# What are Visual Programming, Programming by Example, and Program Visualization?

*Brad A. Myers*
Dynamic Graphics Project
Computer Systems Research Institute
University of Toronto
Toronto, Ontario, M5S 1A4
Canada

*ABSTRACT*

There has been a great interest recently in systems that use graphics to aid in the programming, debugging, and understanding of computer programs. The terms "Visual Programming" and "Program Visualization" have been applied to these systems. Also, there has been a renewed interest in using examples to help alleviate the complexity of programming. This technique is called "Programming by Example." This paper attempts to provide more meaning to these terms by giving precise definitions, and then uses these definitions to classify existing systems into a taxonomy.

*RESUME*

Les systèmes qui utilisent l'infographie pour aider à la programmation, à la mise-au-point et à la comprehension de logiciels ont récemment suscité beaucoup d'intérêt. Les termes "programmation visuelle" et "visualisation de programmes" ont été associés à ces systèmes. Il y a aussi eu un renouveau d'intérêt pour l'utilisation d'exemples pour aider à simplifier la programmation. On parle alors de "programmation par exemples". Nous essaierons de définir ces termes avec plus de précision et utiliserons ces définitions comme base pour établir une taxonomie des systèmes disponibles actuellement.

**Key Words and Phrases**: Visual Programming, Program Visualization, Programming by Example, Inferencing, Automatic Programming, Flowcharts, Debugging Aids, Program Synthesis, Documentation, Computer Languages.

**Extended Summary.**

*NOTE: This paper is a summary of [Myers 86].* The reader should refer to that paper for full information.

As the distribution of personal computers and the more powerful personal workstations grows, the majority of computer users now do not know how to program. They buy computers with packaged software and are not able to modify the software even to make small changes. In order to allow the end user to reconfigure and modify the system, the software may provide various options, but these often make the system more complex and still may not address the users' problems. "Easy-to-use" software, such as the "Direct Manipulation" systems [Shneiderman 83] actually make the user-programmer gap *worse* since more people will be able to use the software (since it is easy to use), but the internal program code is now much more complicated (due to the extra code to handle the user interface). Therefore, systems are moving in the direction of providing end user programming. It is well-known that conventional programming languages are difficult to learn and use [Gould 84], requiring skills that many people do not have. In an attempt to make the programming task easier, recent research has been directed towards using graphics. This has been called "Visual Programming" or "Graphical Programming". Some Visual Programming systems have successfully demonstrated that non-programmers can create fairly complex programs with little training [Halbert 84].

Another motivation for using graphics is that it tends to be a higher-level description of the desired actions (often de-emphasizing issues of syntax and providing a higher level of abstraction) and may therefore make the programming task easier even for professional programmers. This may be especially true during debugging, where graphics can be used to present much more information about the program state (such as current variables and data structures) than is possible with purely textual displays. This is one of the goals of Program Visualization. Other Program Visualization systems use graphics to help teach computer programming.

Programming-by-Example is another technology that has been investigated to make programming easier, especially for non-programmers. It involves presenting to the computer examples of the data that the program is supposed to process and using these examples during the development of the program. Many, although not all, Programming-by-Example systems have also used Visual Programming, so these two technologies are often linked.

Recently, there has been a large number of articles about systems that incorporate some or all of these features [Grafton 85][Raeder 85]. Unfortunately, the terms have been used imprecisely[1], and there has not been a comprehensive taxonomy that classifies these systems. This paper summarizes research that attempts fill this gap in the literature. The full results are reported in [Myers 86]. First, the important terms are defined in a precise manner, and then these definitions are used to differentiate some example systems.

There are many systems that could be included in this paper in the various categories, but no attempt has been made to be comprehensive. It is hoped that the selection of systems listed will help the reader understand the intent of the classification system.

**Definitions.**

Programming. What is meant by computer "programming" is probably well understood, but it is important to have a definition that can be used to eliminate some limited systems. In this paper, "program" is defined as "a set of statements that can be submitted as a unit to some computer system and used to direct the behavior of that system" [Oxford 83]. While the ability to compute "everything" is not required, the system must include the ability to handle conditionals and iteration, at least implicitly.

Interactive vs. Batch Any programming language system may either be "interactive" or "batch." A batch system has a large processing delay before statements can be run while they are compiled, whereas an interactive system allows statements to be executed when they are entered. This characterization is actually more of a continuum than a dichotomy since even interactive languages like LISP typically require groups of statements (such as an entire procedure) to be specified before they are executed.

[1]For example, Zloof's Query-By-Example system [Zloof 77 and 81] is not a Programming by Example system.

Visual Programming "Visual Programming" (VP) refers to any system that allows the user to *specify* a program in a two (or more) dimensional fashion. Conventional textual languages are not considered two dimensional since the compiler or interpreter processes it as a long, one-dimensional stream. Visual Programming includes conventional flow charts and graphical programming languages. It does not include systems that use conventional (linear) programming languages to define pictures. This eliminates most graphics editors, like Sketchpad [Sutherland 63].

