly vertical a vertical joining line is added; otherwise horizontal lines are added.

In 'Stretch segment only' the annotation is split into two segments and the top segment scaled to fill the gap (see Figure 3i.). Unlike the whole annotation stretch, only the segment above the gap is scaled.

For all split algorithms the annotation was segmented based on lines. When the stroke crossed a line it was segmented into two. To ensure the annotation looked smooth an extra point is extrapolated at the line boundary. This point becomes the final point of the old segment and the first point of the new segment.

C. Connectors

We implemented four algorithms for refitting connectors: none have been investigated previously (see Figure 4). In *vsInk* adding a connector annotation automatically adds an associated note. This note is styled as a yellow post-it note. Any strokes added within this note are automatically grouped together as a complex annotation (e.g. text or drawing). Annotations have two anchor points: one at the start and one at end of the stroke. The primary anchor point is the start point.

In 'Unmodified' the annotation is unchanged by any context changes (see Figure 4b.). This causes the annotation to move together with the primary anchor point. When the associated note is anchored to a different line, the two annotations become separated as space is added between their anchor points.

In 'Whole stretch' the annotation is stretched using only the Y coordinates of the start and end points (see Figure 4c.). The stretch amount is the ratio of the old vertical distance to the new vertical distance. The annotation is only stretched vertically which can result in some distortion.

In 'Points stretch' the annotation is stretched using both the X and Y coordinates of the start and end points (see Figure 4d.). The stretch amount is the ratio of the old Euclidean distance to the new Euclidean distance. This tends to result in a smoother scaling of the line.

The final algorithm, 'Dynamic stretch' switches between 'Unmodified' and 'Points stretch' depending on where the end point is relative to the associated note. When the end point is within the note boundary the annotation is unmodified. When the end point moves beyond the boundary of the note the end point 'sticks' to the top, or bottom, of the note and 'Points stretch' is used.

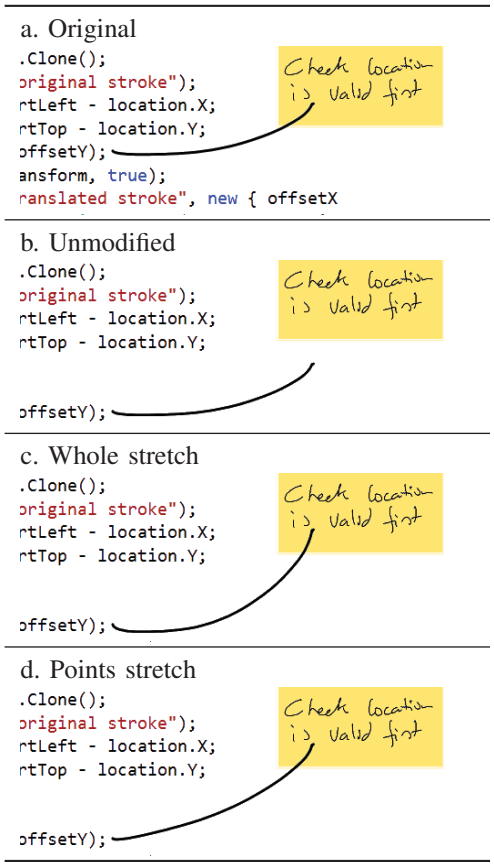

Fig. 4. Examples of Connector Refitting.

V. User Study

To evaluate people's reactions we performed two rounds of a user study. We were interested in what people would think of the different adaptations and which they would prefer for each annotation category. The two studies followed the same protocol, based on the study by Bargeron and Moscovich [4]. The only difference between the two studies was the algorithms used.

Study One evaluated the following approaches:

1) **Single line**: unmodified, split by word, split by character;
2) **Vertical lines**; unmodified, whole stretch, simple split, split with vertical line, split with horizontal line, segment stretch;
3) **Circles**: unmodified, whole stretch, simple split, split with vertical line, split with horizontal line, segment stretch;
4) **Connectors**: unmodified, vertical stretch, points stretch.

Study Two evaluated the following approaches:

1) **Single line**: unmodified, split by word, split with lines, split with stretch;
2) **Vertical lines**; whole stretch, split with vertical line, segment stretch;
3) **Circles**: unmodified, whole stretch, split with vertical line, split with horizontal dashes, segment stretch;
4) **Connectors**: unmodified, vertical stretch, points stretch, dynamic stretch.

