The Spreadsheet Paradigm: A Basis for Powerful and Accessible Programming

Gary Miller
University of Technology Sydney, Australia
gary.miller@uts.edu.au

Abstract

This paper takes a cognition-centric approach for programming languages. It promotes the spreadsheet paradigm, with two concrete goals. First, it calls for the design and implementation of several language features to enhance the expressiveness of spreadsheet programming. Second, it describes a plan for rigorous empirical studies to retain the learnability of spreadsheet programming.

Categories and Subject Descriptors D.3.m [*Programming Languages*]: Language Design; H.1.2 [*Information Systems*]: Human factors

General Terms Languages, Human Factors, Experimentation

Keywords Spreadsheet Paradigm, Programming Language Design, End-User Programming, Empirical Testing

1. The Purpose

This project will develop a set of related programming languages, inspired by the spreadsheet paradigm, and test their features for cognitive accessibility by humans.

The need for empirical testing in programming language design is acknowledged [5, 17, 18, 21, 22] yet seldom undertaken [8, 23]. We believe that the basic notions of the spreadsheet paradigm can be built upon to create a powerful computational representation, and that empirical testing will enable us to maintain human cognitive accessibility to the added features.

The spreadsheet is arguably one of the most popular forms of programming. We contend that this is because it is a form of programming that is closer to innate human cognitive abilities than most others. At the very least, the cognitive abilities that emerge from most current general education systems map closely to the abstractions present in the spreadsheet paradigm.

Yet in its current form spreadsheet programming is problematic for numerous reasons. Spreadsheets are error prone [19]. Worse, we contend that they are cognitively limiting. In order to given an example of a cognitively limiting aspect of spreadsheets, we define two levels of programming;

- End-user development (EUD) is programming or modeling for self-use.
- Application programming, where the artefacts are used by others.

Users are generally constrained to one level, that of end-user development. There are likely many interwoven deficiencies of spreadsheets that contribute to this; expressive power, robustness, information hiding, and ability to automate repetitive processes. This containment to end-user programming is subconscious and as such current spreadsheets do not substantially enable Computational Thinking [27], which is characterised by abstraction and automation.

This work, in large part, is based on a previous attempt to create a richer modelling environment [16]. The motivation of which was to capture design patterns used by expert spreadsheet users.

The motivation of this work is our beliefs that the Relative Linguistic hypothesis [26] is relevant to programming. If people acquire better language they are cognitively richer. We acknowledge that the majority of linguists do not give much credence to the Whorfian hypothesis, but hold our belief. Noting that all the linguistic evidence we are aware of if based in natural language (type 0 grammars) and recent supporting work on relative verse absolute spatial terms [4].

2. The Goals

The primary goal of this research is to evolve a programming language and environment to mitigate the tension between learnability and power. Warth et al. [25] points out that they know of no end-user programming system that provides both a gentle learning curve and high ceiling.

Current Spreadsheets are learnable but not powerful. There are a number of discontinuities (eg. the difference

between reference syntax and lookup functions), missing features (eg. user defineable functions) and missteps (eg. datatables) in the current paradigm. This may well be good enough for the majority of spreadsheet users who require no more than table based text layout, but leaves modellers in an unacceptable situation. The goal is to enable needy and curious modelers to access more powerful computational representations.

A secondary goal is to learn the techniques of empirical testing and help harden some of the soft science approaches we feel could assist the programming language community [17].

3. The Approach

We will investigate semantic language proposals and empirical testing options. We will also investigate syntactic and environment issues as they play an equal or even larger part in the accessibility of these features.

We are taking a "... constructive approach to the problem of program correctness ..." Dijkstra [6]. As such the main focus of the empirical testing is on program comprehension and modification. Program composition and debugging are beyond the scope of this work.

At their base all our spreadsheet based experimental languages contain the common elements of a grid of cells contained in sheets. Each cell is capable of containing an expression which can reference other cells. This is similar to the definition in [11] and [1].

New features are added to the experimental languages to enable more powerful spreadsheet modelling. The features are inspired by common patterns in programming and expert modelling. The majority of the differences between the languages are syntactic and semantic variations in the new language features.

Whilst we can show that these features are more powerful it is not possible to determine, from first principles, the form and interpretation that best map to innate human cognitive abilities.

