RCA: Experiences with an IDE Annotation Tool

Richard Priest and Beryl Plimmer
Department of Computer Science
University of Auckland, Private Bag 92019
Auckland, New Zealand
rpri032@ec.auckland.ac.nz, beryl@cs.auckland.ac.nz

ABSTRACT
Ink annotation is a common method for recording feedback on a paper document. However, reviewing code on paper is difficult due to its non-linear nature. This project extends existing research ideas to develop a digital ink annotation tool within an Integrated Development Environment (IDE). The aim is to provide code reviewers with an effective tool for directly commenting on code within the IDE. We describe scenarios where ink annotation would provide benefits, along with requirements and our implementation of the Rich Code Annotation Tool (RCA).

Author Keywords
Ink Annotation, code review support, pen-base interaction

ACM Classification Keywords
H5.2. Information interfaces and presentation: User Interfaces. - Graphical user interfaces

INTRODUCTION
Annotating documents with a pen is a natural way to record comments and emphasize parts of the document. Digital ink annotation is emerging as a way to support annotation over digital documents. Existing research suggests people enjoy the added functionality that a digital ink environment provides [18, 19, 23]. We are interested in whether this success extends to annotating program code within an Integrated Development Environment (IDE). There are major technical challenges to overcome when developing annotation tools including re-flowing the digital ink when the underlying document changes [5, 10, 16, 22] and the tangibility factors associated with traditional annotation [25].

To evaluate whether ink annotation is a useful tool when used within an IDE, we developed the 'RichCodeAnnotation' tool (RCA) which makes it possible to use digital ink annotation in the code editor of an IDE. This approach to reviewing code offers the advantages that are consistent with typical code review processes, as explained by Fagan [8]. However this tool offers additional advantages. As the tool is integrated into an IDE, we retain all the benefits that the underlying code editor has to offer such as object-definition searches, font properties, debugging and execution of code and existing wizards. Code is non-linear, it is arranged in logical classes and procedures that are not intended to be read sequentially like a book. Therefore simply printing out code and annotating with a red pen makes the review process more difficult as the IDE navigation support is not available. Text comments can be added to code however digital ink stands out from the underlying document making the annotations easy to distinguish from the original code. As we want both inking and IDE support, digital ink is an appropriate approach.

This tool provides benefits in five distinct code reviewing scenarios. First, in a classroom the teacher can explain code in an IDE and annotate it to emphasize particular points. After the lecture, the teacher can make the annotated code available to the students. Second, markers can add comments to students' assignments to indicate what code works correctly, where they went wrong and give an overall mark. Third, commercial software development teams can use ink annotation during formal inspection meetings. For example a group of developers and testers can discuss a piece of code and use colored ink annotations to distinctively indicate different types of coding errors e.g. security, logic, efficiency and syntax errors. These issues when marked directly on the code with digital ink can be sent back to the developer for adjustment and also form part of the software development artifact. Fourth, this tool can be used by developers to comment their own code, much as they would with keyboard text. Commenting using ink annotation allows the developer to work in a relaxed setting e.g. on a couch, a beach or a plane. Finally this tool can support developers conducting self or peer-reviews of their implementation. The code could be advertised on an online forum where friends, mentors and coding fanatics illustrate bugs with red digital ink much like traditional markup on paper essays.

Chinz'06, July 6–7, 2006, Christchurch, New Zealand.

In the next section we present a summary of related work. These, together with the scenarios above, are used to define a set of requirements for a code review annotation tool. The implementation section describes our realization of these requirements as RCA and a brief evaluation is given. We conclude with a discussion and the future plans for this work.

RELATED WORK

One of the first tools developed to explore digital ink annotation was Wang Freestyle [9]. It provides the user with simple free-form ink annotation over a static page, without computation. XLibris [25] was developed to offer users an active reading experience, with a main goal of overcoming the tangibility challenges of reading online documents. It provides users with an interface and features similar to that of paper. Again this system only deals with static documents, though it has been extended [10] to support reflowing and reshaping of digital ink when the underlying layout changes. Margin bars, circles and underlines stretch or shrink when the underlying layout changes through font resizing, zooming or varying device characteristics. Annotations also reshape when underlying text splits over line breaks and page breaks.