A Microsoft Surface tablet was used for both rounds of testing. The reading was all done in *Visual Studio 2013* [15] using the modified *vsInk* extension [13]. *Morae* [16] was used to record the screen to allow for analysis after the testing.

A. Procedure

Each participant performed the task in a controlled environment. Prior to the task each participant was welcomed, had the process explained to them (including gaining consent) and filled in a short questionnaire on their reading and annotating background.

After the questionnaire was completed, the researcher demonstrated *vsInk*. The participants were then given 5 minutes to familiarize themselves with the software and ask any questions. If desired, the participant could finish this part early. The participants were then given 15 minutes to read a file of C# code. The instructions were to read and annotate the so they could explain it to another programmer. They were specifically instructed not to change the code. During the reading time the researcher sat next to the participant and observed them.

After the reading task, the researcher selected several annotations of each category. If there were insufficient annotations in a category the participant was asked to add some more. The researcher modified the text underneath each annotation by adding additional text or newlines as needed. The participant was then shown each way the annotation could be refitted. The participant then rated each adaptation using a five point Likert scale on the effectiveness of the algorithm in preserving the meaning: one was strongly agree through to five being strongly disagree. Then they were asked to rank the algorithms in order from most preferred to least preferred.

After reviewing the annotations, the participant was asked for any recommendations on how annotations could be refitted. This then concluded the user study.

All the participants in this study were volunteers. They were not given any remuneration (financial or otherwise) for their time.

B. Analysis

After each participant, the questionnaire, ratings and ranking data was added into a database. The data was entered in an anonymous format so the participants could not be identified. The data was analysed using R 3.2.2 [17], [18]. A combination of graphs and inferential statistics were used.

VI. Results

The adaptations were evaluated over two rounds of the study with different participants in each round. The protocol and task was the same for both rounds: the only difference was the algorithms evaluated.

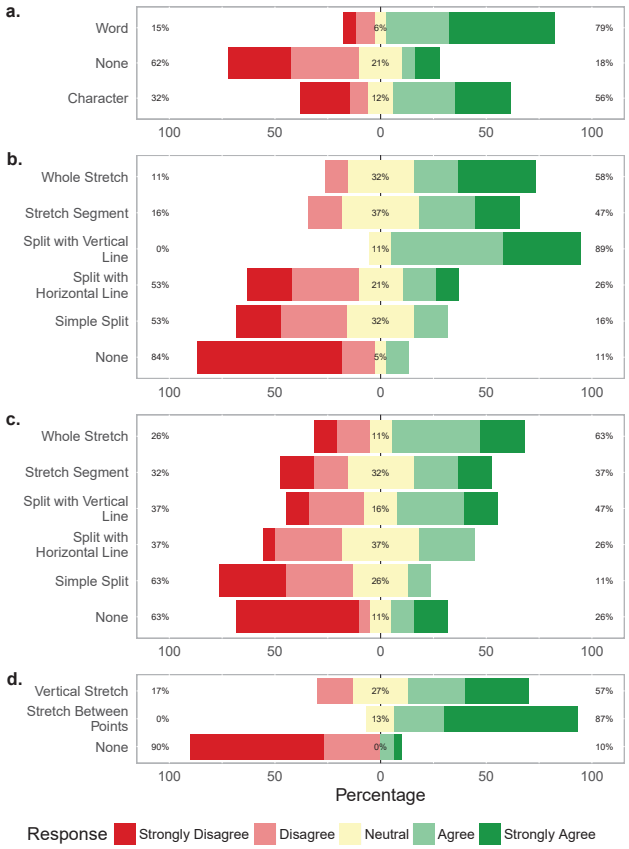

Fig. 5. Participants' rating and ranking for each adaptation algorithm in round one. *a.* horizontal lines; *b.* vertical lines; *c.* circles; *d.* connectors. The percentage is the number of responses for each rating.

