Submitted to the Handbook of Software Engineering and Knowledge Engineering,
Volume 2 (see http://www.ksi.edu/seke/hand.html)
URL= http://tim.menzies.com/pdf/00vp.pdf.

Evaluation Issues for Visual Programming Languages

Tim Menzies,

Department of Electrical & Computer Engineering
University of British Columbia, Canada
<tim@menzies.com>

November 28, 2000

1. Introduction

Many systems used in software engineering and knowledge engineering use some sort of visual presentation. Many researchers claim numerous benefits for visual frameworks. For example:

> *When we use visual expressions as a means of communication, there is no need to learn computer-specific concepts beforehand, resulting in a friendly computing environment which enables immediate access to computers even for computer non-specialists who pursue application* [8].

This case that pictures assist in explaining complicated knowledge seems seems intuitively obvious. But is it correct? Other widely held intuitively obvious beliefs have been found to be incorrect, and sometimes even spectacularly so:

- Galen's incorrect descriptions of human physiology were treated as virtual gospel for 1300 years until one up-start surgeon had the gall (pun intended) to pick up a scalpel and perform dissections for himself (see Versalius' *De Humani Corporis Fabrica, 1543*).
- In took six decades before empirical studies [14] demolished the traditional picture of managers as systematic planners. Those studies found that, in the usual case, managers lacked the time to be systematic. For example, that study found:
  - Foremen who performed one new task every 48 seconds during their entire shifts;
  - Managers who worked for half an hour or more without interruption only once every two days.

Clearly, pre-experimental intuitions must be verified, not matter how compelling they may seem. This article takes a critical look at the available evidence on the efficacy of visual programming (VP) systems. After an introduction to VP, we will review theoretical studies and small scale experimental studies suggest an inherent utility in visual expressions. However, when we explore the available experimental evidence, we find numerous contradictory results.

2. A (Brief) Introduction to Visual Programming

As a rough rule-of-thumb, a visual programming system is a computer system whose execution can be specified *without scripting* except for entering unstructured strings such as *Monash University Banking Society* or simple expressions such as *X above 7* . Visual representations have been used for many years (e.g. Venn diagrams) and even centuries (e.g. maps). Executable visual representations, however, have only arisen with the advent of the computer. With falling hardware costs, it has become feasible to build and interactively manipulate intricate visual expressions on the screen.

More precisely, a non-visual language is a one-dimensional stream of characters while a VP system uses at least two dimensions to represent its constructs [3]. We distinguish between a *pure VP* system and a *visually supported* system:

Pure VP systems: These must satisfy two criteria.

1. *Rule 1:* The system must execute. That is, it is more than just a drawing tool for software or screen designs.
2. *Rule 2:* The specification of the program must be modifiable within the system's visual environment. In order to satisfy this second criteria, the specification of the executing program must be configurable. This modification must be more than just (e.g.) merely setting numeric threshold parameters.

Visually supported: Most commercial VP systems such as VISUAL BASIC do not satisfy rules one and two, yet offer some graphical support such a tree description of the class hierarchy. We call these systems *visually supported*, not *pure VP*.

3. Arguments for the Advantages of VP

Many authors argue that VP systems are a better method for users to interact with a program. Green et. al. [7] and Moher et.al. [15] summarize claims such the above quote from [8] as *the superlativist position*; i.e. graphical representations are inherently superior to textual representations. Both the Green and Moher groups argue that this claim is not supported by the available experimental evidence. Further, they argue against claims that visual expressions offer a higher *information accessibility*; for example:

> *Pictures are superior to texts in a sense that they are abstract, instantly comprehensible, and universal.* [8]

My own experience with students using visual systems is that the visual environment is very motivating to students. Others have had the same experience:

> *The authors report on the first in a series of experiments designed to test the effectiveness of visual programming for instruction in subject-matter concepts. Their general approach is to have the students construct models using icons and then execute these models. In this case, they used a series of visual labs for computer architecture. The test subjects were undergraduate computer science majors. The experimental group performed the visual labs; the control group did not. The experimental group showed a positive increase in attitude toward instructional labs and a positive correlation between attitude towards labs and test performance.* [22]

For another example of first year students being motivated by a VP language, see [5] (p18-19). However, merely motivating the students is only half the task of an educator. Apart from motivating the students, educators also need to train students in the general concepts that can be applied in different circumstances. The crucial case for evaluating VP systems is that VP systems improve or simplify the task of comprehending some conceptual aspect of a program. If we extend the concept of VP systems to diagrammatic reasoning in general, then we can make a case that VP has some such benefits. Larkin and Simon [12] distinguish between:

- *Sentential representations* whose contents are stored in a fixed sequence; e.g. propositions in a text.
- *Diagrammatic representations* whose contents are indexed by their position on a 2-D plane.