With the experiences researchers gained from the reflowing extension of XLibris and the reflowing support within Microsoft Word [17], PREP editor [21] and DIANE [4], methods were investigated to support robust positioning that handles text changes in the underlying document [2, 5, 24]. One successful idea is based on grouping annotation strokes according to the spatial and temporal relationships between the strokes and then anchoring the annotation to a portion of the text or a line.

A web annotation tool [24] successfully reflows digital ink annotations on dynamic web pages. It uses W3C's document object model DOM [27] to map digital ink to text and images. The tool also supports ink gestures which are specific ink strokes used to invoke functions (such as copy).

Ink annotation of code has been used successfully in a Computer Science course at the University of Alaska, where code was prepared before class then annotated in front of students during class time, thus replacing the traditional blackboard [18]. These notes could then be placed on the course web page for future reference. However the annotations are not directly available in the IDE. A similar system links a Tablet PC wirelessly to a data projector, allowing the lecturer to walk around the room while annotating [1]. There have also been several studies where students use Tablet PCs to annotate the lecture material collaboratively, sharing their annotations with others in the class [12, 26]. This gives students several different perspectives on the material from other students. These annotations can then be stored and used for revision in the future.

With the increase in online assignment submissions across many universities, Penmarked [23] was designed to be used by academic staff to mark digital documents. Once students have submitted their assignments, the marking system allows for digital ink annotations to be made directly on the student's submission. The system also handles multiple files within an assignment, displaying each in a tabbed window. This system also contains a marking schedule that is attached to each student's submission. The marking schedule consists of marking criteria each with a maximum and minimum value, and an input panel for the marker to allocate marks. The Penmarked system only deals with static documents, yet this is acceptable because assignments are finished documents and won't be modified. As the system opens the files as text documents, it does not provide the support offered within an IDE. Most importantly it doesn't allow the assignment to be directly compiled and run, it also doesn't include the coloring scheme found in most IDEs for color coding of keywords. Marktool [11] provides a similar approach but uses drag-and-drop shapes and text boxes for the annotation. Gild [20] provides similar marking functionality with in the Eclipse IDE but does not support digital ink annotation.

In industry technical review procedures [13], developers submit code to their project managers, and then a team of reviewers is organized. Each team member first conducts an individual private review of the code. Once all private reviews are completed, all of the issues raised by all of the participants are consolidated and made public. Participants next review the issues raised by others. Once the issues have been understood a group meeting is held where the reviewers discuss all issues and decide on each issue's severity. After the group meeting a review report is created; it lists all issues identified during the review process along with their locations, severity and error type. Finally this report is given to the developer to resolve the issues. Research has shown this significantly reduces errors and reduces the cost of resolving future errors. In fact in some cases this procedure worked so well that traditional testing was made obsolete [14]. However with most reviews of this kind, the code is printed out and distributed, or a copy of the program is distributed, and participants list issues on paper or in a text editor. Consolidating individual private issues into a distinct list of public issues is done manually by the project manager. The creation of the review meeting report is also done manually. Hence the process can be seen as time consuming: this is one of the main reasons given for not performing a formal review process.

The existing annotation tools and code review scenarios above provide evidence that an ink annotation tool within the IDE may prove useful for highlighting important points. Annotating in the IDE preserves existing functionality that helps the reviewer interpret the code. In particular the IDE assists navigation through code in a non-linear manner and

makes reading the code easier with the keyword coloring schemes.

REQUIREMENTS

To develop an ink annotation tool within an IDE we formulated a set of basic requirements. Our prototype will need to incorporate these for the tool to be successful in different code review settings. This section presents an outline of the requirements and their associated benefits. Principally: the IDE must be extensible; the ink must be freeform, preserved and modifiable; each set of ink strokes must be grouped such that the ink can be reflowed; ink groups must also include an indicator representing the severity of each issue; finally this tool must work seamlessly with the IDE.