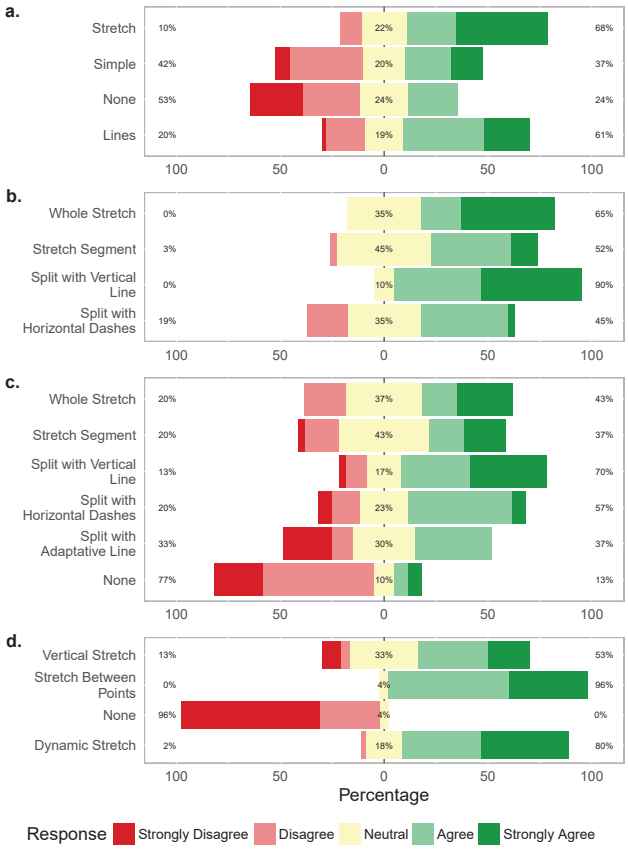

Fig. 6. Participants' rating and ranking for each adaptation algorithm int round two. *a.* horizontal lines; *b.* vertical lines; *c.* circles; *d.* connectors. The percentage is the number of responses for each rating.

A. Round One

In the first round there were seven participants. Three were professional developers and four were graduate students. All participants were familiar with reading program code and could understand the code provided. All participants reported reading code on a regular basis (at least daily). All of the participants were used to annotating while reading; although only four reported annotating program code.

There were 102 annotations collected in round one (34 single lines, 19 vertical lines, 19 circles and 30 connectors). Figure 5 shows the results from the first round.

Both the word and character level algorithms were generally liked (79% and 56%), while doing nothing was disliked (62%). This matches the preferences: segmenting by word was most preferred and doing nothing least preferred. However there were some circumstances where these were reversed. Users preferred the line to stay with the associated syntactical unit but if it did not, then the users preferred that the annotation was not modified.

For vertical lines, doing nothing was disliked (84%) and never chosen as the first preference. This is because vertical lines are context specific, with both the top and bottom position of each line being significant. Splitting the annotation without any visualisation was also disliked (53%). Without any visualisation annotations appeared to be split into two (or more) separate annotations.

There was less agreement about how the annotation should be refitted. Split with vertical line was most liked (89%), followed by stretching the whole annotation (58%) and then by stretching the segment (47%). Split with vertical line was most preferred; followed by stretching the whole annotation.

While circle annotations use the same adaptation routines as vertical line, the results are different and less consistent. One notable difference is participants also liked the unmodified algorithm because, in some situations, the circle is a form of single line annotation. Therefore they expected them to be refitted in a similar way to single line annotations. A second reason for the differences is the change location relative to the annotation makes a difference. If the surrounding segments of the annotation were almost vertical, then a vertical line was preferred; otherwise stretching the whole annotation was preferred.

All participants stated that connector annotations should be adapted as the connector should stay with the associated note. Participants liked the unmodified algorithm until the note detached. At this point the participants stated the meaning was lost. For the remaining two algorithms the results were very similar, as in the majority of scenarios the results of two

algorithms looked very similar. Stretching between the points were slightly more preferred as occasionally the vertical only stretch deformed the annotation. This typically occurred when the annotation was not a straight line (for example when an arrowhead was included on the connector).

B. Round Two

During the first round, participants made numerous suggestions which lead to changes in the algorithms. These were evaluated in a second round of testing with ten participants. All were professional developers. All participants reported reading code on a regular basis. Only three of the participants reported annotating program code.

