

Metrics for Visual Software Development

Initial Research and Findings

Paul A. Szulewski, *Mercury Computer Systems*
Faye C. Budlong, *Draper Laboratory*

This article provides a summary of recent research that investigated the use of visual languages (VLs) and visual programming environments (VPEs). The study reviewed the state of the practice for developing software using VLs and managing these development activities. The study concluded that there is little evidence of the use of mature practices and recommends candidate metrics for VLs and VPEs as a first step toward a method to estimate the effort required to develop software using VLs and VPEs.

Increased demand for reliable and useful software applications has led to generations of advancements in software languages and development environments. Examples include

- The evolution from early languages, such as assembly language, to higher-order languages and fourth-generation languages.
- The development and implementation of frameworks or software engineering environments that are populated with any number of productivity tools.
- The development of graphical (or visual) front-ends for existing computer languages called VPEs.
- The development and use of VLs that allow developers to generate applications entirely within a visual environment.

Currently, the use of VPEs and VLs for general-purpose programming is undergoing such rapid adoption that it could be called a visual explosion. Applications developed using VPEs and VLs are developed rapidly and differently from applications based entirely on textual languages. It is important to understand these differences and to approach managing projects that use VPEs and VLs in a way that will allow effective project control, i.e., delivering quality software on time and within budget without interfering with the advantages inherent in the use of visual tools.

Definitions

The following definitions are used to help form a context for the tools used to build applications visually.

- **Visual Language** – A computer language that uses a visual syntax, such as pictures or forms, to express programs. Text can be part of a visual syntax.
- **VL Taxonomy** – A system to classify VLs.
- **Visual Programming** – Software development that uses a visual representation of the software and allows developers to create software through managing and manipulating objects on a visual palette. Also called graphical programming.
- **Visual Programming Environment** – The graphical user interface (GUI) and graphical tools that are used to manage and manipulate objects on a visual palette, construct programs, interface with other software, manage the software, and execute the software.

This article provides a summary of recent research concerning the use of VLs and VPEs. The study reviewed the state of the practice for developing software using VLs and managing the development activities. Our research revealed that there is little evidence of the use of mature practices and recommends candidate metrics for VLs and VPEs as a first step toward a method to estimate the effort required to develop software using VLs and VPEs.

Purpose of the Research

VLs and VPEs are being studied to learn how to estimate and manage software development using these new languages and environments. The goals of this initial research are to

- Identify “countables” or metrics related to VL and VPE development processes and software products.
- Develop an estimation model for software developed visually, i.e., using VLs and VPEs.

VLs and VPEs are of interest because they are presently being used to develop real applications in a range of sizes and degrees of criticality. Examples include

- GUI and GUI-related application development.
- Database search engines, e.g., visual query languages.
- Data capture and maintenance.
- Real-time data presentation.
- Space-qualified guidance, navigation, and control.
- Other real-time control systems, including aerospace and automotive applications.

With all this activity, little evidence has been found that mature practices are being used to manage development using VLs and VPEs. These types of languages are reported to be “fun to use,” and the literature has yet to address the management issues that may be involved in moving from a textual model to a visual model of software development.

In addition, no evidence was found that groups using VLs and VPEs use a repeatable method to estimate development cost, effort, size, or schedule. The issues of developing large-scale applications where formal estimates and management tracking are important have only recently been addressed at any level, and the research is still in its infancy.

Initial Results

The research focused on six specific areas:

- Finding definitions for visual languages.
- Identifying examples of commercially available VL and VPE products.
- Identifying published productivity gains and other benefits of using VLs and VPEs.
- Finding evidence of VL and VPE use in government software applications.
- Examining current VL-related measurement work.
- Identifying potential metrics for VL and VPE development.

Examples

Tables 1 and 2 provide a limited set of examples of commercially available VLs and VPEs, respectively. The examples provide comparisons between VLs and VPEs, e.g., the output from a VPE generally is code for a specific textual programming language, and they indicate the variety of domains currently served by VLs and VPEs.

Advantages and Disadvantages

Developing software visually has a number of advantages and disadvantages, which are summarized in the following paragraphs. Support for the advantages regarding quick results and potential increases in developer productivity are documented in the literature on visual programming. For example,

- An empirical study reports that it is easier to write programs visually than textually [1].
- Comparative studies report that there is a four to 10 times produc-

Visual Programming Environment	Computer Language Output	Vendor	Domains
Virtual Programmer	C++ and Ada95	VZCORP	General Purpose Component-Based Development
MatrixX	C, Ada95, Proprietary Scripting Language	ISI	Control Systems
VXP	Motif	Free from OSF	GUI Builder for X Applications
Café	Java	Symantec	Internet and Intranet Applications

Table 2. *Examples of commercially available VPEs.*

tivity gain over traditional programming techniques when working in a visual environment [2].

