# Block-based versus Flow-based Programming for Naive Programmers

Dave Mason
*School of Computer Science*
*Ryerson University*
Toronto, Canada
dmason@ryerson.ca

Kruti Dave
*School of Computer Science*
*Ryerson University*
Toronto, Canada
kruti.dave@ryerson.ca

***Abstract*—There is wide consensus that most people should have some programming capability, whether to control the Internet of Things, or to analyse the world of data around them. While some people focus on teaching conventional text-based languages like Javascript or Python, there is evidence that visual programming languages are more accessible for naive programmers.**

**Visual programming languages fall into two broad categories: block-based, imperative programming, or flow-based, functional programming. However there have not been empirical studies to evaluate the relative merits of the two categories.**

**This paper describes a study of hundreds of random people via Amazon Mechanical Turk attempting to program some simple problems in one or the other of two environments designed to be as similar as possible, except for the choice of paradigm.**

***Index Terms*—Flow-based programming, Block-based programming, Visual Programming Language, Naive programmers**

## I. Introduction

We are creating a visual programming environment for sporadic users/programmers - not necessarily novice, but not classically trained. Rather than multi-media, sprite-based exploratory programming, we are interested in democratizing the vast swaths of (open and proprietary) data becoming available with a programming-for-the-rest-of-us environment to access, analyse, and extract meaning from that data. The environment is deployed in Javascript. Versions will be available for the web as well as iOS and Android devices, and in the longer term we plan to also run it in a back-end mode on servers.

We aim to facilitate ordinary citizens posing problems like: "I wonder if there is a correlation between street lighting and the occurence of potholes."; discovering a city database of potholes and a power company listing of streetlight locations and writing a short program to extract and correlate the data and produce a nice little infographic that answers the question. In the coming data-heavy world, we believe such programming will become even more ubiquitous than experimenting with spreadsheets is today, and we believe it is important to have a fairly powerful programming environment to allow citizens to go beyond canned answers to questions that someone else has determined are important. Our assumption is that, like spreadsheets, while there will be "power"-users who spend much of their time using the environment, most users will use it on the order of an hour a day.

We want to explore the design space by considering a block-based language loosely similar to Scratch[1] or Boxer[2], and a dataflow-based language similar to Pure Data[3] and others[4], [5], [6].

The rest of this paper is organized as follows: §II describes the competing models for visual programming; §III describes the experiment we ran, and the environments we constructed; §IV describes related work; and §V gives a few conclusions and describes some future work.

## II. Programming Models

Historically, the first high-level programming languages were textual based and provided an imperative programming model that mimicked the preceding assembly language and the hardware that executed the programs. Over time, other textual programming models developed: functional, object-oriented, constraint, etc.

Similarly, as visual programming languages were developed, they aligned with the same models. Object-oriented visual programming languages are less popular than their text-based equivalent. Most visual programming languages fall into two camps:

1) imperative languages: these are the block languages and they are reasoned about as mutating state; and
2) functional languages: these are the flow-based languages and they are reasoned about as data flowing from one node to another.

Debates have gone on for decades between these two camps as to which is the more natural way to reason about programming. When you recognize that spreadsheets are a form of flow-based programs, it becomes clear that functional programming is the most common model. It is difficult to compare functional and imperative textual based programming languages because of a variety of syntactic differences that tend to swamp any kind of direct comparison. Because visual programming languages are essentially syntax-free, it is much more tractable to compare paradigms here.

## III. Experiment

While the evidence is clear about the advantage of block and dataflow languages over traditional text-based, C-style languages, we are unaware of any comparison between these

