Inking in the IDE: Experiences with Pen-based Design and Annotation

Beryl Plimmer[1], John Grundy[1, 2], John Hosking[1] and Richard Priest[1]

[1]Department of Computer Science and [2]Department of Electrical and Computer Engineering
University of Auckland, Private Bag 92019
Auckland, New Zealand
{beryl@cs, john-g@cs, john@cs, rpri032@ec}.auckland.ac.nz

Abstract

Hand-drawn designs and annotations are a common, human-centric approach frequently used during software design and code inspection. We describe our research experiences of adding support for hand-drawn design and annotation to three Integrated Development Environments (IDEs): a software design tool; a user interface design tool; and a programming tool. The aim of this work is to provide users with more natural interaction techniques seamlessly integrated into their IDEs through the use of hand-drawn diagrams, layouts and code mark-ups.

1 Introduction

When designing software, it is very common for software developers to use sketching and annotation as an aid to the process. Examples include hand drawn and hand annotated UML diagrams, sketched user interface designs, and hand annotations on code printouts. Such sketches and annotations form an important part in the exploration of alternative designs, formalization of the designs, and communication of issues between collaborating designers. Evidence suggests that hand sketching results in superior designs, with collaborating designers more prepared to critique and modify sketches than with more formalized artifacts such as screen mockups [3],[7],[17].

There has been considerable work in the area of pen based sketch input of designs, with support for formalization of sketches into design artifacts. One of the earliest, SILK [11], allows software designers to sketch an interface using an electronic pad and stylus. SILK recognizes widgets and other interface elements as soon as they are drawn using Rubine's single stroke gesture recognition algorithm [21] and can transform sketches into standard Motif widgets. Denim [13] provides a similar approach for web interface design while Knight [5] and SUMLOW [4] support UML diagram sketching. However, while these systems support sketching and sketch formalization, they typically lack close integration with other development tools. Pen and ink annotation on paper documents is a natural way to record comments and offers many advantages over keyboard and text based annotations [14], [15], [1]. Yet, computerization allows additional support for sharing, storage, transmission and manipulation of both the original document and the annotations [10], [1]. Others have developed tools for dynamic documents such as web pages where the ink must be anchored to specific words or sections of the document and reflow with document changes [1], [8]. User evaluations we have undertaken on our standalone design sketch system, SUMLOW [4] showed that sketched designs are very useful during early phase UML diagramming and when collaboratively reviewing and revising designs. Similarly PenMarked [19] user studies demonstrated that annotating code to mark programming assignments works well when ink annotations are computerized. These studies have affirmed to us that preserving sketch content and having it formalized and adding pen-based code annotation support are both appropriate and useful during software design and review. However, both studies indicated that better integration with an IDE was essential.

Our thesis is that only with such integration will the reported benefits of sketching and annotation have practical impact. This is due to the overhead of importing from and exporting to (if possible) standalone tools, which are generally research prototypes that are difficult to evaluate on real development problems. Another impetus is the non-linear nature of modern software designs and code which makes paper-based annotation less viable than previously. For example, Fagan [6] showed the advantages of code review in 1976, yet it is still not widely practiced. Similarly, collaborative reviews of UML designs have been shown to be beneficial [9], but this is poorly supported in current tools. Our sub-thesis is that the ability to informally annotate documents is central to the software review process, but is poorly supported in current IDEs.

2 Motivation

Consider a software development team collaborating over the development of a software component. Figure 1 shows three examples of their collaborations: (a) a sketched UML design for the component on paper; (b) some sketched screen layout designs on a whiteboard; and (c) code printouts with annotations used to support code review/critiquing and planning for changes.