- A more recent empirical study concludes that visual representation improves human performance [3].

Advantages

Advantages gained through using VLs and VPEs include the following:

- They provide an opportunity for domain engineers, rather than software engineers, to develop software applications.
- Visual communication is intuitive—visual communication uses pictures rather than words (code).
- They provide quick initial results—you can examine the results sometimes within hours rather than months.
- Rather than using formal specifications to guide development, they provide a means to implement participatory development approaches using prototyping techniques and “conversations.”
- They take advantage of powerful workstations and tools by providing

the capability to work with pictures rather than words.

- They provide the potential to increase software development productivity.
- They provide the potential to lower lifecycle costs.

Disadvantages

There are some potential problems that may be expected from using VLs and VPEs. These disadvantages are derived from discussions with managers and software developers who work with VLs and VPEs.

- VLs and VPEs require a new way of doing business throughout the software lifecycle, including development, test, acceptance, and maintenance—the rules have been significantly changed.
- Programmers (or software engineers) are not required; however, the quality of software produced by domain engineers may be suspect. (It is too easy to jump right in and program.)
- No industry standards are in place to control the visual languages and environments. More traditional languages, e.g., C and Ada, are standardized through concurrence of members of the software engineering community and maintenance by standards organizations. This control does not yet exist for VLs and VPEs.
- Little or no formal qualification is done for new applications because of the lack of specifications and known requirements.
- Often, especially for VLs, the bindings to other languages are weak or nonexistent.

Table 1. *Examples of commercially available VLs.*

Language	Vendor	Domains
LabVIEW	National Instruments	Data Acquisition, Analysis, and Display
Prograph/Pictorius Net Builder	Pictorius	Macintosh Applications
Visual AppBuilder	Novell	Windows Server Applications
VEE	Hewlett Packard	Test Equipment
PowerBuilder	Austin Software Foundry	Windows Applications

- Configuration control is not often considered, and the visual representations may be difficult to control using current commercially available configuration management tools.
- Development issues can be deceptively complex.
- The apparent ease of use for these tools invites abuse.

New Development Models

New software development models are rapidly evolving as VL and VPE applications become more accepted. In general, these models are typified as being highly participatory with developers and users or other domain experts working closely to develop each application. The application tends to be its own “specification” where little upfront documentation is developed and “approved” in the traditional sense of approval.

Participatory development styles tend to involve developers, users, and other stakeholders in several ways, including

- A conversation model where the software developer and user work together with the computer to interactively build an application [4].
- “Memos and demos” that allow multiple iterations with documented output, high user visibility, and minimal specification.
- Evolutionary development using an integrated small “hot team” that consists of software developers, domain engineers, and other stake-

holders to concurrently develop an application and gain approval of it.

These approaches show many similarities with rapid prototyping, including strong user (or customer) interaction during development. The software is developed, used, and refined as necessary, based on lessons learned, rather than waiting for traditional qualification or validation.

In general, formal milestones, e.g., requirements and design reviews, often are either missing or ill-defined. There are often no (or limited) formal reviews. Requirements and design are implicit in an acceptable application. Usually, the electronic design as implemented is the only representation of the application. There is often either limited or no formal testing. The project is done when the user and the developer agree that it is, or when the money runs out.

Critical government application development using VLs and VPEs have a somewhat different approach—attempts have been made to integrate evolutionary development with formal documentation and decision points. However, the concept of application development and testing appears to need further refinement. Some of the questions that should be addressed to provide confidence in these applications include

- What is a visual software “unit”?
- How detailed are the requirements?
- How do you verify software for critical applications?
- Is there a new concept of complexity?

Evidence of Government Use

Table 3 provides examples of government agencies that have used VLs or VPEs to help meet their software needs, the name of the VL or VPE used, the application domain in which the VL or VPE is used, and a brief description of the program or application area. Many other examples could be cited, but these provide an indication of the breadth of government applications being developed using VLs and VPEs.

VL-Related Software Measurement

Some inroads have been made into defining measures that are applicable to VLs and VPEs. Empirical information has been gathered, as previously discussed, and some related studies have been completed. In addition, some commercial information has been developed that may be applicable to software developed visually. Examples include

- Studies such as Jeffrey V. Nickerson’s “Visual Programming” [5] and E. Glinert’s “Towards Software Metrics for Visual Programming” [6].
- Commercial information such as “Project Management for OO Development” [7] and “Counting a GUI Application” [8].