While these two representations may contain the same information, their computational efficiency may be different. Larkin and Simon present a range of problems modeled in a diagrammatic and sentential representation using production rules. Several effects were noted:

- *Perceptual ease:* Certain features are more easily extracted from diagrams than from sentential representations. For example, adjacent triangles are easy to find visually, but require a potentially elaborate search through a sentential representation.
- *Locality aids search:* Diagrams can group together related concepts. Diagrammatic inference can use the information in the near area of the current focus to solve current problems. Sentential representations may store related items in separate areas, thus requiring extensive search to link concepts.

A common internal representation for a VP systems is one that preserves physical spatial relationships. For example, Narayanan et.al. [16] use Glasgow's array representation [4] to reason about device behaviors. In an array representation, physical objects are mapped into a 2-D grid. Adjacency and containment of objects can be inferred directly from such a representation. Inference engines can then be augmented with diagrammatic reasoning operators which execute over the array (e.g. boundary following, rotation).

Other authors have argue that diagrams are useful for more than just spatial reasoning. Koedinger [11] argued that diagrams can support and optimize reasoning since they can model whole-part relations. Kindfield [10] studied how diagram used changes with expertise level. According to Kindfield, diagrams are like a temporary swap space which we can use to store concepts that:

- Don't fit into our head right now and...
- Can be swapped in rapidly; i.e. with a single glance.

Goel [6] studied the use of ill-structured diagrams at various phases of the process of design. In a well-structured diagram (e.g. a picture of a chess board), each visual element clearly denotes one thing of one class only. In a ill-structured diagram (e.g. an impressionistic charcoal sketch), the denotation and type of each visual element is ambiguous. In the Goel study, subjects explored

- preliminary design,
- design refinement, and
- design detailing

using a well-structured diagramming tool (MacDraw) and a ill-structured diagramming tool (freehand sketches using pencil and paper). Free-hand sketches would generate many variants. However, the well-structured tool seemed to inhibit new ideas rather than help organize them. Once something was recorded in MacDraw, that was the end of the evolution of that idea.

> *One gets the feeling that all the work is being done internally and recorded after the fact, presumably because the external symbol system (MacDraw) cannot support such operations.* [6]

Goel found that ill-structured tools generated more design variants (i.e. more drawings, more ideas, more use of old ideas) than well-structured tools. We make two conclusions from Goel's work. Firstly, at least for the preliminary design, ill-structured tools are better. Secondly, after the brain-storming process is over, well-structured tools can be used to finalize the design.

4. Evaluating the Arguments for VP

It is not clear which of the above advantages apply to general software or knowledge engineering. Many software engineering or knowledge engineering problems are not naturally two-dimensional. For example, while we write down an entity-relationship diagram on the plane of a piece of paper, the inferences we can draw from that diagram are not dependent on the physical position of (e.g.) an entity.

In terms of the ill-structured/well-structured division, the VP tools I have seen in the SE/KE field are all well-structured tools. That is, they are less suited to brain-storming than producing the final product.

Jarvenpaa and Dickson (hereafter, JD) report an interesting pattern in the VP literature [9]. In their literature review on the use of graphics for supporting decision making, they find that most of the proponents of graphics have never tested their claims.

Further, when those tests are performed, the results are contradictory and inconclusive. For example:

- JD cite 11 publications arguing for the superiority of graphics over tables for the purposes of elementary data operations (e.g. showing deviations, summarizing data). None of these publications tested their claims. Such tests were performed by 13 other publications which concluded that graphics were better than tables (37.5 percent), the same as tables (25 percent), or worse than tables (37.5 percent)
- JD cite 11 publications arguing for the superiority of graphics over tables for the purposes of decision making (e.g. forecasting, planning, problem finding). None of these publications tested their claims. Such tests were performed by 14 other papers which concluded that graphs were better than tables (27 percent), the same as tables (46 percent), or worse than tables (27 percent).

Similar contradictory results can be found in the study of control-flow and data-flow systems.

- The utility of flowcharts for improving program comprehension, debugging, and extensibility was studied by Shneiderman [19]. Shneiderman found no difference in the performance of the subjects using/not using control-flow diagrams.
- On the other hand, recent results have been more positive [18].
- Studies have reported that Petri nets are comparatively worse as specification languages when compared to pseudo-code [1] or E-R diagrams [20].
- On the other hand, another study suggests that Petri nets are better than E-R diagrams for the maintenance of large expert systems [20].

