Programmer Experience Design and the Spreadsheet Paradigm

Gary Miller

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

Keywords: POP-VI.F. exploratory, POP-V.A. theories of design, POP-III.B. spreadsheets

Abstract. This doctoral consortium report proposes using the continuing evolution of the spreadsheet paradigm to frame a discussion about research into the usability of programming languages. In particular it argues that research into programming language design needs to be hardened, and should adopt a perspective similar to that described in *The Prospects for Psychological Science in Human-Computer Interaction* (Newell & Card, 1985). In summary there are three components, the need to work on good applied problems, using empirical studies to provide the basis for theories to aid programming language designers.

1 Introduction

In this work the first two components, the good applied problem and empirical studies are reasonably well defined. The good applied problem is making spreadsheet modelling more expressive. For example by adding a cell grouping feature supported by a more expressive cell referencing style, a concrete example can be seen in (G. Miller & Hermans, 2016). Empirical studies into different syntactic and semantic forms of this cell referencing style are proposed. The third component, that of theories and principles for programming language design are more speculative.

The rest of this paper is structured as following. Section 2 discusses the background to more expressive forms of spreadsheet modelling and the intersection of programming language research and spreadsheets. Section 3 discusses the proposed studies into cell referencing and the general state of empirical studies into programming language usability. Section 4 discusses the state of theories in the area of programming languages design and proposes and discusses some example theories. Concluding with a brief discussion in section 5.

2 Background

The spreadsheet paradigm is arguably the most used form of programming. A substantial amount of academic research has looked at spreadsheets as programming (Ambler, 1987, 1990; Ambler & Broman, 1998) (Cunha et al., 2014). Of particular interest much of this research is concerned with both the technical and human behaviours aspects of programming languages design (Saariluoma & Sajaniemi, 1989; Burnett et al., 2001; Jones, Blackwell, & Burnett, 2003; Kankuzi & Sajaniemi, 2015). However spreadsheets are generally considered to be something different to programming, and neither spreadsheet vendors or the programming language designers seam to have paid much attention. The spreadsheet paradigm still produces error prone models, is inexpressive and is lacking in common programming abstractions.

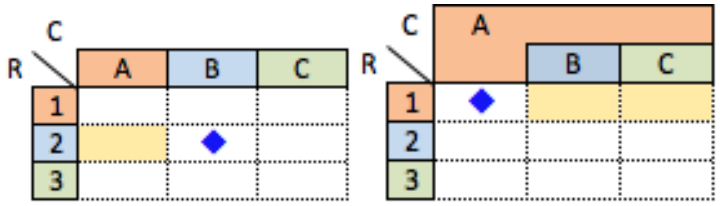
This research is based on the work historically done at a start-up Sumwise that explicitly tried to evolve the spreadsheet paradigm by capturing the modelling patterns of expert spreadsheet modellers as application features. Sumwise identified that a common behaviour shared by many expert spreadsheet modellers is thinking and modelling in-terms of groups of cells. Another common behaviour is the extent to which they provide labels around these cell groups. Sumwise used these two observations to provide declarative cell grouping supported by an enhanced referencing style (D. Miller, Miller, & Parrondo, 2010).

3 Empirical Studies

There is a long history in the call for empirical research into the usability of programming languages (Klerer, 1972; Weinberg, 1971; Shneiderman, 1980), which has mainly been ignored (Stefik et al., 2014). However there appears to be a renewed interest in this area (Myers et al., 2016; Anslow, LaToza, & Sunshine, n.d.).

We believe that usability questions around more expressive modelling can only be answered with human behavioural studies, and intent to undertake these are part of this research. Specifically these experiments will look at the usability of enhanced referencing.

For example (see figure 1), based on the hypothesis that people confuse the rows and columns when using the R1C1 syntax, we would like to test $RC[-1]$ against a made up syntax $[Left][ThisRow]$ or when using structure $RC[Children]$ verses $[ChildrenRight][ThisRow]$.