Extending the IDE

To incorporate ink annotations the chosen IDE needs a way in which it can be extended. Some IDEs are open-source, for example Eclipse [7], where the code can be downloaded by anyone and edited to add the required functionality. Other IDEs allow the developer to create 'Add-ins', for example .NET [3] where the application allows the user to write supplementary code that is incorporated when the IDE starts up.

Free-form Ink Annotation

Ink annotation is free-form and may consist of words, symbols and text selection marks. Such markings may be made anywhere; without limitation on shape or content [25]. This is usually achieved by attaching a transparent ink overlay over a window. As the underlying window scrolls, so to does the transparent window that holds the digital ink. This gives the illusion that the digital ink is attached to the underlying text.

Integrating into the IDE

The code annotator must integrate seamlessly into the IDE so that it does not add an overhead to the review task. Integrating the ink file directly into the project allows the IDE to perform as normal while supporting code annotation.

Persistence of the ink is an important function the tool needs to handle. If a user selects a source file to ink over and no ink file exists, then a new ink file needs to be automatically created and stored as part of the development project. Once a reviewer has finished annotating the code, the ink needs to be saved, allowing the developer or others to view the annotations at a later date. The tool must also load ink files directly into the project. Whenever the IDE loads a code file, a search for its corresponding ink file would follow.

Modifying Digital Ink

With traditional pen and paper, once the user has marked on the script with red ink, it is difficult to alter this ink. Modifying existing ink involves concealing or crossing out the ink then rewriting the annotation, which looks messy. Digital ink offers a computerized approach allowing users to efficiently and cleanly erase, select, move and recolor ink.

Reflowing Digital Ink

Code within an IDE is not fixed; therefore when the code is altered the existing digital ink must also be reflowed to remain consistent with its underlying context. In order to reflow digital ink, each set of strokes must be grouped to form an annotation issue and each annotation linked to a specific part of the document. When the underlying code moves up/down because of insertions or deletions to the code, then the corresponding annotations must also be moved. Grouping strokes is most successful if both the location properties and the time between strokes are considered [2, 28].

When reflowing code we need only consider moving digital ink up or down: it is rarely necessary to alter the ink horizontally for two reasons. First, each code line involves a strict structure; this means a code line can not be re-arranged like a sentence. So once a code line is written, if it is substantially re-written the line would probably look rather different and the digital ink context is lost. Second, minor amendments may result in the digital ink being a little offset from its original code context. However research has shown [2] slightly offset ink annotations are not a significant problem. For these reasons, reflowing efforts in this tool concentrate on vertical repositioning of the digital ink.

Issue Severity

Assigning a severity rating to an issue during a technical review meeting is a common practice [8, 13]. This is also a convenient feature to use during a self or peer-review. When an ink annotation is made, it signifies there is an issue with that portion of code; an issue could be a negative statement that implies a defect or positive feedback. When reviewers locate a defect, they may want to attach a severity rating to the issue, in terms of low, moderate or high. The severity indicators represent to the developer the seriousness of each issue, which can be used to decide what issues need most attention. The indicator could also be used to tick off resolved issues.

This severity scheme may also be useful for assignment markers in an educational setting, where the severity of defect correlates to the number of marks deducted. The marker could use the 'resolved' severity to indicate positive feedback for certain portions of code. In a teaching environment, the indicators could be used to denote the importance or relevance of certain lines of code.

Multiple Code Windows

Most software programs involve multiple source files. IDE frameworks accommodate this by providing a separate tabbed window for each code file, allowing developers to iterate easily through each individual file. An annotation tool incorporated into an IDE must also manage users' ink

annotations over these multiple files. This must be efficient as users frequently switch between different files to follow the path of execution.

IMPLEMENTATION

We decided to developed RCA within the Visual Studio .NET 2005 IDE (VS) as a plug-in, because VS supports a wide range of different languages. The languages supported include C++, C# and VB, which are all taught in our computer science department.