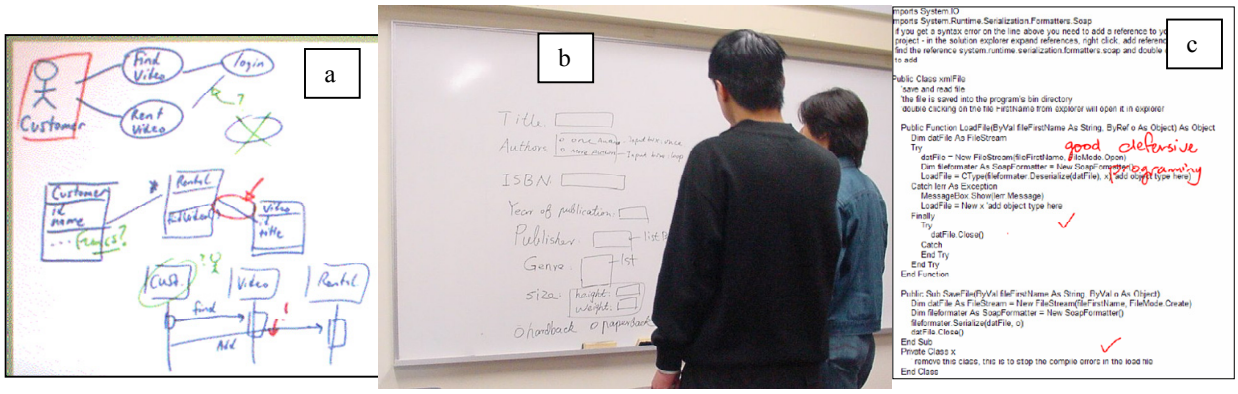

Figure 1. Human-centric sketching and annotation with whiteboards, pen and paper.

These sketched designs, annotations, revisions and comments are a very natural, human-centric approach regularly used on these artifacts of the software lifecycle. However, in each case here, the design decisions have been made separately from the software tools that the designers are using to actually develop the software. This makes it both difficult to reconcile decisions made with the evolving software artifact and to formally record them. Sharing paper designs and annotations with those not present is difficult.

Ideally it is desirable to support design sketches and their formalizations and code annotations and their resolution within the IDE. This means they are raised to first-class objects and are maintained with the rest of the software artifact, including retention via versioning support, maintenance through artifact modification, sharing via distributed collaboration technologies etc. However, no existing widely used IDE provides such comprehensive sketching and annotation support.

Abstracting from our motivating example we assert the following requirements for supporting "inking" (as a generic term for pen-based sketch/annotation) in an IDE:

- enabling "pen"-based input of content e.g. via a mouse, tablet PC, and/or e-whiteboard
- capturing such pen-based input into the IDE in a SEAMLESS way
- support for formalization of the sketched content e.g. into design diagrams or code review issues
- the ability to link the sketched content to e.g. code files/lines, or design diagram/elements. This needs to be trainable and customizable to allow new types of artifact to be incorporated as the IDE is upgraded
- support for consistency so that changes to the design/code in the IDE imply that the ink is changed and vice-versa
- accurate recognition of ink content
- ink is treated as first-class objects in the IDE, ie it can be saved, loaded, kept with IDE content and versioned

3 Three Exemplars

To explore realization of these requirements our approach has been to develop three exemplars with the aim to generalize from then, following the *Three Examples* pattern of the *Evolving Frameworks Pattern Language* [20]. Each exemplar adds plug-in ink support to an existing, commonly used IDE.