Program Visualization "Program Visualization" (PV) is an entirely different concept from Visual Programming. In Visual Programming, the graphics is the program itself, but in Program Visualization, the program is specified in the conventional, textual manner, and the graphics is used to illustrate some aspect of the program or its run-time execution. Unfortunately, in the past, many Program Visualization system have been incorrectly labeled "Visual Programming" (as in [Grafton 85]). Program Visualization systems can be divided along two axes: whether they illustrate the *code* or the *data* of the program, and whether they are *dynamic* or *static*. "Dynamic" refers to systems that can show an animation of the program running, whereas "static" systems are limited to snapshots of the program at certain points. If a program created using Visual Programming is to be displayed or debugged, clearly this should be done in a graphical manner, but this would not be considered Program Visualization. Although these two terms are similar and confusing, they have been widely used in the literature, so it was felt appropriate to continue to use the common terms.

Programming by Example The term "Programming by Example" (PBE) has been used to describe a large variety of systems. Some early systems attempted to create an entire program from a set of input-output pairs. Other systems require the user to "work through" an algorithm on a number of examples and then the system tries to *infer* the general program structure. This is often called "automatic programming" and has generally been an area of Artificial Intelligence research.

Recently, there have been a number of systems that require the user to specify everything about the program (there is no inference involved), but the user can work out the program on a specific example. The system executes the user's commands normally, but remembers them for later re-use. Bill Buxton coined the phrase "Programming *with* Examples" to more accurately describe these systems. Halbert [84] characterizes Programming with Examples as "Do What I Did" whereas inferential Programming by Example might be "Do What I Mean". The term "Programming by Example" will be used to include both inferencing systems and Programming With Example systems.

Of course, whenever code is executed in any system, test data must be entered to run it on. The distinction between normal testing and "Programming *with* Examples" is that in the latter the system requires or encourages the specification of the examples *before* programming begins, and then applies the program as it develops to the examples. This essentially requires all Programming-*with*-Example systems (but not Programming-by-Example systems with inferencing) to be interactive.

## Taxonomy of Programming Systems.

This paper presents two taxonomies. The first is for systems that support programming. The second taxonomy is for systems that use graphics *after* the programming process is finished (Program Visualization systems).

A meaningful taxonomy can be created by classifying *programming systems* into eight categories using the orthogonal criteria of

- Visual Programming or not,
- Programming by Example or not, and
- Interactive or batch.

Of course, a single system may have features that fit into various categories and some systems may be hard to classify, so this paper attempts to characterize the systems by their most prominent features. Figure 1 shows the division with some sample systems.

## Taxonomy of Program Visualization Systems.

The systems listed below are not *programming* systems since code is created in the conventional manner. Graphics in these are used to *illustrate* some aspect of the program after it is written. Figure 2 shows some Program Visualization systems classified by whether they attempt to illustrate the code or the data of a program (some provide both), and whether the displays are static or dynamic.

## Conclusions.

Visual Programming, Programming by Example and Program Visualization are all exciting areas of active computer science research, and they promise to improve the user interface to programming environments. A number of interesting systems have been created in each area, and there are some that cross the boundaries. This paper has attempted to classify some of these systems in hopes that this will clarify the use of the terms and provide a context for future research.

ACKNOWLEDGEMENTS
For help and support of this article, I would like to thank Bill Buxton, Ron Baecker, Bernita Myers, and many others at the University of Toronto. The research described in this paper was partially funded by the National Science and Engineering Research Council (NSERC) of Canada.

**Not Programming by Example**

| | Batch | Interactive |
|---|---|---|
| Not VP | All Conventional Languages: Pascal, Fortran, etc. | LISP, APL, etc. |
| VP | Grail [Ellis 69]<br>AMBIT/G/L [Christensen 68,71]<br>Query by Example [Zloof 77, 81]<br>FORMAL [Shu 85]<br>GAL [Albizuri-Romero 84] | Graphical Program Editor [Sutherland 66]<br>PIGS [Pong 83]<br>Pict [Glinert 84]<br>PROGRAPH [Pietrzykowski 83,84]<br>State Transition UIMS [Jacob 85] |

**Programming by Example**

| | Batch | Interactive |
|---|---|---|
| Not VP | I/O pairs* [Shaw 75] | Tinker [Lieberman 82] |
| VP | [Bauer 78] traces* | AutoProgrammer* [Biermann 76]<br>Pygmalion [Smith 77]<br>Graphical Thinglab [Borning 86]<br>SmallStar [Halbert 81,84]<br>Rehearsal World [Gould 84] |