RCA was designed for use on tablet PCs; however it can also be used on desktop computers with the aid of a tablet USB input device. RCA is implemented in C# using both the Microsoft .NET extensibility model and the Microsoft Ink API. The extensibility model provides access to the underlying information on the IDE framework, such as a list of all the open windows and their types and content. This content includes the text within code windows, the text's font and line numbers. A range of IDE event notifications are also available (e.g. documentOpened and documentSaved, lineChanged and windowActivated.). The ink API supports collection, selection and manipulation of digital ink, and notification of inking events (e.g. InkAdded and InkDeleted, SelectionMoved and SelectionChanged).

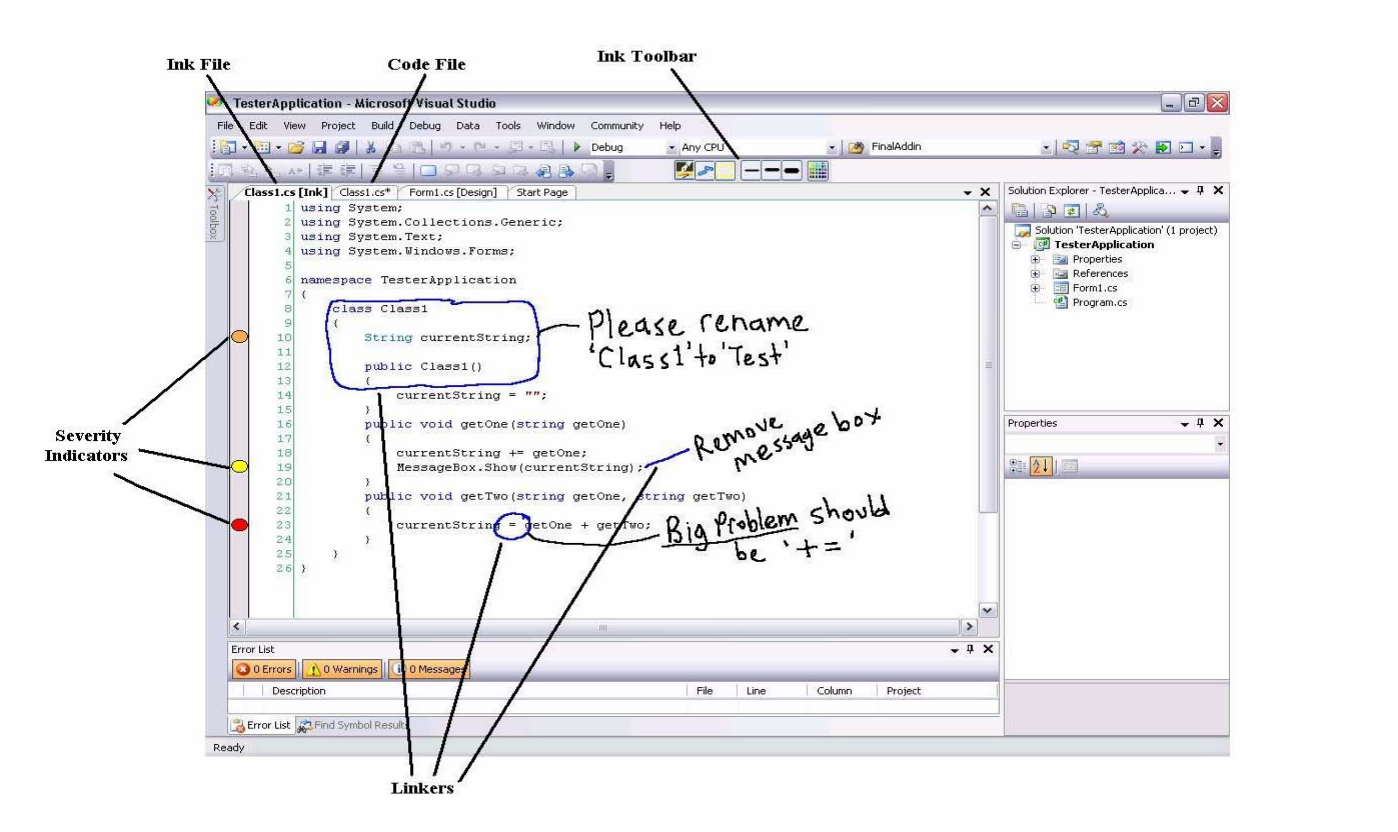

Figure 1 RichCodeAnnotation in Visual Studio .Net

We have implemented an 'Ink' window in the IDE (figure 1). When a user brings up the code file, and selects 'ink mode' from the ink toolbar, a new ink window is created based on the current code window (if not already created). The user edits the source file in the code window and annotates over the source file in the corresponding ink window.