3.1 New Language Features Enabling Grouping

As mentioned the new features being implemented are based in the design patterns of expert modellers. The most obvious pattern we observed is that these users think in terms of groups of cells. Many of the subsequent new language features come from our design of cell grouping.

Wang and Ambler [24] advocates for the replacement of cell-level manipulation with region-level manipulation, pointing out that this is "probably closer to the mental images ... of the users". We are not advocating for the replacement of cell level editing, but for the addition of grouping. Our notion of cell groups is a separate list of reference, formula pairs *(reference,formula)** . This is similar to, but more flexible than the way Lotus-Improv [15] apply formulae to cells.

Enhanced Referencing: To realise the grouping feature, richer referencing semantics are required. This is for two reasons.

- Firstly for a richer way of referring to cells to be included in a group.
- Secondly, more flexible referencing, than the current options of absolute or relative forms, enables more opportunities for cells to be grouped together and share a single formula.

We propose to start experimenting with syntaxes that allow for aspects of both the A1 and R1C1 forms to coexist. In addition features that enable querying cell's contents currently done in lookup formulae and formulae that have criteria arguments. Much of our experimentation is expected to centre on the intuitiveness of referencing syntax and semantics. As this is a key enabler for may of the other features.

Enhanced Axes: Axes is the term used to refer to rows and columns, and nodes or a node is the term for an individual row or column. In order to significantly enhance referencing we propose to allow for more information to be captured in the axes and nodes. That is; 1.) tree structured axes, and 2.) Nodes capable of contain multiple names. The structure and names are user defined. This enables referencing to be enhanced with connotations applicable to named hierarchies. For example; children, parent, name, path, offset from a name ignoring hierarchy.

Atticus Operator and Returned Cells: A common and necessary design pattern in spreadsheet modelling is the use of repeated blocks for intermediate calculations. These blocks violate the don't repeat yourself (DRY) principle. A feature, dubbed the Atticus Operator was added to enable the removal of these repeated blocks. The operator when used in an expression changes the context cell used when evaluating a reference.

It takes advantage of the geometric nature of the cells and appear to be novel. We have not been able to find a similar syntactic operator in other programming languages. We took the liberty of naming after Atticus Finch's in [14].

This feature is a good candidate for testing elements of the environment (aka IDE) (e.g. prompting re-factoring) to aid learnability.

3.2 Implementation

At this point in the research the only firm implementation requirements are that the testing environment is generally available and the languages are easy to mutate.

The first requirement suggests a web technology solution. The second has pointed the language implementation to be undertaken in composable grammars. At this point at least three separate parser grammars are needed. A structure grammar to define the sheets and axes, an expression grammar and a reference grammar. The interpreter is being implemented as a number of tree re-writes.

3.3 Science of User-centred Programming Language Design

The area of Empirical Studies contains the most unknown unknowns for us. We propose to first engage in a number of reproduction studies and create some less technically involved experiments (e.g. surveys). There are a number of texts (e.g. [28]) that will aid in this area.

A later proposed form of the empirical testing is to provide subjects with two implemented models solving the same problem. The first in a common spreadsheet application and the second in one of experimental languages. The whole cohort will receive the same spreadsheet and one of two test languages. Programming tasks will be asked of the participants and as much information about their completion of the task recorded.

This will then be used to measure the fitness of the new features against each other. The tasks with differ depending on what aspect of programming is being tested eg comprehension, debugging, modification or learning. As a starting point the solution to the task maybe provided for the spreadsheet. This test setup does not enable the testing of new program composition, this is by design due to the nature of having a new unfamiliar programming language as part of the experiment.

4. Related Work

There is a long history of academic investigation into the spreadsheet paradigm and visual programming. Of special note is Ambler's work generalising spreadsheets, and hypothesising on the relationship between forms of programming and innate human abilities [2, 3, 24].

Jones et al. [10] work includes both the ideas of evolving the spreadsheet paradigm and human focus on the proposed ideas. Erwig has done work on a number of areas aimed at spreadsheet program correctness and error detection, an example of which is [7]. Sestoft et al. [20] looks at implementing and extending a spreadsheet engine for research purposes.