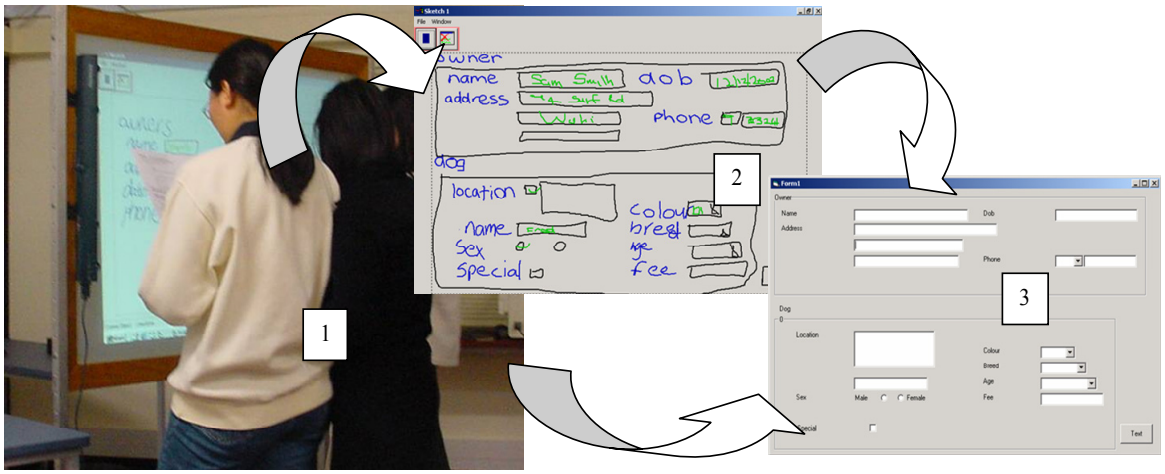

Figure 2. Sketching user interface designs in FreeForm.

3.1 *FreeForm*

Freeform is a sketch tool for user interface design [16],[17]. It is tightly integrated into the Visual Basic IDE and accessed from the VB tool bar. User interface forms can be sketched, executed (as sketches), recognized and beautified before conversion into a VB form. The sketch space Figure 2 (1) is an unconstrained canvas. Here the user can explore UI design ideas much as they would on paper. The storyboard shows thumbnails of each sketch page and facilitates the establishment of navigation links between forms. In execute mode (2) designers can 'play computer' filling the form and following links between forms. Once satisfied with the design, the recognition engine and beautifier are used to convert the sketch to a VB form (3). The effectiveness of an integrated sketch design tool is dependent on reliable recognition and appropriate beautification. Freeform has a three phase recognition engine. First, ink strokes (both drawing and writing) are recognized using a modified Rubine's algorithm [21]. The libraries of basic shapes and characters used for this pattern matching are exposed to the user so that they can modify them. Second, characters are combined into words and matched against a small vocabulary of common UI words. The last phase is the recognition of UI components. This is achieved by application of adjacency rules such as *a* contains *b*, or *a* is beside *b*, for example a radio button might be defined as 'a small circle with a word beside it'. As with the shape library, the rule base is exposed to the user who can add or change both components and rules.

Recognition results are superimposed on the sketch. At this point the user can correct any mis-recognition. When the user clicks 'create form' a new form is created in the VB project and components are 'beautified' and placed on it. Beautification includes aligning components to a grid and standardizing sizes. The user can move between VB forms and sketches, regenerating forms as required.

3.2 *MaramaSketch*

MaramaSketch is an extension to Marama, an Eclipse IDE plug-in to support diagram editor generation and realization. Marama provides conventional CASE tool diagramming support via drag-and-drop within a form, a tool palette, and diagram content manipulation. MaramaSketch allows a user to instead sketch a Marama diagram with mouse or Tablet PC stylus and have the diagram symbols recognized and on-demand converted into Marama computer-drawn content.

Figure 3 shows examples of MaramaSketch in use for the MaramaMTE software architecture design tool. MaramaSketch provides a canvas (centre, 1) where the user can sketch diagram content, as shown for a simple web-based system architecture design. At left and top are standard Eclipse IDE file browser and menus. At right is the MaramaSketch recognition control panel (which can be hidden if not required). The user has the option of having sketched content being recognized on user request or to have them immediately recognized and converted into beautified MaramaMTE architecture symbols as they are drawn. In (2) some symbols and text have been recognized and converted to architecture diagram symbols and textual property values, while the customer service has not yet been converted. The user has asked for both sketched content and MaramaMTE symbols to be shown here.