Below we describe the seven main features of RCA: extending the IDE to create an ink window; linking annotations to a specific line; grouping ink into an annotation; editing ink; reflowing ink as the underlying code is modified; saving and loading ink and registering issue severity.

Ink Window Extension

We explored three approaches to ink windows. We tried to add a transparent overlay to hold the digital ink to each code window. We also tried to create a new associated ink window much like an associated 'design' window for a user form in a .NET windows application. Both approaches were difficult to implement due to the limitations of the extensibility model. However, these approaches maybe more successful if the IDE code is accessible and modifiable (such as Eclipse).

The third approach was to create a distinct tool window control that holds ink, much like the 'Shim Control' example in [6], was adopted. The text in the code window

is copied to the ink window where it can then be annotated. The drawback with this approach is that when the code in the source file window is changed, then code in the ink window must also be refreshed to maintain consistency.

Linkers

The linker concept is used to link a group of ink strokes to a specific line. When the mouse/pen cursor resembles a hand, the system is in 'linker mode'. In 'linker mode' there are three available actions: either a line or a circle to create a link to a particular line of code, or ink that hits an existing annotation. These actions automatically change the mode to 'inking mode' and the mouse/pen pointer resembles a pen. Annotations in 'inking mode' are attached to the linker stroke and are free-form.

If the linker is recognized as a line, the corresponding annotations are attached to the code line closest to the start point of the linker. A line is differentiated from a circle by having the start point touching the left border of the bounding box (there is usually no room in the left-hand margin of a code window) and the last point of the stroke touching the right border of the bounding box as shown in figure 2.

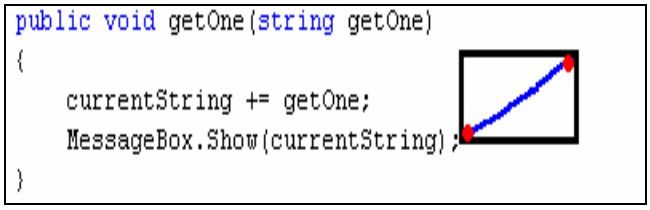

Figure 2 Line linker

If the linker is recognized as a circle then the corresponding annotations are attached to the line closest to the mid-point of the circle's bounding box. A circle is recognized by having the distance between the first point and last point less than the hypotenuses of 25 percent of the bounding box's width and height. That is the length ab must be smaller than the length $a'b'$ as shown in figure 3.

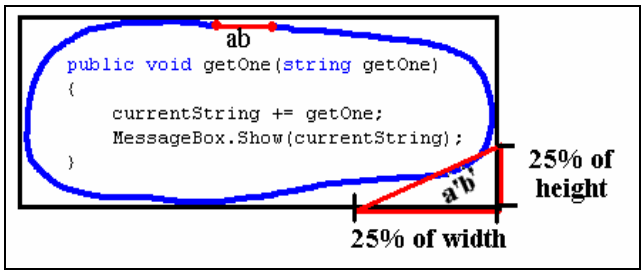

Figure 3 Circle Recognized

If the system is in 'linker mode' and the user taps on an existing annotation, then that annotation group is selected and the interface changes to ink mode. New strokes are joined to the selected annotation.

Ink Grouping

The annotation strokes must be grouped and attached to a linker to allow the system to automatically reflow the digital ink when the underlying code changes. The grouping of ink strokes is decided based on the spatial and temporal properties of strokes [2, 28].