There were 165 annotations collected in this round (59 horizontal lines, 31 vertical lines, 30 circles and 45 connectors). Figure 6 shows the results from this study.

For horizontal line annotations doing nothing was again least liked (53%): none of the participants ever choose it as their preferred algorithm. The few times when it was rated as a two is because the annotation was either long enough or the change small enough the annotation remained underneath the relevant words. Talking with the participants revealed the only time doing nothing was acceptable was when the relevant words were still underneath. Simple split was less preferred in this round as it does not show when an annotation has been modified.

Finally, there is an even liking between the other two algorithms (68% for stretching and 61% for lines). The participants can be divided into two groups: those who liked to see the annotations had been adapted and those who preferred the annotations stayed natural looking. The main reason mentioned for preferring the stretch algorithm is the annotations look more natural and matches the original annotation. In contrast, the main reason for liking the dashed line visualisation is it shows that the annotation has been split. One participant mentioned that without the dashed lines he would have had no idea that the annotation had been split.

For vertical lines there is also an even split between stretching and adding dashed lines, although split with vertical lines was more liked (90%). Vertical dashed connecting lines were preferred over horizontal dashes as they showed the segments of the annotation belonged together. The participants often stated the two stretch algorithms were similar and it was hard to tell them apart. For some vertical lines (e.g. braces) it was more obvious that only part of the annotation had been stretched. In this case the majority of the participants preferred the whole stretch as it maintained the natural look of the annotation.

Again, the preferences for circle adaptations are less obvious for similar reasons to those reported in round one. In this study, some participants still preferred no adaptation in certain scenarios (when the circle is associated with a single line); otherwise it was disliked (77%). However if the circle is a multiple line annotation then it should be modified as the top and bottom positions have significance.

A new algorithm in this round was the adaptive algorithm. In this approach the annotation refitted based on the surrounding segments. This algorithm was generally disliked (33%). According to the participants the main problem is the results were unpredictable. One participant stated this was "confusing with a vertical line on one side and horizontal on the other." Likewise, stretch segment was also less liked, although less so than the adaptive algorithm. This algorithm would sometimes distort the annotation, causing it to lose its natural shape.

The results for the remaining three algorithms (whole stretch, split with vertical line and split with horizontal dashes) are roughly similar. All three algorithms were ranked as the number one algorithm multiple times and were more liked than disliked (70% for split with vertical lines, 57% for split with horizontal lines and 43% for whole stretch). Again, there were some participants who preferred the natural appearance and others who preferred to see that the annotation had split. Horizontal dashes and vertical lines were equally liked.

One trend between categories is the participants often preferred the same 'style' of refitting. If the participant preferred horizontal lines stretched, then they would also prefer vertical lines and circles stretched. The same pattern also applies for adding a indicator. Based on this we were able to categorise participants as belonging to one of two groups: those who preferred stretching (natural appearance) and those who preferring an indicator that the context had changed.

Again, all participants agreed that connector annotations should be refitted. The preferred algorithm was the new dynamic algorithm as it provided two benefits. First, it stayed in the original form when relevant and only refitted as necessary. Second, it did adapt when necessary in a natural appearing way. Similar to the first round, participants once again found very few differences between the two stretch algorithms. One participant also suggested that "the note annotation could be considered part of the connector annotation." That is, there should only be one anchor for the combined connector and note, with the note staying fixed to this anchor.

VII. Discussion

This study set out to determine if and how freeform annotations should be refitted.

A. Hypothesis One

Our first hypothesis was participants would prefer their annotations refitted. For the majority of annotations this was true but there was a consistent set of exceptions. First, the underlying context needed to change enough for the refitting to be meaningful, otherwise the participants saw no value in refitting. A common example, was underlines where the underline extended beyond the end of the line. Second, some circle annotations should be treated similar to horizontal line annotations. While we posit these circles should adapt in a similar way to horizontal lines we did not investigate this.

The main reason why participants wanted their annotations to refit is the meaning is often related to the annotation's context. For example, underlines only have meaning when

associated with the underlined words. If the annotation moved away from those words then it would become either meaningless or confusing (as one participant stated 'I would wonder why I had only underlined that part not the whole line').