This information leads to the conclusion that a number of “countables” can be defined to support definition of VL metrics. Candidate countables are discussed in the next section.

Candidate Metrics for VLs and VPEs

The countable items currently being considered as candidates for further research fall into four categories. Examples of each of these categories along with possible advantages and disadvantages follow.

Physical Measures

Physical measures are measures of the outputs from the development effort. Those identified include the following:

- **Run-time memory size**, e.g., kilobytes or megabytes of memory.
Advantages: Provides hard data that can be compared to applications

Table 3. Evidence of government applications using visual languages.

Government Agency	VL or VPE	Domain	Description
NASA/JPL	LabVIEW	Data Analysis	Telemetry Data Analyzer for Galileo Mission
U.S. Army	LabVIEW	Data Analysis	Graphical User Interface
U.S. Air Force	LabVIEW	Instrument Data Analysis	Stand-Alone Instrumentation Evaluation Tool
NASA/JPL	VEE	Instrument Control	Software to Support the Test of Flight Electronics
NASA	MatrixX	Control Systems	International Space Station
U.S. Air Force	Virtual Programmer Ada95	Ada Components	Ada Joint Program Office-Sponsored VPE validation

built using traditional textual languages.

Disadvantages. May not correlate well with effort for applications that need to be extremely efficient, e.g., real-time embedded systems with processor limitations. Size of application could grow substantially with unnecessary features, use of interpretive (rather than compiled) languages, etc.

- **Processor(s) utilization**, e.g., cycle time, number of cycles used, and percent of processor resources required to run an application.

Advantages. Provides hard data that can be compared to applications built using traditional textual languages.

Disadvantages. May not correlate well with effort for applications that need to be extremely efficient, e.g., real-time embedded systems with processor limitations. Size of application could grow substantially when processor utilization is not considered to be application critical, e.g., for data systems or other systems where memory and processing time do not need to be optimized. Interpretive language applications generally use significantly more processing resources than compiled applications, may be much easier to develop and verify, and may provide substantially less functionality when compared with compiled counterparts.

- **Source lines of code (SLOC) equivalents**, e.g., high-level language SLOC outputs from a VPE and functional cell contents from a spreadsheet.

Advantages. SLOC are still the most used indicators of application size and allow data to be normalized based on an understandable concept. The concept of SLOC is generally understandable to software managers.

Disadvantages. The concept of SLOC may not be in any way applicable to some VLs. A clear definition of SLOC needs to be used consistently to obtain consistent results. Where SLOC are automatically generated from a VPE, derived

measures such as descriptiveness may not be useful or applicable.

There are likely to be differences in SLOC output depending on whether the count is of automatically generated code from a VPE or hand-generated script code that may be an adjunct to the visual aspects of a VL or VPE.

Countables in the Visual Medium

These items are entities in the physical design representation. They include

- **Objects** (number, semantic complexity), e.g., items on a diagram, number of diagrams, and complexity of the content of a diagram or item on a diagram.

Advantages. Objects can be visually examined and counted. Within a single language or environment, object counts should yield repeatable results across several applications. This metric should help to quantify effort and schedule when combined with other measures such as number of connectors, number of interconnections, and some concept of inheritance. Some work already has been completed on complexity of applications developed with VLs.

Disadvantages. May not be comparable across languages or environments. May not be easy to estimate until a design is well under way.

- **Connectors** (number, data complexity, control complexity), e.g., connectors between items on a diagram or indicating interfaces to items on connecting diagrams.

Advantages. Connectors can be visually examined and counted. Within a single language or environment, connector counts should yield repeatable results across several applications. This metric should help to quantify effort and schedule when combined with other measures such as number of objects and some concept of bandwidth.

Disadvantages. May not be comparable across languages or environments. May not be easy to estimate until a design is well under way.

OO-Related Measures

These items include measures that have been developed for object-oriented (OO) applications. There is an inherent assumption in these measures that applications developed visually use an extended concept of object orientation. Thus, the candidate measures include

- **Inheritance**, e.g., depth of inheritance and number of children within a class.

Advantages. Can be counted in a design medium. If OO development techniques are used, will provide one of the primary OO measures of complexity.

Disadvantages. Provides a secondary input to estimation needs. Provides a measure of complexity more than a measure of size. May be useful to support estimates of test effort for an OO application. VL development may not use OO techniques.

- **Encapsulation**, e.g., measures of how well a class (with its subclasses) provides information hiding and consistent object representation from a single (or minimal number of) source(s). Examples are lack of cohesion in methods or coupling between classes.