Spatially adjacent strokes are considered to be in the same group. This idea is used in two ways:

- if the user is in 'linker mode', he/she can click on an existing annotation. Then new strokes are joined to the selected annotation.
- if the user is in 'ink mode' and a stroke is made outside the annotation region, even though we are in 'inking mode', the stroke is considered a linker for a new annotation group. This is because annotations made a significant distance away from the current annotation generally represent a new annotation.

Temporal grouping suggests a stroke made shortly after the previous stroke is likely to be part of the same annotation group. Consider when a user annotates 'Incorrect', once the user writes 'I' the 'n' stroke is made shortly after, and therefore should be in the same group as 'I', the same idea works for multiple words and associated diagram strokes. Research has shown that the average time between strokes of the same annotation is approximately 500ms [10]. However, this system uses two seconds between annotation strokes to allow for the user's thinking process. After this time the mode changes from 'inking mode' back to 'linker mode'. The interface indicates the mode change to the user by changing the pen cursor to a hand.

Ink Editing

Ink editing is an important part of any ink annotation tool: one of the advantages of digital ink is that it is easy to alter ink properties and modify existing ink strokes [23]. We support moving annotations by selecting the appropriate linker (using the ink toolbar); this selects all the strokes attached to this linker's annotation. When the annotation group is moved, the group is re-attached to an appropriate line based on the new location of the linker.

Ink strokes can be erased by selecting the eraser from the ink annotation toolbar. If the stroke selected is not a linker then the stroke is simply removed. If the stroke is a linker, the entire annotation is erased. We decided that users must confirm their decision when deleting a linker, as removing a linker also removes its corresponding group of ink annotations.

Ink strokes can be written in different colors as annotation research shows annotators like to use specific color codes to indicate different types of issues [15]. When the user wants to change the ink color, he/she can do so using the color dialog box provided in the toolbar.

Ink Reflow

Ink reflow is necessary in an annotation tool if the underlying document is dynamic [23, 25]. When dealing with text editing tools there is a definite need for reflowing the digital ink [17, 24]. As code is line-based we have concentrated our efforts on vertical reflow. We handle this

situation by picking up the LineChanged event which fires whenever a line of text is changed. This means the tool must handle 4 situations.

- If no lines are added or removed then we don't need to reflow the existing digital ink.
- If x lines are added at line y, then all annotations below line y must move down x lines.
- If x lines are deleted ending at line y, then all annotations below line y must move up x lines.
- Any annotation that exists within deleted lines must also be deleted.

Saving/Loading

When a code document is opened a search for its corresponding ink document is made. If an ink document exists then it is also opened. When a code document is saved, either by the user or when the IDE closes, any corresponding ink document is also saved.

The name assigned to an ink document is automatically derived from the name of its corresponding code document. For example if a code document is named 'HelloWorld.cs' then its corresponding ink document would contain the same name but with an extension of type 'ink serialized format', hence being 'HelloWorld.isf'. This file is stored in the solution's project directory in a folder called 'Ink_Files'.

Issue Severity

An issue's severity is indicated using a symbol within the gray margin bar of the ink document and looks similar to a breakpoint or bookmark. The symbol's color is changed with a tap or click; iterating through green, yellow, orange and red. The green indicator can be used to indicate either a positive comment or that the issue has been 'resolved' by the developer. The other symbols can be used to represent low, medium and high severity respectively.

When a line is annotated in the ink window a bookmark symbol is attached to the corresponding line in the code window. Then when the user is in the code window a bookmark indicates an ink annotation may be associated with this line. This approach allows the user to keep track of where the annotations are when working in the code window.

EVALUATION

So far several informal user evaluations have helped inform our work. The participants chosen were: a post graduate student with experience in .NET hired by the university to mark student assignments; a senior professor who teaches both VB and C#; and a project manager from a software development organization where the majority of staff use .NET as their development platform.

Overall the feedback was positive; however there were a few usability issues that caused a little difficulty. One problem was that users made the mistake of thinking the interface is in 'inking mode' when the mode had automatically reverted to 'linker mode', then saying "Oops, this should be an ink stroke". They then had to delete the newly created linker stroke and select the old group of annotations and rewrite the ink stroke. To alleviate this problem we added a red border around the current annotation group as shown in figure 4. This is achieved by simply finding the bounding box of the annotation. This gives the user a better indication of the mode, as well as showing them the set of strokes that a new stroke will be grouped with.