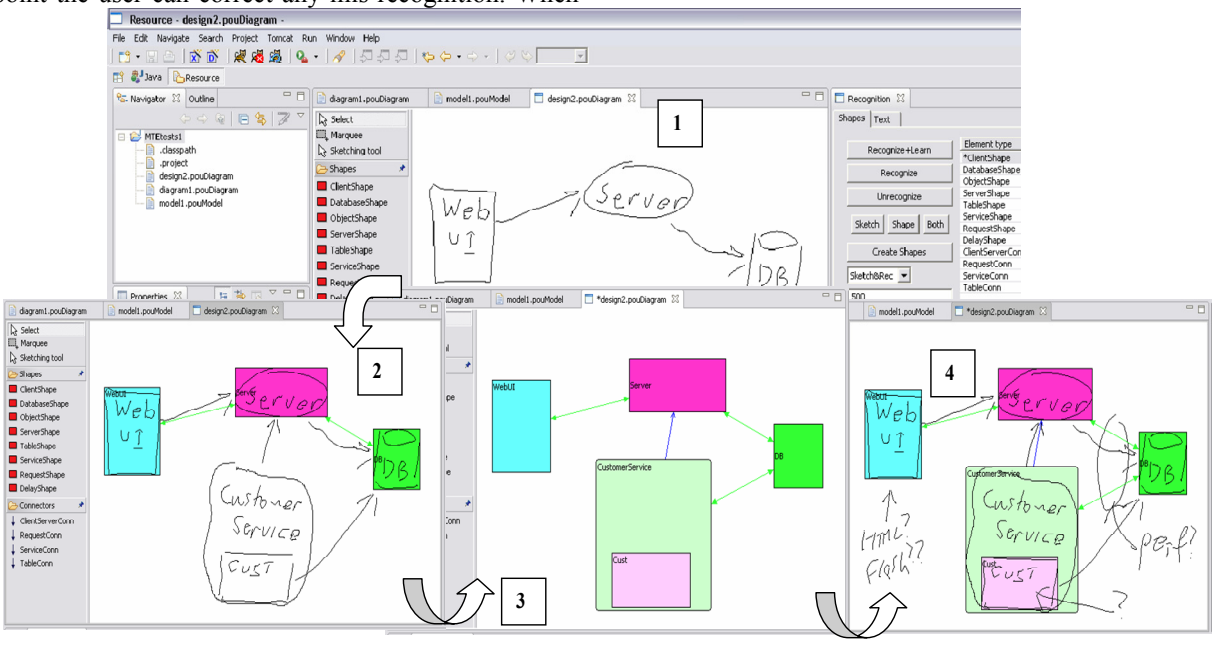

Figure 3. Sketching a software architecture design in MaramaMTE.

In (3) the user has asked for ink to be hidden, the recognized MaramaMTE architecture diagram notation symbols only showing. In (4) the user is showing both sketched and computer-drawn content and annotating the diagram to discuss possible revisions with others.

MaramaSketch provides an overlay to standard Marama diagramming tools, allowing ink sketches to be on-request or immediately recognized and converted to appropriate Marama diagram elements. Key design principles of MaramaSketch were to permit the user to choose when symbols are recognized and converted to Marama beautified symbols and (like Freeform) to allow training on-the-fly of the recognizer. The user may request only sketch, only Marama symbol, or both be displayed, or selectively show/hide ink and symbol. The user may elect to have newly drawn sketches immediately beautified to Marama symbols or do this on request for selected ink. Changes to sketched ink e.g. resize or move or delete are immediately reflected in the associated recognized Marama symbol, and vice-versa. The result is a very flexible, user-controlled morphing between sketch and Marama symbols seamlessly integrated into the IDE.

3.3 RichCodeAnnotation

RCA is a tool developed as a VisualStudio .NET IDE plug-in to support inking over dynamically changing code files [19]. This tool allows the user to note issues found in the code file within the digital document using coloured ink annotations, much like marking up a paper code printout with pen. When the user wishes to mark a portion of code, they can bring up the code file's associated 'Ink' window. The ink window is an exact replica of its corresponding code counter-part, however you can ink over this window (Figure 4). When the user makes an annotation, it is automatically attached to a line along with its 'ink bookmark' in the margin of the ink window with an initial severity rating of medium (orange). Our current implementation links the ink to the line closest to the start-point of a new annotation. If the annotation was attached to the wrong line, the user can drag the 'ink bookmark' to the correct line. The user can then alter the severity to low (green), high (red) or simply keep it as medium. The user can switch between the normal and annotated view of the code via the window tabs or project explorer. As lines are added or removed from the code, existing ink annotations in the ink window dynamically move with the line they are attached to. Ink files are saved and loaded automatically with a project.