(a) Offset (b) Structure

Fig. 1: Example spreadsheets for proposed empirical studies

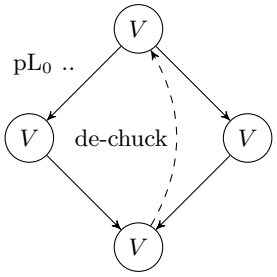


Fig. 2: Combined model: Concept Graph, ACT-R with de-chunck costs

4 Theories Pertinent to Programming Language Design

In this section the desired and current states of theories into programming language design and how this relates to this work are discussed. We feel that the desired state is eloquently put in (Newell & Card, 1985)

> *"... is the need for an engineering-style theory of how the user interacts with the computer. This implies three important characteristics: task analysis, calculation, and approximation."*

As noted in (Hansen, Lumsdaine, & Goldstone, 2012) this in not the current state of psychology of programming.

An obvious analytical theory that applies is Power Law of Practise (i.e. time and error rate decrease exponentially with practise) and associated chunking theory of learning (Newell & Rosenbloom, 1981). The power law is often used in empirical design. For example with enough of an effect size or sufficient data two syntactically different language features can be compare simply by the slope and asymptote of their power laws. However this theory is too low level to be applied directly to language design, for example it does not help create possible referencing styles to be tested. The main use of this theory might be to validate higher level theories (e.g. this is done by ACT-R based intelligent tutoring systems (Anderson, Conrad, & Corbett, 1989)).

Higher level theories like Cognitive Dimensions of Notation (A. Blackwell & Green, 2003) help in designing the materials for empirical tests, but do not have analytical or engineering qualities. The Attention investment (A. F. Blackwell & Green, 1999) model attempts to move in this more analytical direction, however it is unclear how this might be used to calculate the usability of languages or language features.

The ACT-R model as applied to intelligent tutoring systems suggests an analytically model that could potentially be used to compare languages features, but not programming languages as a whole. For example the production co-coefficients (Corbett, Anderson, & Derson, 1994) (pL_0 - Initial learning, pT - Acquisition, pG - Guess, pS - Slip) could be used to directly compare language features. However one of the weaknesses of the ACT-R system is that it does not model incorrect choices (Anderson, 1993, chapter 13 Reflections on the Theory). This work speculates that similar programming languages could be compared using a modified ACT-R model. That is, by modelling programming languages as a directed acyclic graph of concepts (ACT-R productions rules), attaching values to the nodes, and adding de-chucking costs when learning backtracks in the graph (see figure 2). The cost of extracting value from a programming language is then a function of the production rules and the order the language is learnt.

This modified ACT-R model for programming language design has similarities to theories in the area of programming language education, specifically Anchor Graphs (also called concept graphs) (Mead et al., 2006) and constructivism in computer science education (Ben-Ari, 2001). These two areas are closely related, and programming language design can learn from and contribute to the large discipline of programming education.

There are many less analytical theories and principles that can be richly applied to programming language design. A few pertinent examples are mentioned. Linguistic abstraction (Visser, 2013) is the capturing of patterns as language features, the cell grouping feature of Sumwise is an example of this. Orthogonality of language features is an under used principle in programming language design, despite the long standing evidence to its benefit (Bowden, Douglas, & Stanford, 1989), though there are some examples of it being explicitly stated in language design (Pike, 2010). Directness and liveliness are also old ideas (Shneiderman, 1993) that have recently resurfaced (Victor, 2012; McDirmid, 2016) with much promise. Many ideas from cognitive psychology can be applied to programming language design, for example the prevalence and intuitiveness of natural language grammars (SVO 44% of all natural languages vs VSO 19%, OVS 0% and OSV 0%) (Anderson, 2000, chapter 11 Language Structure).

5 Discussion

There is no discipline dedicated to programming language design from a human perspective. It therefore looks to existing areas like HCI, psychology of programming, computer science education and user experience design. This

paper would like to start a discussion about whether such a discipline, call it Programmer Experience Design (PX) should exist, and proposes that it could be modelled on the early HCI as described in (Newell & Card, 1985).

The continuing evolution of the spreadsheet paradigm would make a good topic area for PX. There has historically been research in spreadsheets that overlap between programming language evolution and human factors, and spreadsheets are a rare example of intuitive and usable programming.

References

Ambler, A. (1987). Forms: Expanding the visualness of sheet languages. *1987 Workshop on Visual Languages*, 105–117.

Ambler, A. (1990). Generalizing the sheet language paradigm. In *Visual languages and applications.*

Ambler, A., & Broman, A. (1998). Formulate Solution to the Visual Programming Challenge. *Journal of Visual Languages & Computing*, 171–209.

Anderson, J. R. (1993). Rules of the Mind.