We did not attempt to refit complex annotations (text and drawings). These annotations have explicit, or semantic, meaning that would need preserving [19]. None of the algorithms described in this paper would preserve their meaning. Any refitting algorithm would need knowledge of the semantics in order to preserve the meaning.

B. Hypothesis Two

Our second hypothesis was there would be specific algorithms that best preserve the meaning for each category. To test this, the annotation types were grouped into four categories based on previous studies [4], [6]. We original categorised lines into vertical or horizontal but our results indict they could be a single category. Further work needs to be done to verify this. One difference between the two lines is what should happen when horizontal line annotations wrap to a different text line.

One disagreement in previous studies was how to categorise circles. *Callisto* treated circles as underlines [4] while *XLibris* treated them as vertical lines [6]. Our results show there are two categories of circles. Short circles around one or more words on the same line are similar to horizontal lines. Taller circles that span multiple lines are in their own category with different algorithm preferences from either vertical and horizontal lines.

The results of this study suggest some general recommendations for each category. All refitted annotations should be recognisable as a single annotation. When splitting an annotation, some visualisation must be added to show the annotation has split. Stretching an annotation is acceptable as long as the annotation is not distorted.

For connectors, the associated annotation must remain linked to the connector. We refitted the connector in our studies; an alternate approach would be to treat the connector and associated annotations as one single annotation. This would simplify adaptation as the annotation would only need to be repositioned relative to the anchor.

In our study. connectors were the only annotations we investigated that had an associated annotation, but we did observe other examples of where annotations should be linked. One common example is a vertical line (emphasizing a range of lines) followed by a connector. These should also be linked and the association maintained. We did not investigate these linked annotations as this involves challenges of how to group together annotations [3].

An additional issue identified in the study is text avoidance for connectors. One participant pointed out they add connectors in the whitespace and they would expect any refitting to respect this. While text avoidance has not been investigated for annotations, a similar challenge has been investigated when manipulating connectors on graph diagrams [9].

C. Hypothesis Three

Our third hypothesis was there would be a small subset of preferred algorithms for all categories. The results of this study is there is not one but two preferred subsets. One group of participants preferred algorithms that kept the original look and feel: we labelled this group as 'stretched'. The second group prefers to see their annotations were refitted. This group we labelled 'split' as the preferred algorithms indicate the underlying text was changed, with the visualisation showing where the change was.

These results suggest two modes for refitting: stretching and splitting. In stretching, annotations are refitted by stretching all or part of the annotation. In splitting mode the annotation is split and a visualisation added. Any implementations should allow the user the choice of mode.

D. Limitations

A limitation of this study is we only investigated refitting annotations on program code. Initially the rationale behind this decision was the availability of a tool (*vsInk*) that we could modify. In addition, code itself is very different from other document types, however the types of annotations on program code are very similar [20].

VIII. Conclusions and Future Work

In this investigation, the aim was to explore how users expected their annotations to be refitted. We found users expect their annotations to refit and the preferred refitting algorithm depends on two factors: the category of annotation and the user's general preference.

As suggested previously [4], [6], the category of annotation is important. In this paper we explored four categories: horizontal lines, vertical lines, circles and connectors. There are different preferences for each category. In addition, we found circles should be two separate categories, depending on whether they are associated with one line or multiple.

One unanticipated result is there are two different groups of users. One group prefers stretched annotations while the other prefers splitting with an added visualisation.

The results from this study suggest the following preferences:

1) Line annotations (horizontal and vertical) should be either segment stretched or split with a visualisation;
2) Multiple line circle annotations should be either whole stretched or split with short perpendicular lines;
3) Connectors should be left as is when the associated note is connected; otherwise they should be stretched.

In addition, there appear to be similar characteristics between single line circles and horizontal lines. However we did not investigate whether using the algorithms for refitting horizontal lines would have similar results when used on single line circles.

Further work might explore the following areas: how single line circles should be refitted; how to reflow connectors around text; whether these results hold for other, non-code, types of document.

REFERENCES

[1] P. Mueller and D. Oppenheimer, "The pen is mightier than the keyboard: Advantages of longhand over laptop note taking," *Psychological Science*, vol. 25, no. 6, pp. 1159–1168, 2014.