Ink annotation of documents aids understanding of the document for both the annotator and subsequent reader of the document. RCA aids code reviewers in analyzing other programmers' work and sending them feedback. Markers in an academic environment can also use this tool, as their job is similar to that of a code reviewer; they can annotate a student's work to provide formative feedback. Likewise teachers may use it in demonstrations to highlight and describe portions of code, these annotations can then be saved and made available to students for future reference.

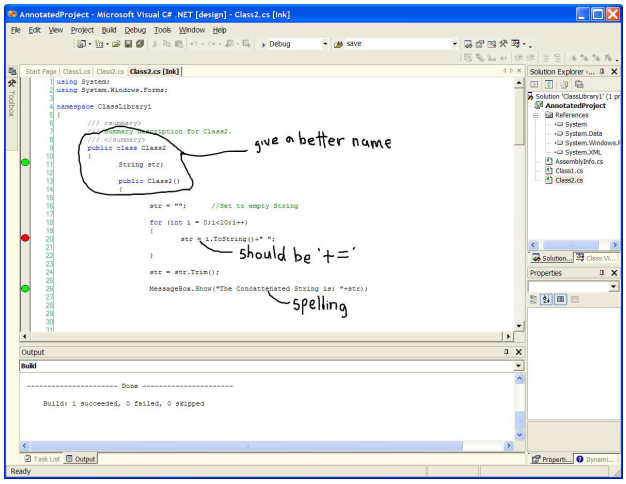

Figure 4. Annotating code in Visual Studio.

4 Discussion

. Pen-based computing platforms such as PDAs, Tablet PCs and large-screen E-whiteboards have become relatively commonplace. However few current applications – especially IDEs – running on such hardware make much if any use of ink. Even ink content for Microsoft Word on the Tablet PC provides only limited recognition of shapes and text and treats such content rather differently to standard keyboard and mouse interactions. In contrast our three exemplars illustrate the possibilities for human-centric interaction using fully-integrated ink sketch and annotation content within a variety of IDEs

Ideally users should be able to add pen-based ink content within their IDEs as seamlessly as they use a mouse to add and manipulate design diagrams and a keyboard to enter and modify program code. Users should be able to move between pen-based interaction and mouse/keyboard interaction seamlessly, and ink content be treated the same as other IDE content from mouse or keyboard. Recognition of ink content and creation and manipulation of standard design diagram elements and program code fragments or code annotations should be supported either eagerly or lazily, depending on user preference and the task at hand.

In order to make this a reality, IDE developers need to support the creation and manipulation of IDE content from a variety of sources – not just pen-based ink but also speech, still image, video and other tactile devices. Non-mouse/keyboard device content and any interactions with other IDE content should be supported with extensible control and configuration. With pen-based ink and pen-based content manipulation user preferences and software development tasks are important to consider and provide

mechanisms for which to adapt. Ink in the IDE should allow progressive recognition and formalization of pen-based content where appropriate and close association of ink with other mouse/keyboard content. Ultimately ink content should be treated as first-class within the environment, allowing, for example, ink to be associated just as readily with other ink content, for example ink annotation of FreeForm and MaramaSketch sketched designs and notes on comments in RichCodeAnnotation.

Visual programming in general lends itself well to ink-based creation and manipulation as demonstrated in the MaramaSketch and FreeForm design environments. Visual language IDE developers could also consider use of ink to support human-centric creation and manipulation of content. As with existing IDEs challenges include appropriate integration of ink with computer-drawn diagrams, use of high quality recognition algorithms, and appropriate timing of recognition and beautification of ink. A further consideration is the need to integrate inking into execution, permitting, for example, annotation of sketches by the environment executing the program or software system derived from the sketch to allow users to visualize execution behavior using the same "formalism" as used to specify the program (as advocated in the *Language Tools* pattern of *Evolving Frameworks Pattern Language* [20]).

Our experiences demonstrate that current ink recognition algorithms require considerable further research and development to enable efficient and effective ink usage within a range of IDEs. Configuration of recognition of ink to ensure appropriate eager vs lazy recognition of ink is essential to ensure both user preferences and IDE task are well-supported.