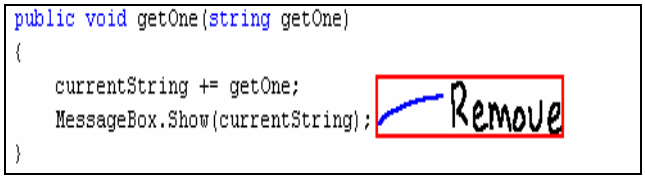

Figure 4 Bounding Box

Second, users were unsure about the spatial constraints on grouping ink. When making a new ink stroke some distance away from the current annotation there is no visual indication as to whether the new stroke will be included in the current annotation or be considered a linker for a new annotation. One user made this statement while in 'inking mode', "as I make an annotation a few lines below this annotation, I would expect this stroke to qualify as a linker". This comment suggested to us that he would rather know for certain that the next stroke would be a linker and not have to wonder whether it might be joined to the current annotation.

To solve this problem we considered the fact that annotators normally write from left to right, then down to the next line. So we extended the bounding box giving extra width and height to the right and below, as shown in figure 5. If a new stroke is within the spare space of the bounding box, then that stroke is grouped with the current annotation group. If a stroke is made outside this bounding box then it is assumed that the user wants the stroke to be a linker for a new annotation group.

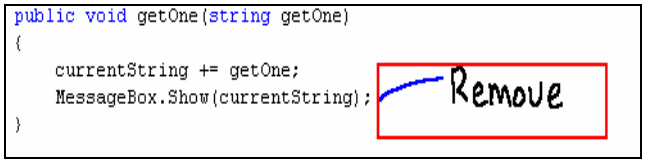

Figure 5 Extended Bounding Box

DISCUSSION AND FUTURE WORK

Our tool supports paperless ink annotation within an IDE code editor. Thus potentially it is of benefit to anyone who reads or writes code. The tool allows developers to comment their code, reviewers to annotate possible coding issues/defects and teachers to explain code, all directly within the IDE.

The advantages of digital ink annotation are clearly documented: providing this functionality within an IDE provides an effective and convenient way to annotate code.

We plan to allow markers, teachers and industry-based project managers and quality assurance staff to use this tool for a period of time. Such a longitudinal study would provide rich feedback for future enhancements. For example, we could extend RCA with marking specific functionality by incorporating ideas found in marking tools [11, 20, 23], so that marking and annotating can be done directly within an IDE. Also several universities use tablet PC's in classes to collaboratively share lecture notes written during class-time [12, 18]. We could include collaboration support so students can use this tool in a shared environment.

This tool can also be extended to provide extra functionality for industry based code analyzers, similar to the formal technical review process FTRrm [13]. This would allow individual reviewers to first conduct private reviews, then have the system collect all annotations and make them public for all reviewers to comment on. The issues raised could then be discussed during a group review meeting, where perhaps more issues would be raised. The system could generate a review meeting report containing all finalized issues along with their line numbers and severity. This report would then be sent back to the developer along with the digital ink annotation files.

CONCLUSIONS

Ink annotation is an effective way to record comments on documents, either on paper or in a digital environment. However program code is different from 'normal' documents in that it is non-linear and that code has well defined, specific syntactic requirements. Program IDEs are specifically designed to support these features, but do not include ink annotation. We have created a plug-in for the Visual Studio IDE to facilitate inking over program code. Early feedback suggests this approach shows promise for both academic and commercial use.