[2] K. O'Hara and A. Sellen, "A comparison of reading paper and on-line documents," in *Proc. of the SIGCHI Conf. on Hum. Factors in Comput. Sys.*, ser. CHI '97. New York, NY, USA: ACM, 1997, pp. 335–342.

[3] C. Sutherland, A. Luxton-Reilly, and B. Plimmer, "Freeform digital ink annotations in electronic documents: A systematic mapping study," *Computers & Graphics*, vol. 55, pp. 1–20, 2016.

[4] D. Bargeron and T. Moscovich, "Reflowing digital ink annotations," in *Proc. of the SIGCHI Conf. on Hum. Factors in Comput. Sys.*, ser. CHI '03. New York, NY, USA: ACM, 2003, pp. 385–393.

[5] K. Conroy, D. Levin, and F. Guimbretière, "Proofrite: A paper-augmented word processor," in *Demo Session of UIST*. ACM, 2004.

[6] G. Golovchinsky and L. Denoue, "Moving markup: Repositioning freeform annotations," in *Proc. of the 15th Ann. ACM Symp. on User Interface Softw. and Technol.*, ser. UIST '02. New York, NY, USA: ACM, 2002, pp. 21–30.

[7] S. Ramachandran and R. Kashi, "An architecture for ink annotations on web documents," in *Proc. of the 7th Int. Conf. on Doc. Analysis and Recognition*, 2003, pp. 256–260 vol.1.

[8] X. Wang, M. Shilman, and S. Raghupathy, "Parsing ink annotations on heterogeneous documents," in *Eurographics Workshop on Sketch-Based Interfaces and Modeling*. Eurographics Association, 2006, pp. 43–50.

[9] B. Plimmer, H. Purchase, and L. Laycock, "Preserving the hand-drawn appearance of graphs," *Visual Languages and Computing*, pp. 347–352, 2009.

[10] P. Reid, F. Hallett-Hook, B. Plimmer, and H. Purchase, "Applying layout algorithms to hand-drawn graphs," in *Proceedings of the 19th Australasian conference on Computer-Human Interaction: Entertaining User Interfaces*. New York, NY, USA: ACM, 2007, pp. 203–206.

[11] J. Arvo and K. Novins, "Appearance-preserving manipulation of hand-drawn graphs," in *Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia*. New York, NY, USA: ACM, 2005, pp. 61–68.

[12] G. Johnson, M. Gross, J. Hong, and E. Do, "Computational support for sketching in design: a review," *Foundations and Trends in Human-Computer Interaction*, vol. 2, no. 1, pp. 1–93, 2009.

[13] C. Sutherland and B. Plimmer, "vsInk: Integrating digital ink with program code in visual studio," in *Proceedings of the Fourteenth Australasian User Interface Conference - Volume 139*, ser. AUIC '13. Darlinghurst, Australia, Australia: Australian Computer Society, Inc., 2013, pp. 13–22.

[14] S. Chang, R. Blagojevic, and B. Plimmer, "RATA.Gesture: A gesture recognizer developed using data mining," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, vol. 26, no. 03, pp. 351–366, 2012.

[15] Microsoft Corporation, "Visual Studio 2013." [Online]. Available: https://visualstudio.com

[16] TechSmith Corporation, "Morae." [Online]. Available: https://www.techsmith.com/morae.html

[17] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2015. [Online]. Available: https://www.R-project.org/

[18] J. Bryer and K. Speerschneider, *likert: Functions to Analyze and Visualize Likert Type Items*, 2015, r package version 1.3.1. [Online]. Available: http://CRAN.R-project.org/package=likert

[19] C. Marshall, "Annotation: From paper books to the digital library," in *Proc. of the 2nd ACM Int. Conf. on Digit. Libr.*, ser. DL '97. New York, NY, USA: ACM, 1997, pp. 131–140.

[20] C. Sutherland, A. Luxton-Reilly, and B. Plimmer, "An observational study of how experienced programmers annotate program code," in *Human-Computer Interaction INTERACT 2015*, ser. Lect. Notes in Comput. Sci., J. Abascal, S. Barbosa, M. Fetter, T. Gross, P. Palanque, and M. Winckler, Eds. Springer International Publishing, 2015, vol. 9297, pp. 177–194.