**Figure 1.**
Classification of programming systems by whether they are visual or not, whether they have Programming by Example or not, and whether they are interactive or batch. Starred systems (*) have inferencing, and non-starred PBE systems use Programming With Example.

| | Static | Dynamic |
|---|---|---|
| Code | Flowcharts [Haibt 59]<br>SEE Visual Compiler [Baecker 86]<br>PegaSys [Moriconi 85] | BALSA [Brown 84]<br>PV Prototype [Brown 85] |
| Data | TX2 Display Files [Baecker 68]<br>Incense [Myers 80,83] | Two Systems [Baecker 75]<br>Sorting out Sorting [Baecker 81]<br>BALSA [Brown 84]<br>Animation Kit [London 85]<br>PV Prototype [Brown 85] |

**Figure 2.**
Classification of Program Visualization Systems by whether they illustrate code or data, and whether they are dynamic or static.

REFERENCES

[Albizuri-Romero 84] Miren B. Albizuri-Romero. "GRASE--A Graphical Syntax-Directed Editor for Structured Programming," *SIGPLAN Notices.* **19(2)** Feb. 1984. pp. 28-37.

[Attardi 82] Giuseppe Attardi and Maria Simi. "Extending the Power of Programming by Example," *SIGOA Conference on Office Information Systems*, Philadelphia, PA, Jun. 21-23, 1982. pp. 52-66.

[Baecker 68] R.M.Baecker. "Experiments in On-Line Graphical Debugging: The Interrogation of Complex Data Structures," (Summary only) *First Hawaii International Conference on the System Sciences.* Jan. 1968. pp. 128-129.

[Baecker 75] R.M.Baecker. "Two Systems which Produce Animated Animated Representations of the Execution of Computer Programs," *SIGCSE Bulletin.* **7**(1) Feb. 1975. pp. 158-167.

[Baecker 81] Ron Baecker. *Sorting out Sorting.* 16mm color, sound film, 25 minutes. Dynamics Graphics Project, Computer Systems Research Institute, University of Toronto, Toronto, Ontario, Canada. 1981. Presented at ACM SIGGRAPH'81. Dallas, TX. Aug. 1981.

[Baecker 86] Ronald Baecker and Aaron Marcus. "Design Principles for the Enhanced Presentation of Computer Program Source Text," *Human Factors in Computing Systems: Proceedings SIGCHI'86.* Boston, MA. Apr. 13-17, 1986.

[Bauer 78] Michael A. Bauer. *A Basis for the Acquisition of Procedures.* PhD Thesis, Department of Computer Science, University of Toronto. 1978. 310 pages.

[Biermann 76] Alan W. Biermann and Ramachandran Krishnaswamy. "Constructing Programs from Example Computations," *IEEE Transactions on Software Engineering.* **SE-2(3)** Sept. 1976. pp. 141-153.

[Borning 86] Alan Borning. "Defining Constraints Graphically," *Human Factors in Computing Systems: Proceedings SIGCHI'86.* Boston, MA. Apr. 13-17, 1986.

[Brown 84] Marc H. Brown and Robert Sedgewick. "A System for Algorithm Animation," *Computer Graphics: SIGGRAPH'84 Conference Proceedings.* Minneapolis, Minn. **18(3)** July 23-27, 1984. pp. 177-186.

[Brown 85] Gretchen P. Brown, Richard T. Carling, Christopher F. Herot, David A. Kramlich, and Paul Souza. "Program Visualization: Graphical Support for Software Development," *IEEE Computer.* **18(8)** Aug. 1985. pp. 27-35.

[Christensen 68] Carlos Christensen. "An Example of the Manipulation of Directed Graphs in the AMBIT/G Programming Language," in *Interactive Systems for Experimental Applied Mathematics*, Melvin Klerer and Juris Reinfelds, eds. New York: Academic Press, 1968. pp. 423-435.

[Christensen 71] Carlos Christensen. "An Introduction to AMBIT/L, A Diagramatic Language for List Processing," *Proc. 2nd Symposium on Symbolic and Algebraic Manipulation.* Los Angeles, CA. Mar. 23-25, 1971. pp. 248-260.

[Ellis 69] T.O. Ellis, J.F. Heafner and W.L. Sibley. *The Grail Project: An Experiment in Man-Machine Communication.* RAND Report RM-5999-Arpa. 1969.

[Glinert 84] Ephraim P. Glinert and Steven L. Tanimoto. "Pict: An Interactive Graphical Programming Environment," *IEEE Computer.* **17(11)** Nov. 1984. pp. 7-25.

[Gould 84] Laura Gould and William Finzer. *Programming by Rehearsal.* Xerox PARC TR SCL-84-1. May, 1984. 133 pages. Excerpted in *Byte.* **9(6)** June, 1984.

[Grafton 85] Robert B. Grafton and Tadao Ichikawa, eds. *IEEE Computer*, Special Issue on Visual Programming. **18(8)** Aug. 1985.