5 Summary

We have developed three exemplar plug-ins to Integrated Development Environments demonstrating the effectiveness of pen-based content creation and manipulation. These domains include software design diagramming, user interface design and code review supported by annotation. Plug-ins have been successfully developed for the Visual Basic, Eclipse and Visual Studio IDEs. Lessons from this work include the need for more open IDEs to allow pen-based content creation and management, more effective and configurable recognition algorithms, and flexibility in configuring and using pen-based ink in the IDE to improve human-centric interaction. Using pen-based interaction in combination with other human-centric techniques, such as speech and gesture recognition, may further enhance the human-centric features of Integrated Development Environments.

References

[1] Bargeron, D. and T. Moscovich. *Reflowing digital ink annotations*. Proc *Chi03*. ACM: p. 385 - 393.

[2] Brush, A.B., et al. *Robust annotation positioning in digital documents*. Proc *Sigchi'2001*, ACM: p. 285-292.

[3] Black, A., Visible planning on paper and on screen: The impact of working medium on decision-making by novice graphic designers. *Behaviour and information technology*, 1990. 9(4): p. 283-296.

[4] Chen, Q., Grundy, J.C. and Hosking, J.G. An E-whiteboard Application to Support Early Design-Stage Sketching of UML Diagrams, Proc HCC'2003, IEEE, 219-226.

[5] Damm, C.H., K.M. Hansen, and M. Thomsen. *Tool support for cooperative object-oriented design: Gesture based modelling on and electronic whiteboard*. Proc *Chi 2000*. 2000: ACM: p. 518-525.

[6] M. E. Fagan, Design and Code Inspections to Reduce Errors in Program Development, *IBM System Journal*, vol. 15 (3), 1976

[7] Goel, V., Sketches of thought. 1995, Cambridge, Massachusetts: The MIT Press.

[8] G. Golovchinsky, L. Denoue, "Moving Markup: Repositioning Freeform Annotations, 2002

[9] P. Grünbacher, M. Halling and S. Biffl, An Empirical Study on Groupware Support for Software Inspection Meetings, Proc. 18th IEEE International Conference on Automated Software Engineering, Montreal, 2003.

[10] A. Huang, T. W. Doeppner, U. C. Readers, Ad-hoc Collaborative Document Annotation on a Tablet PC, 2003

[11] Landay, J. and B. Myers. *Interactive sketching for the early stages of user interface design*. Proc *Chi '95*. ACM: p. 43-50.

[12] Landay, J. and B. Myers, *Sketching Interfaces: Toward more human interface design.* Computer, 2001. **34**(3): p. 56-64.

[13] Lin, J., Newman, M.W., Hong, J.I. and Landay, J. A. Denim: Finding a tighter fit between tools and practice for web design, Proceedings of CHI'2000, ACM Press, pp. 510-517

[14] C. C. Marshall, Annotation: from paper books to the digital library, 1997

[15] C. C. Marshall, Reading and Interactivity in the Digital Library: Creating an experience that transcends paper, *Proc CLIR/Kanazawa Institute of Technology Roundtable*, Kanazawa, Japan, July 3-4, 2003, pp. 5.4.1-20.

[16] Plimmer, B.E. and M. Apperley. *Software for Students to Sketch Interface Designs*. proc *Interact*. 2003. Zurich: p. 73-80.

[17] Plimmer, B.E. and M. Apperley. INTERACTING with sketched interface designs: an evaluation study. proc SigChi 2004. 2004. Vienna: ACM: p. 1337-1340.

[18] Plimmer, B.E. and J. Grundy. *Beautifying sketch-based design tool content: issues and experiences*. Proc *AUIC 2005*. CRPIT, pp. 31-38.

[19] Plimmer, B. and Mason, R. A Pen-based Paperless Environment for Annotating and Marking Student Assignments, Proc. Of the 6th *Australasian User Interface Conference*, CRPI, Hobart, Australia, Jan 2006.

[20] Roberts, D., Johnson, R., Evolving Frameworks a pattern language for developing object-oriented frameworks, http://st-www.cs.uiuc.edu/users/ droberts/evolve.html

[21] Rubine, D. *Combining gestures and direct manipulation*. proc *CHI '92*. 1992: p. 659-660.