978-1-5386-2480-7/17/$31.00 ©2017 IEEE

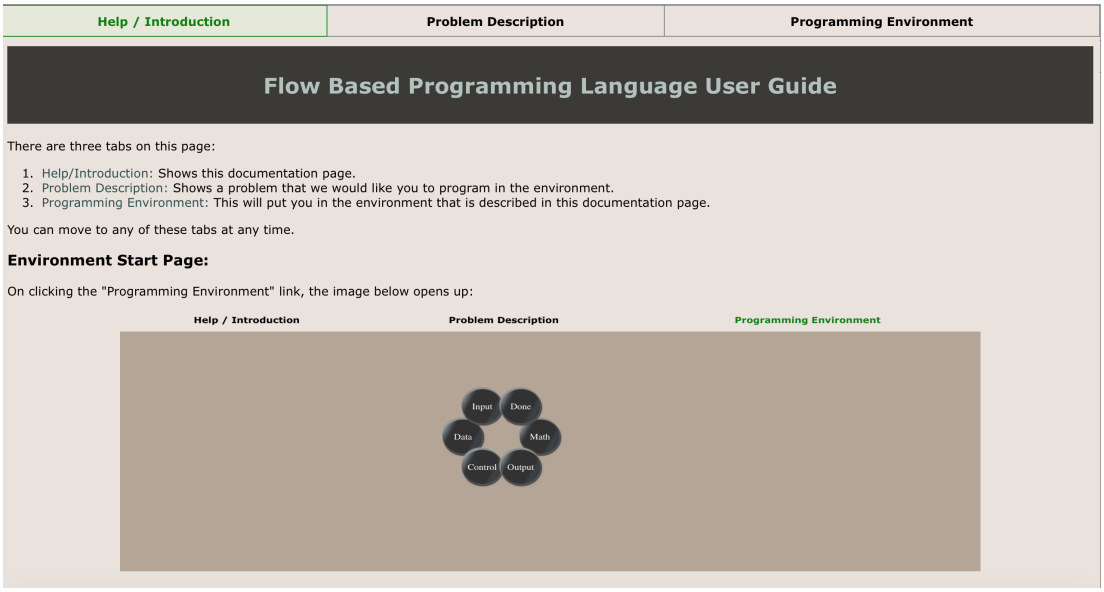


Fig. 1. Introduction for Flow

two styles of visual languages. Our working hypothesis is that dataflow programming will be easier for people with no previous programming experience.

To make as useful a comparison as possible, the block and flow environments are developed within the same code base, and use common code for menus, calculations, widgets, etc.

We will be running the experiment on Amazon's Mechanical Turk (AMT)[7] having truly naive programmers attempt to solve one of a series of simple programs, chosen at random, using one of the two paradigms, also chosen at random. When the AMT worker (participant) starts, they go to a page that makes the random choice of environment and problem. If they return to the page, they will get the same choices, by the use of cookies.

Next they are presented with a page with three tabs: Help/Introduction, a problem description, and the random environment. See figure 1 for the Introduction page for the flow version - the block version is similar.

The problem programs are:

1) find the odd numbers in a list, producing a new list,
2) square the numbers in a list, producing a new list,
3) calculate the mean of a list of numbers,
4) choose the larger of two given numbers and output the double of that number,
5) find the number of times a particular value is found in a list, and
6) given two lists, output the values from the first list that are greater than the corresponding values in the second list. (a block solution to this problem is in figure 2 and a flow solution to this problem is in figure 3)

While none of these are very difficult (constrained by the workers needing to complete the task in a reasonable time) they cover a range of difficulties. They are also chosen so they can all be solved using only a few principles in either of the environments.

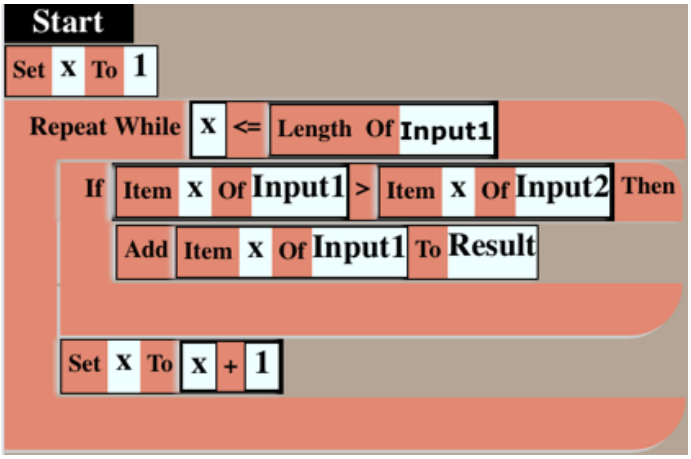


Fig. 2. Block-environment solution to one of the experiment problems

There are two independent variables:

1) flow versus block environment, and
2) the chosen problem out of the available six.

Each participant will initially experience a single combination from the 12 possible. The reason that we are using AMT is that we are targeting 600 participants, so each combination will have a sample of approximately 50 participants. We will compare these initial experiences statistically and depend on the number of participants for validity.

We will also offer each participant an opportunity to continue with a follow-on experience (the other environment, or a different problem in the same environment - chosen at random), and analyse these in a separate analysis.