[Haibt 59] Lois M. Haibt. "A Program to Draw Multi-Level Flow Charts," *Proceedings of the Western Joint Computer Conference.* San Francisco, CA. **15** Mar. 3-5, 1959. pp. 131-137.

[Halbert 81] Daniel C. Halbert. *An Example of Programming by Example.* Masters of Science Thesis. Dept. of EE&CS, University of Calif., Berkeley and Xerox Corporation Office Products Division, Palo Alto, CA. June, 1981. 55 pages.

[Halbert 84] Daniel C. Halbert. *Programming by Example.* PhD Thesis. Computer Science Division, Dept. of EE&CS, University of California, Berkeley. 1984. Also: Xerox Office Systems Division, Systems Development Department, TR OSD-T8402, December, 1984. 83 pages.

[Jacob 85] Robert J.K. Jacob. "A State Transition Diagram Language for Visual Programming," *IEEE Computer.* **18(8)** Aug. 1985. pp. 51-59.

[Lieberman 82] Henry Lieberman. "Constructing Graphical User Interfaces by Example," *Graphics Interface'82*, Toronto, Ont. Mar. 17-21, 1982. pp. 295-302.

[London 85] Ralph L. London and Robert A. Druisberg. "Animating Programs in Smalltalk," *IEEE Computer.* **18(8)** Aug. 1985. pp. 61-71.

[Moriconi 85] Mark Moriconi and Dwight F. Hare. "Visualizing Program Designs Through PegaSys," *IEEE Computer.* **18(8)** Aug. 1985. pp. 72-85.

[Myers 80] Brad A. Myers. *Displaying Data Structures for Interactive Debugging.* Xerox Palo Alto Research Center Technical Report CSL-80-7. June, 1980. 97 pages.

[Myers 83] Brad A. Myers. "Incense: A System for Displaying Data Structures," *Computer Graphics: SIGGRAPH '83 Conference Proceedings.* **17(3)** July 1983. pp. 115-125.

[Myers 86] Brad A. Myers. "Visual Programming, Programming by Example, and Program Visualization; A Taxonomy." *Proceedings SIGCHI'86: Human Factors in Computing Systems.* Boston, MA. April 13-17, 1986.

[Oxford 83] *Dictionary of Computing.* Oxford: Oxford University Press, 1983.

[Pietrzykowski 83] Thomas Pietrzykowski, Stanislaw Matwin, and Tomasz Muldner. "The Programming Language PROGRAPH: Yet Another Application of Graphics," *Graphics Interface'83*, Edmonton, Alberta. May 9-13, 1983. pp. 143-145.

[Pietrzykowski 84] T. Pietrzykowski and S. Matwin. *PROGRAPH: A Preliminary Report.* University of Ottawa Technical Report TR-84-07. April, 1984. 91 pages.

[Pong 83] M.C. Pong and N. Ng. "Figs--A System for Programming with Interactive Graphical Support," *Software--Practice and Experience.* **13(9)** Sept. 1983. pp. 847-855.

[Raeder 85] Georg Raeder. "A Survey of Current Graphical Programming Techniques," *IEEE Computer.* **18(8)** Aug. 1985. pp. 11-25.

[Shaw 75] David E. Shaw, William R. Swartout, and C. Cordell Green. "Inferring Lisp Programs from Examples," *Fourth International Joint Conference on Artificial Intelligence.* Tbilisi, USSR. Sept. 3-8, 1975. 1 pp. 260-267.

[Shneiderman 83] Ben Shneiderman. "Direct Manipulation: A Step Beyond Programming Languages," *IEEE Computer.* **16(8)** Aug. 1983. pp. 57-69.

[Shu 85] Nan C. Shu. "FORMAL: A Forms-Oriented Visual-Directed Application Development System," *IEEE Computer.* **18(8)** Aug. 1985. pp. 38-49.

[Smith 77] David C. Smith. *Pygmalion: A Computer Program to Model and Stimulate Creative Thought.* Basel, Stuttgart: Birkhauser, 1977. 187 pages.

[Sutherland 63] Ivan E. Sutherland. "SketchPad: A Man-Machine Graphical Communication System," *AFIPS Spring Joint Computer Conference.* **23** 1963. pp. 329-346.

[Sutherland 66] William R. Sutherland. *On-line Graphical Specification of Computer Procedures.* MIT PhD Thesis. Lincoln Labs Report TR-405. 1966.

[Zloof 77] Moshe M. Zloof and S. Peter de Jong. "The System for Business Automation (SBA): Programming Language," *CACM.* **20(6)** June, 1977. pp. 385-396.

[Zloof 81] Moshe M. Zloof. "QBE/OBE: A Language for Office and Business Automation," *IEEE Computer.* **14(5)** May, 1981. pp. 13-22.