REFERENCES

1. Anderson, R., Anderson, R., Simon, B., Wolfman, S. A., VanDeGrift, T., Yasuhara, K. Experiences with a tablet PC based lecture presentation system in computer science courses. Proc. SIGCSE 2004, ACM Press (2004), 56-60.
2. Bargeron, D. and Moscovich, T. Reflowing digital ink annotations. Proc. SIGCSE 2003, ACM Press (2003), 385-393
3. Basics of .NET., http://www.microsoft.com/net/basics.mspx.
4. Bessler, S., Hagar, M., Benz, H., Mecklenburg, R., and Fischer, S. DIANE: A multimedia annotation system.Proc. ECMAST 1997, Springer-Verlag, 183-198.
5. Brush, A. J., Bargeron, D., Gupta, A., and Cadiz, J. J. Robust annotation positioning in digital documents. Proc. CHI 2001, ACM Press, 285-292.
6. Conwell, J. Blog Reader Add-In for Visual Studio .NET. http://www.codeproject.com/dotnet/BlogReaderArticle.asp.
7. Eclipse.org Home. http://www.eclipse.org/.
8. Fagan, M. Design and code inspections to reduce errors in program development. IBM System Journal 15, 3 (1976), 182-211.
9. Francik, E. Rapid, integrated design of a multimedia communication system. HumanComputer Interface Design (1995), 36-69.
10. Golovchinsky, G., Denoue, L., Moving Markup: Repositioning Freeform Annotations. Proc. UIST 2002, ACM Press (2002), 21-30.
11. Heinrich, E. and Lawn, A. Onscreen marking support for formative assessment. Proc. Ed-Media (2004), 1985-1992.
12. Huang, A., Doeppner, T. W. and Cetintemel, U., Ad-hoc Collaborative Document Annotation on a Tablet PC. Brown University. http://www.cs.brown.edu/publications/theses/ugrad/2003/ashuang.pdf.
13. Johnson, P. M. An instrumented approach to improving software quality through formal technical review. Proc. ICSE (1994), 113-122.
14. Lowell, J. A. Improving Software Quality, Wiley Professional Computing, 1993.
15. Marshall, C. C. Annotation: from paper books to the digital library. Proc DL 1997, ACM Press (1997), 131-140.
16. Marshall, C. C. Toward an ecology of hypertext annotation. Proc. HyperText 1998, ACM Press (1998), 40-48.
17. Microsoft Word. http://www.microsoft.com/office/word/using.htm
18. Mock, K., Teaching with Tablet PCs, Proc. Journal of Computing Sciences in Colleges 20, 2 (2004) 17-27
19. Moran, T. P., C, P. and Van Melle, W. Pen-based interaction techniques for organizing material on an electronic whiteboard. In Symposium on User Interface Software and Technology (1997), 45-54.
20. Myers, D., Hargreaves, E., Ryall, J., Thompson, S., Burgess, M., German, D. and Storey, M. Developing Marking Support within Eclipse. Proc. of OOPSLA 2004, ACM Press (2004), 62-66.
21. Neuwirth, C., Kaufer, D., Chandhok, R., and Morris, J. Computer Support for Distributed Collaborative Writing: Defining Parameters of Interaction. Proc. CSCW 1994, ACM Press (1994), 145-152.
22. Phelps, T., and Wilensky, R. Robust Intra-document Locations. Proc. WWW9 2000, North-Holland Publishing Co. (2000), 105-118.
23. Plimmer, B. and Mason, P. A Pen-based Paperless Environment for Annotating and Marking Student

Assignments. Proc.7th Australasian User Interface Conference, CRPIT press (2006), 37-44.

24. Ramachandran, S. and Kashi, R. An architecture for ink annotations on web documents. Proc. 17th International Conference on Document Analysis and Recognition, IEEE Computer Society (2003), 256-260.
25. Schilit, B. N., Golovchinsky, G. and Price, M. N. Beyond Paper: Supporting Active Reading with Free
26. rm Digital Ink Annotations. Proc. CHI 1998, ACM Press (1998), 249-256.
27. Simon, B., Anderson, R., Hoyer, C. and Su, J. Preliminary Experiences with a Tablet PC Based System to Support Active Learning in Computer Science Courses. Proc. SIGCSE 2004, ACM Press (2004), 213-217.
28. W3C Document Object Model. http://www.w3.org/DOM/.
29. Wilcox, L. and Chiu, P. A Dynamic Grouping Technique for Ink and Audio Notes. Proc. UIST 1998, ACM Press (1998), 195-202.