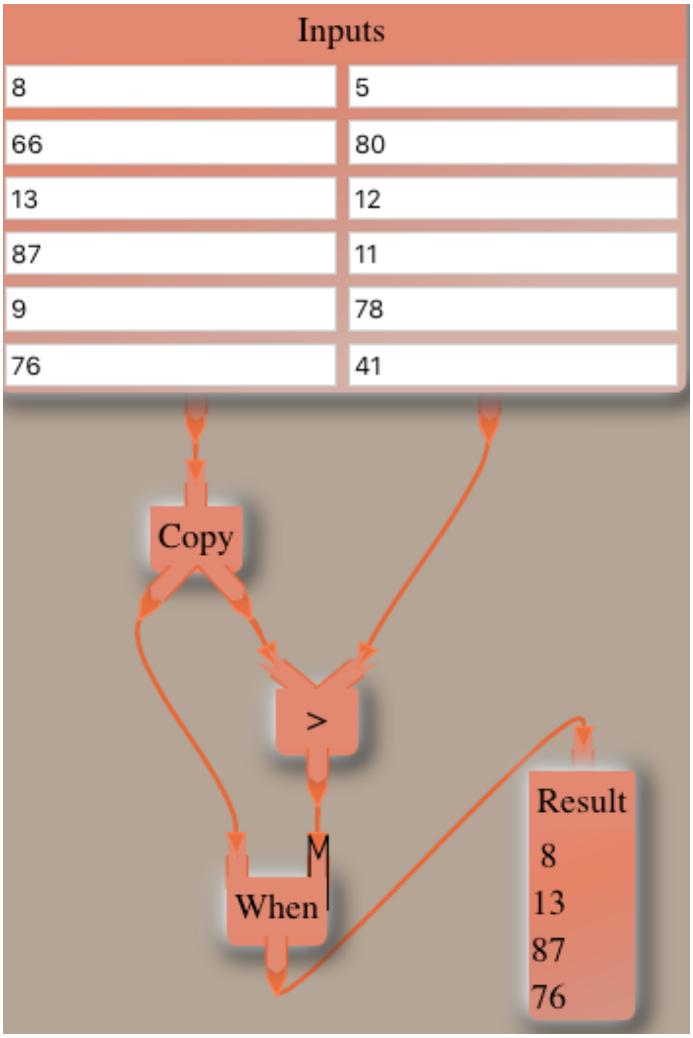


Fig. 3. Flow-environment solution to one of the experiment problems

There are a variety of dependent variables we will be examining, including:

- time to complete,
- correctness,
- time spent consulting help,
- what visual objects created.

In this way, we hope to be able to gain an understanding of what issues naive programmers face when attempting to program for the first time.

## IV. Related Work

There are many efforts underway to enable users to access the vast pool of data at our electronic fingertips. Search engines, such as Google[8], when presented with a search such as "correlation between streetlights and potholes" are capable of finding what *other people* have already discovered, but can't create such information themselves. Systems such as Wolfram's Alpha[9] and Apple's Siri[10] may aspire to answer such questions but - for now - simply answer, "no match found". IFTTT[11] provides very rudimentary programming relating to web events and actions - but only those events and actions that someone else has programmed.

The programming model most closely related to ours is that of dataflow programming[12], [13]. We have attempted to take the best ideas of systems like Prograph[14], combine them with ideas from Scratch[1], and bring them into a modern, web/touch-device interface.

Principles of design and structure are inspired by Papert[15] and Victor[16], [17]. While Papert is talking about teaching children math, the principles of embodiment and feedback are key to understanding for any novice programmer. Victor points out that traditionally we expect programmers to simulate program execution in their heads, which is an especially high expectation for our users.

There have been several experiments over the decades to compare graph-based and textual-based program representations[18], [19], [20], [21] that have had indeterminate results or were not actual flow-based languages. Two more direct studies[22], [23] showed flow-based languages to be better for program comprehension.

Similarly there have been several experiments that have compared block-based and textual-based program representations[24], [25], [26], [27], [28]. The results have been generally positive in favour of the block-based environments, but several of these studies have problems of comparing against difficult textual languages (e.g. Java) or having small numbers.

## V. Conclusions

Creating a language in which non-programmers will be happy programming creates many opportunities and challenges.

Non-programmers will not have the same investment in a particular syntax/semantics, which allows for language evolution in a way completely inconceivable for traditional languages and their programmers.

Conversely, non-programmers will not as easily recognize earlier code, so it is imperative that the data connections and language representation be as accessible as possible.

Of course, we can't know in advance if people will use the environment for programming in the large, but we are initially explicitly making decisions to preferentially support casual and small-scale approaches wherever a decision must be made - i.e. we are attempting to grow the language using agile principles.

We have not yet run this with the large target audience on AMT. We have only very preliminary results from a much smaller and less random sample, but they appear to suggest that there was little significant difference between the block-based and flow-baed environment for our participants.

Although we have been very careful in the design of the environments, languages, and experiment, there will doubtless be confounding factors - such as the choice of primitives (block/node types) provided in each environment.

## REFERENCES


[1] K. A. Peppler and Y. B. Kafai, "Creative coding: Programming for personal expression," 2005.

[2] A. A. diSessa. Boxer. [Online]. Available: http://www.soe.berkeley.edu/boxer/

[3] M. Puckette. Pure data. [Online]. Available: http://puredata.info/

[4] A. LLC. Marten. [Online]. Available: http://www.andescotia.com/products/marten/

[5] N. Instruments. Labview. [Online]. Available: https://en.wikipedia.org/wiki/LabVIEW

[6] B. McNeel and S. Davidson. Grasshopper. [Online]. Available: http://www.grasshopper3d.com/

[7] "Amazon mechanical turk," https://www.mturk.com/.