Advantages. Measures of encapsulation provide an indication of the quality and maintainability of an OO application. Can be counted in a design medium. Also provide an indication of the effort required to test an application thoroughly.

Disadvantages. Provides a secondary input to estimation needs. Provides measures of design quality, understandability, and complexity more than measures of size. May be useful to support estimates of test effort for an OO application. VL development may not use OO techniques.

- **Number of interconnections**, e.g., counts of “uses” and “used by” for a class or all classes within an application. Also could be counts of interfaces with external items.

Advantages. Combined with number of classes in an application, provides a primary indication of application size and a “quick” estimate of application complexity. Can be counted

in a design medium. Probably most useful for estimate refinement during design. Could be useful for visual applications that do not use OO techniques.

Disadvantages: May not be available early enough in the software life-cycle to support effort estimation prior to the completion of a design. May be best used for estimate refinement during development or to estimate the effort required for maintenance activities.

Function Point-Related Measures

Function points have been developed and used successfully for a number of years. Classical function points and extensions to function points could be applicable to estimates of effort for applications developed using VLs and VPEs. The applicable measures could include

- **Function points**, as defined in the International Function Point Users Group counting practices manual [8].

Advantages: Provides a well-documented and understood approach to derive estimates of size, effort, and schedule for software applications. Can be counted in a design medium. Although function points have been shown to be useful in the information systems domain, some advocates claim that extensions, such as object points and feature points, can be adapted for OO and real-time applications.

Disadvantages: May not be available early enough in the software life-cycle to support effort estimation until a reasonable amount of time has been expended on design. For maintenance, there has been little success with the development of any automated code analysis tool that can count function points in a completed application. Function point counting is complex and probably will need some adaptation for VL applications.

- **Object points**, e.g., counts of objects in an OO development environment.

Advantages: Can be counted in a design medium. Can be used to develop size estimates for OO applications. May be useful for VL applications that do not use OO development techniques. May be useful in combination with counting objects on a visual palette.

Disadvantages: May not be available early enough in the software life-cycle to support effort estimation until a reasonable amount of time has been expended on design.

Counts of abstract objects and their utility to estimate VL or VPE applications is unclear.

- **Feature points**, e.g., extensions to function points to account for the effort required to implement algorithms for real-time applications.

Advantages: Provides a well-documented approach to derive estimates of size, effort, and schedule for software applications that have real-time constraints.

Disadvantages: Requires interpolation and may need to be combined with other metrics, e.g., SLOC estimates, to incorporate the algorithmic information necessary to develop cost and effort estimates. Not easy to define or implement.

May not be available early enough in the software lifecycle to support effort estimation until a reasonable amount of time has been expended on design. May not be "countable" in completed applications.

Next Steps

This research has identified a gap in the state of software development practice for estimation and measurement. Several of the practitioners of VLs and VPEs we contacted in the course of this research (including government organizations, academia, consultants, and industry) share our interest in continuing this work and have expressed the desire to form a special interest group or consortium.

We are actively seeking sponsorship and collaborators to continue this work. We have a plan to develop, using the combined expertise of our collaborators, and verify a metrics-based effort

estimation model for VLs and VPEs. Once the estimation model is developed and validated, the technology will be made available to the software community at large. ♦

Acknowledgments

This work was funded by the U.S. Air Force Embedded Computer Resources Support Improvement Program (ESIP).

During the course of this research, we knocked on many doors to obtain the information we required to perform this research. To our surprise, we found many doors open and with "welcome" signs up. We acknowledge the interest and support of

- Lt. Col. Joe Jarzombek (U.S. Air Force), ESIP director.
- Bruce Allgood, ESIP office.
- Maj. Joe Stanko (U.S. Air Force), Office of the Secretary of the Air Force for Acquisition.
- Joe Kochocki, Draper Laboratory.
- Margret Burnett, Oregon State University.
- Stan Colby, VZ Corp.
- Ed Baroth, Jet Propulsion Laboratory.
- Larry Putnam Jr., Quantitative Software Measurement.