Some interesting work on empirical studies of two dimensional programming languages and its comparison to Fortran were conducted [12, 13]. Hanenberg and Stefik's work [9] and general talks on the topic of the need for empirical testing of programming languages [23] have been inspiring.

Acknowledgments

This work is supported by an industrial scholarship provided by System Builder Development Pty Limited.

References

[1] R. Abraham, M. Burnett, and M. Erwig. Spreadsheet programming, 2009.

[2] A. Ambler. Generalizing the sheet language paradigm. In *Visual Languages and Applications*. 1990.

[3] A. Ambler and J. Leopold. Interactive communication over inferred indexed iteration. pages 1–8, 1999.

[4] L. Boroditsky. How language shapes thought. *Scientific American*, 304(2):62–65, 2011. .

[5] B. Curtis, E. M. Soloway, R. E. Brooks, J. B. Black, K. Ehrlich, and H. R. Ramsey. Software psychology: the need for an interdisciplinary program. *Proceedings of the IEEE*, 74(8):1092–1106, 1986. .

[6] E. Dijkstra. The tide, not the waves. *Beyond calculation: The next fifty years of computing*, 1997.

[7] M. Erwig, R. Abraham, S. Kollmansberger, and I. Cooperstein. Gencel: a program generator for correct spreadsheets. *Journal of Functional Programming*, 16(03):293, 2006. .

[8] S. Hanenberg. Faith, hope, and love: an essay on software science's neglect of human factors. In *ACM Sigplan Notices*, volume 45, pages 933–946, 2010.

[9] S. Hanenberg, S. Kleinschmager, R. Robbes, É. Tanter, and A. Stefik. *An empirical study on the impact of static typing on software maintainability*. 2013. .

[10] S. P. Jones, A. Blackwell, and M. Burnett. A user-centred approach to functions in excel. In *ACM SIGPLAN Notices*, volume 38, pages 165–176, 2003.

[11] A. Kay. Computer software. *Scientific American*, 251(3):52–59, 1984.

[12] M. Klerer. Software modeling studies experimental study of a two dimensional language versus fortran for first-course programmers. 1981.

[13] M. Klerer and J. May. Two-dimensional programming*. In *Proceedings of the November 30–December 1, 1965, fall joint computer conference, part I*, pages 63–75, 1965.

[14] H. Lee. To kill a mockingbird. *New York City: JB Lippincott Company*, 1960.

[15] Lotus. Lotus improv.

[16] D. Miller, G. Miller, and L. M. Parrondo. Sumwise : A smarter spreadsheet. In *EuSpRiG*, 2010.

[17] A. Newell and S. Card. The prospects for psychological science in human-computer interaction. *Human-Computer Interaction*, 1(3): 209–242, 1985. .

[18] J. Pane and B. Myers. The influence of the psychology of programming on a language design: Project status report. *Human-Computer Interaction*, (April), 2000.

[19] R. Panko. What we dont know about spreadsheet errors today. In *EuSpRIG*, 2015.

[20] P. Sestoft, J. D. Rask, and S. Eikeland. End-user development via sheet-defined functions. *Software Engineering Methods in Spreadsheets*, page 8, 2014.

[21] B. a. Sheil. The psychological study of programming. *ACM Computing Surveys*, 13(1):101–120, 1981. .

[22] B. Shneiderman. *Software psychology: Human factors in computer and information systems (Winthrop computer systems series)*. Winthrop Publishers, 1980.

[23] A. Stefik and S. Hanenberg. The programming language wars: Questions and responsibilities for the programming language community. In *Proceedings of the 2014 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software*, pages 283–299, 2014.

[24] G. Wang and A. Ambler. Solving display-based problems. In *Visual Languages, 1996. Proceedings., IEEE Symposium on*, pages 122–129, 1996.

[25] A. Warth, Y. Ohshima, T. Yamamiya, and S. Wallace. Toward a more scalable end-user scripting language. *Proceedings - 6th International Conference on Creating, Connecting and Collaborating through Computing, C5 2008*, (0639876):172–178, 2008. .

[26] B. L. Whorf. Language, thought, and reality: selected writings. 1956.

[27] J. M. Wing. Computational thinking. *Communications of the ACM*, 49(3):33–35, 2006.

[28] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in software engineering: An introduction*. 2000. .