Given these conflicting results, all that can conclude at this time is that the utility of control-flow or data-flow visual expressions are an open issue.

In other studies, the Green group explored two issues: superlativism and information accessibility (defined above). Subjects attempted some comprehension task using both visual expressions and textual expressions of a language. The Green group rejected the superlativism hypothesis when they found that tasks took longer using the graphical expressions than the textual expressions. The Green group also rejected the information accessibility hypothesis when they found that novices had more trouble reading the information in their visual expressions than experts. That is, the information in a diagram not *instantly comprehensible and universal*. Rather, such information can only be accessed after a training process.

The Moher group performed a similar study to the Green group. In part, the Moher study used the same stimulus programs and question text as the Green group. Whereas the Green group used the LABVIEW data-flow system, the Moher group used Petri nets. The results of the Moher group echoed the results of the Green group. Subjects were shown three variants on a basic Petri net formalism. In no instance did these graphical languages outperform their textual counterparts.

The Moher group caution against making an alternative superlativism claim for text; i.e. text is better than graphics. Both the Moher and Green groups distinguished between sequential programming expressions such as a decision true and circumstantial programming expressions such as a backward-chaining production rule. Both sequential and circumstantial programs can be expressed textual and graphically. The Moher group comments that:

> *Not only is no single representation best for all kinds of programs, no single representation is ... best for all tasks involving the same program.* [15]

Sequential programs are useful for reasoning forwards to perform tasks such as prediction. Circumstantial programs are output-indexed; i.e. the thing you want to achieve is accessible separately to the method of achieving it. Hence, they are best used for hypothesis-driven tasks such as debugging.

5. VP as Explanation

The core of the case for VP is something like VP lets us *explain* the inner workings of a system at a glance. This section explores the issue of VP and explanation using the BALSA system.

In the BALSA animator system [2], students can (e.g.) contrast the various sorting algorithms by watching them in action. Note that animation is more than just tracing the execution of a program. Animators aim to explain the inner workings of a program. Extra explanatory constructs may be needed on top of the programming primitives of that system. For example, when BALSA animates different sorting routines, special visualizations are offered for arrays of numbers and the relative sizes of adjacent entries.

Animators like BALSA may or may not be pure VP systems. BALSA does not allow the user to modify the specification of the animation. To do so requires extensive textual authoring by the developer. BALSA therefore does not satisfy the Rule 2 of pure VP system (defined above).

One drawback with the BALSA system is that its explanations must be hand-crafted for each task. General principles for explanation systems are widely discussed in AI. Wick and Thompson [21] report that the current view of *explanation* is more elaborate than merely *print the rules that fired* or the *how* and *why* queries of traditional rule-based expert systems. Explanation is now viewed as an inference procedure in its own right rather than a pretty-print of some filtered trace of the proof tree. In the current view, explanations should be customized to the user and the task at hand. For example:

- Paris [17] describes an explanation algorithm that switches from process-based explanations to parts-based explanations whenever the explanation procedure enters a region which the user is familiar with.
- Leake [13] selects what to show the user using eight runtime algorithms. For example, when the goal of the explanation is to minimize undesirable effects, the selected structures are any pre-conditions to anomalous situations. Leake's

explanation algorithms require both a cache of prior explanations and (like Paris) an active user model.

Summarizing the work of Wick and Thompson, Leake, and Paris, I diagnosis the reason for the lack of generality in BALSA's explanation system as follows. BALSA's explanation systems were hard to maintain since BALSA lacked:

1. The ability to generate multiple possible explanations;
2. An explicit user model
3. A library of prior explanations;
4. A mechanism for using (2) and (3) to selectively filter (1) according to who is viewing the system.

6. Summary

On the positive side, we have seen that:

- Visual systems are more motivating for beginners than textual systems.
- In the case of spatial reasoning problems, a picture may indeed be worth 10,000 words [12]. Given some 2-D representation of a problem (e.g. an array representation), spatial reasoning can make certain inferences very cheaply.
- Also, ill-structured diagramming tools are a very useful tool for brainstorming ideas.

