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, debug­
ging, and understanding of computer programs. The
terms “Visual Programming” and “Program Visuali­
zation” 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.

Extended Summary.
NOTE: This paper is a summary of [Myers 86].
The reader should refer to that paper for full informa­
tion.

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] actu­
ally 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 han­
dle the user interface). Therefore, systems are mov­
ing in the direction of providing end user program­
manship. It is well-known that conventional program­
ning 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 Program­
ning 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 Pro­
gram Visualization systems use graphics to help teach
computer programming.

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

Graphics Interface '86 Vision Interface '86
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, 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.

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].

Programming-by-Example. "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.

---

Graphics Interface '86  Vision Interface '86

---

Footnote:

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

---

**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.

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