[8] "Google search engine," http://google.com/.

[9] "Wolfram alpha," http://www.wolframalpha.com/.

[10] "Apple Siri," http://www.apple.com/ios/siri/.

[11] "IFTTT: If This Then That," https://ifttt.com/wtf.

[12] W. R. Sutherland, "The on-line graphical specification of computer procedures," Ph.D. dissertation, Massachusetts Institute of Technology. Dept. of Electrical Engineering, 1966.

[13] M. Boshernitsan and M. Downes, "Visual programming languages: A survey," http://techreports.lib.berkeley.edu/accessPages/CSD-04-1368.html, Computer Science Division, EECS University of California, Berkeley, Tech. Rep. UCB/CSD-04-1368, Dec. 2004.

[14] S. Matwin and T. Pietrzykowski, "Prograph: a preliminary report," *Comput. Lang.*, vol. 10, no. 2, pp. 91–126, Aug. 1985. [Online]. Available: http://dx.doi.org/10.1016/0096-0551(85)90002-5

[15] S. Papert, *Mindstorms : children, computers, and powerful ideas,* $2^{nd}$ *Edition*. New York, USA: BasicBooks, 1993.

[16] B. Victor, "Magic ink: Information software and the graphical interface," http://worrydream.com/MagicInk/, 2006.

[17] ——, "Inventing on principle," in *Canadian University Software Engineering Conference*, Montreal, Canada, Jan. 2012.

[18] D. Stein and S. Hanenberg, "Comparison of a visual and a textual notation to express data constraints in aspect-oriented join point selections: A controlled experiment," in *Proceedings of the 2011 IEEE 19th International Conference on Program Comprehension*, ser. ICPC '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 141–150. [Online]. Available: http://dx.doi.org/10.1109/ICPC.2011.9

[19] Z. Sharafi, A. Marchetto, A. Susi, G. Antoniol, and Y.-G. Guéhéneuc, "An empirical study on the efficiency of graphical vs. textual representations in requirements comprehension." in *ICPC*. IEEE Computer Society, 2013, pp. 33–42.

[20] T. R. G. Green and M. Petre, "Usability analysis of visual programming environments: A ?cognitive dimensions? framework," *Journal of Visual Languages & Computing*, vol. 7, pp. 131–174, 1996.

[21] H. R. Ramsey, M. E. Atwood, and J. R. V. Doren, "Flowcharts versus program design languages: An experimental comparison," *Communications of the ACM*, vol. 26, no. 6, pp. 445–449, 1983. [Online]. Available: http://doi.acm.org/10.1145/358141.358149

[22] N. Cunniff and R. P. Taylor, "Empirical studies of programmers: Second workshop," G. M. Olson, S. Sheppard, and E. Soloway, Eds. Norwood, NJ, USA: Ablex Publishing Corp., 1987, ch. Graphical vs. Textual Representation: An Empirical Study of Novices' Program Comprehension, pp. 114–131. [Online]. Available: http://dl.acm.org/citation.cfm?id=54968.54976

[23] K. N. Whitley, L. R. Novick, and D. Fisher, "Evidence in favor of visual representation for the dataflow paradigm: An experiment testing labview's comprehensibility," *Int. J. Hum.-Comput. Stud.*, vol. 64, no. 4, pp. 281–303, Apr. 2006. [Online]. Available: http://dx.doi.org/10.1016/j.ijhcs.2005.06.005

[24] D. Weintrop, "Comparing text-based, blocks-based, and hybrid blocks/text programming tools," in *Proceedings of the eleventh annual International Conference on International Computing Education Research*, ser. ICER '15. New York, NY, USA: ACM Press, 2015, pp. 283–284.

[25] D. Weintrop and U. Wilensky, "To block or not to block, that is the question," in *Proceedings of the 14th International Conference on Interaction Design and Children*, ser. IDC '15. New York, NY, USA: ACM Press, 2015, pp. 199–208.

[26] ——, "Using commutative assessments to compare conceptual understanding in blocks-based and text-based programs," in *Proceedings of the eleventh annual International Conference on International Computing Education Research*, ser. ICER '15. New York, NY, USA: ACM Press, 2015, pp. 101–110.

[27] T. Booth and S. Stumpf, "End-user experiences of visual and textual programming environments for arduino," *End-User Development Lecture Notes in Computer Science*, pp. 25–39, 2013.

[28] F. McKay and M. Kölling, "Predictive modelling for hci problems in novice program editors," in *Proceeding BCS-HCI '13 Proceedings of the 27th International BCS Human Computer Interaction Conference*, 2013.