About the Authors



Paul A. Szulewski has held a variety of engineering-related technical and management positions since 1973. He is currently technical program manager for engineering at Mercury Computer Systems, Inc. of Chelmsford, Mass. Prior to his current position, he was at The Charles Stark Draper Laboratory, Inc. for nearly 20 years and with Sanders Associates for five years. He specializes in project planning, measurement, assessment, and process definition. He is a founding member of the National Software Council, now known as the Center for National Software Studies. He is a distinguished reviewer for *IEEE Software* and the Pentagon acquisition staff. He has pioneered research in software metrics and evaluation methods for products, processes, and organizations.

see *METRICS*, page 30

Chicago, IL 60631
Voice: 773-467-2673
Fax: 773-594-2618
E-mail: rag@safco.com



C.R. Carlson holds a doctorate from the University of Iowa. He is a professor in the computer science department at Illinois Institute of Technology. He has published extensively in the fields of database design, information architecture, and software engineering. His research interests include object-oriented modeling, design and query languages, and software process issues.

Computer Science Department
Illinois Institute of Technology
10 West 31st Street
Chicago, IL 60616
Voice: 312-567-5152
Fax: 312-567-5067

References

1. Burnstein, I., T. Suwanassart, and C.R. Carlson, "The Development of a Testing Maturity Model," *Proceedings of the Ninth International Quality Week Conference*, San Francisco, May 21-24, 1996.
2. Burnstein, I., T. Suwanassart, and C.R. Carlson, "Developing a Testing Maturity Model," *CROSSTALK*, Software Technology Support Center, Hill Air Force Base, Utah; Part I: August 1996, pp. 21-24; Part II: September 1996, pp. 19-26.
3. Paulk, M., C. Weber, B. Curtis, and M. Chrissis, *The Capability Maturity Model: Guideline for Improving the Software Process*, Addison-Wesley, Reading, Mass., 1995.
4. Zubrow, D., W. Hayes, J. Siegel, and D. Goldenson, "Maturity Questionnaire," Technical Report, Software Engineering Institute, CMU/SEI-94-SR-7, June 1994.
5. Masters, S. and C. Bothwell, "A CMM Appraisal Framework, Version 1.0," Technical Report, Software Engineering Institute, CMU/SEI-95-TR-001, February 1995.
6. ISO/IECJTC1/WG10, "SPICE Products," Technical Report, Type 2, June 1995.
7. Homyen, A., "An Assessment Model to Determine Test Process Maturity," Diss., Illinois Institute of Technology, July 1998.
8. Puffer, J. and A. Litter, "Action Planning," *IEEE Software Engineering Technical Council Newsletter*, Vol. 15, No. 2, pp. 7-10.
9. Grom, R., "Report on a TMM Assessment Support Tool," Technical Report, Illinois Institute of Technology, April 1998.

METRICS, from page 25

Mercury Computer Systems
199 Riverneck Road
Chelmsford, MA 01824-2820
Voice: 978-256-0052 ext. 320
Fax: 978-256-3599
E-mail: paulski@mc.com
Internet: <http://www.mc.com>



Faye C. Budlong is a principal member of the technical staff at The Charles Stark Draper Laboratory, Inc. in Cambridge, Mass. She provides expertise in software standards development and application, software product evaluations, software requirements analysis, standard-compliant software, and document development within Draper Laboratory. She has a bachelor's degree in mathematics from Roger Williams University in Bristol, R.I. and a master's degree in education from Northeastern University in Boston, Mass.

The Charles Stark Draper Laboratory, Inc.
555 Technology Square
Cambridge, MA 02139
Voice: 617-258-2054
Fax: 617-258-3939
E-mail: budlong@draper.com
Internet: <http://www.draper.com>

References

1. Pandey, R.K. and M. Burnett, "Is It Easier to Write Matrix Manipulation Programs Visually or Textually? An Empirical Study,"

- Proceedings of the 1993 IEEE Symposium on Visual Languages (VL93)*, 1993, pp. 344-351.
2. Baroth, E. and C. Hartsough, "Visual Programming in the Real World," *Visual Programming*, M. Burnett, et al., eds., Manning Publications, 1995.
3. Whitley, K.N., "Visual Programming Languages and the Empirical Evidence For and Against," *Journal of Visual Languages and Computing*, October 1996.
4. Baroth, E. and C. Hartsough, "Visual Programming Improves Communication Among the Customer, Developer and Computer," Presentation at National Instruments User Symposium, 1995.
5. Nickerson, J.V., *Visual Programming*, Diss., New York University, New York, N.Y., 1994.
6. Glinert, E., "Towards Software Metrics for Visual Programming," *International Journal of Man-Machine Studies*, Vol. 330, Academic Press, 1989, pp. 425-445.
7. *Project Management for Object-Oriented Development*, Austin Software Factory, Austin, Texas, 1996.
8. *Function Point Counting Practices Manual*, International Function Point Users Group, Waterville, Ohio, Version 4, January 1994.

Note

1. Domain engineers are subject matter experts and may include, for example, mathematicians and control engineers.