On the negative side, beyond the above three specific claims, the general superlativist case for VP improving SE and KE tasks is not very strong:

- Many software engineering and knowledge engineering problems are not inherently spatial.
- Most of the VP systems I am aware of do not support Goel's ill-structured approach to brainstorming.
- The JD research suggests that claims of the efficacy of VP systems have been poorly documented.
- The Moher and Green groups argue that VP evaluations cannot be made in isolation to the task of the system being studied.
- Lastly, a diagram may not necessarily support information accessibility for knowledge. A good explanation device requires far more than impressive graphics (recall the BALSA case study).

References

1. D.A. Boehm-Davis and A.M. Fregly. Documentation of concurrent programs. *Human Factors*, 27:423–432, 1985.
2. M.B. Brown and R. Sedgewick. Techniques for algorithm animation. *IEEE Software*, pages 28–39, January 1985.
3. T.B. Brown and T.D. Kimura. Completeness of a visual computation model. *Software- Concepts and Tools*, pages 34–48, 1994.
4. J. Glasgow, H. Narayanan, and B. Chandrasekaran (eds). *Diagrammatic Reasoning : Cognitive and Computational Perspectives*. MIT Press, 1995.
5. E.P. Glinert and S.T. Tanimoto. Pict: An interactive graphical programming environment. *IEEE Computer*, pages 7–25, November 1984.
6. V. Goel. "ill-structured diagrams" for ill-structured problems. In *Proceedings of the AAAI Symposium on Diagrammatic Reasoning Stanford University, March 25-27*, pages 66–71, 1992.
7. T.R.G. Green, M. Petre, and R.K.E. Bellamy. Comprehensibility of visual and textual programs: The test of superlativism against the "match-mismatch" conjecture. In *Empirical Studies of Programmers: Fourth Workshop*, pages 121–146, 1991.
8. M. Hirakawa and T. Ichikawa. Visual language studies - a perspective. *Software- Concepts and Tools*, pages 61–67, 1994.
9. S.L. Jarvenpaa and G.W. Dickson. Graphics and managerial decision making: Research based guidelines. *Communications of the ACM*, 31(6):764–774, June 1988.
10. A.C.H. Kindfield. Expert diagrammatic reasoning in biology. In *Proceedings of the AAAI Symposium on Diagrammatic Reasoning Stanford University, March 25-27*, pages 41–46, 1992.
11. K.R. Koedinger. Emergent properties and structural constraints: Advantages of diagrammatic representations for reasoning and learning. In *Proceedings of the AAAI Symposium on Diagrammatic Reasoning Stanford University, March 25-27*, pages 154–159, 1992.
12. J.H. Larkin and H.A. Simon. Why a diagram is (sometimes) worth ten thousand words. *Cognitive Science*, pages 65–99, 1987.
13. D.B. Leake. Focusing Construction and Selection of Abductive Hypotheses. In *IJCAI '93*, pages 24–29, 1993.
14. H. Mintzberg. The Manager's Job: Folklore and Fact. *Harvard Business Review*, pages 29–61, July-August 1975.
15. T.G. Moher, D.C. Mak, B. Blumenthal, and L.M. Leventhal. Comparing the comprehensibility of textual and graphical programs: The case of petri nets. In *Empirical Studies of Programmers: Fifth Workshop*, pages 137–161, 1993.
16. N. Hari Narayanan, Masaki Suwa, and H. Motoda. Behaviour hypothesis from schematic diagrams. In J. Glasgow and B. Chandrasekaran N.H. Narayanan, editors, *Diagrammatic Reasoning*, pages 501–534. The AAAI Press, 1995.
17. C.L. Paris. The Use of Explicit User Models in a Generation System for Tailoring Answers to the User's Level of Expertise. In A. Kobsa and W. Wahlster, editors, *User Models in Dialog Systems*, pages 200–232. Springer-Verlag, 1989.
18. D.A. Scanlan. Structured flowcharts outperform psuedocode: an experimental comparison. *IEEE Computer*, 6(5):28–36, 1989.
19. B. Shneiderman. Direct manipulation: A step beyond programming languages. *Computer*, pages 57–69, August 1983.

20. K.M. Swigger and R.P. Brazile. Experimental comparisons of design/documentation formats for expert systems. *International Journal of Man-Machine Studies*, 31:47–60, 1989.
21. M.R. Wick and W.B. Thompson. Reconstructive expert system explanation. *Artificial Intelligence*, 54:33–70, 1992.
22. M.G. Williams, W.A. Ledder, J.N. Buehler, and J.T. Canning. An empirical study of visual labs. In *Proceedings 1993 IEEE Symposium on Visual Languages*, pages 371–373. IEEE Comput. Soc. Press., 1993.