Anderson, J. R. (2000). *Cognitive psychology and its implications .* (fifth edit ed.). WH Freeman/Times Books/Henry Holt & Co.

Anderson, J. R., Conrad, F. G., & Corbett, A. T. (1989). Skill acquisition and the LISP tutor. *Cognitive Science*, *13*(4), 467–505.

Anslow, C., LaToza, T., & Sunshine, J. (n.d.). *(2009-2016). PLATEAU: Workshop on Evaluation and Usability of Programming Languages and Tools.* Retrieved from `https://sites.google.com/site/workshopplateau/`

Ben-Ari, M. (2001, 3). Constructivism in computer science education. *Journal of Computers in Mathematics and Science Teaching*, *20*(1), 45–73. Retrieved from `http://doi.acm.org/10.1145/274790.274308` doi: 10.1145/274790.274308

Blackwell, A., & Green, T. (2003). Notational systems–the cognitive dimensions of notations framework. *HCI Models, Theories, and Frameworks: Toward an Interdisciplinary Science. Morgan Kaufmann*.

Blackwell, A. F., & Green, T. (1999). Investment of attention as an analytic approach to cognitive dimensions. In *Collected papers of the 11th annual workshop of the psychology of programming interest group (ppig-11)* (pp. 24–35).

Bowden, E., Douglas, S., & Stanford, C. (1989). Testing the Principle of Orthogonality in Language Design. *Human-Computer Interaction*, *4*(2), 95–120. doi: 10.1207/s15327051hci0402{_}1

Burnett, M., Atwood, J., Djang, R. W., Gottfried, H., Reichwein, J., & Yang, S. (2001). Forms/3: A First-Order Visual Language to Explore the Boundaries of the Spreadsheet Paradigm. *Functional Programming*, *11*(2), 155–206.

Corbett, A. T., Anderson, J. R., & Derson, J. (1994). Knowledge tracing: Modeling the acquisition of procedural knowledge. *User modeling and user-adapted interaction*, *4*(4), 253–278.

Cunha, J., Costa, M., Fernandes, J., Soares, P., Mendes, J., Pereira, R., & Saraiva, J. A. (2014). MDSheet-Model-Driven Spreadsheets.

Hansen, M., Lumsdaine, A., & Goldstone, R. L. (2012). Cognitive architectures: A way forward for the psychology of programming. In *Proceedings of the acm international symposium on new ideas, new paradigms, and reflections on programming and software* (pp. 27–38). New York, NY, USA: ACM. Retrieved from `http://doi.acm.org/10.1145/2384592.2384596` doi: 10.1145/2384592.2384596

Jones, S. P., Blackwell, A., & Burnett, M. (2003). A user-centred approach to functions in Excel. In *Acm sigplan notices* (Vol. 38, pp. 165–176). Retrieved from `http://dl.acm.org/citation.cfm?id=944721`

Kankuzi, B., & Sajaniemi, J. (2015). A mental model perspective for tool development and paradigm shift in spreadsheets. *International Journal of Human-Computer Studies*, *86*, 149–163.

Klerer, M. (1972, 10). The Economics, Politics, and Sociology of Two-dimensional Systems. *SIGPLAN Not.*, *7*(10), 11–22. Retrieved from `http://doi.acm.org/10.1145/942576.807010` doi: 10.1145/942576.807010

McDirmid, S. (2016). The Promise of Live Programming.

Mead, J., Gray, S., Hamer, J., James, R., Sorva, J., Clair, C. S., & Thomas, L. (2006). A Cognitive Approach to Identifying Measurable Milestones for Programming Skill Acquisition. In *Working group reports on iticse on innovation and technology in computer science education* (pp. 182–194). New York, NY, USA: ACM. Retrieved from `http://doi.acm.org/10.1145/1189215.1189185` doi: 10.1145/1189215.1189185

Miller, D., Miller, G., & Parrondo, L. M. (2010). Sumwise : A Smarter Spreadsheet. In *Eusprig*. Retrieved from `http://www.sumwise.com/wordpress/wp-content/uploads/Sumwise-A-Smarter-Spreadsheet.pdf`

Miller, G., & Hermans, F. (2016). Gradual Structuring in the Spreadsheet Paradigm. In *Visual languages and human-centric computing (vl/hcc), 2016 ieee symposium on.*

Myers, B. A., Stefik, A., Hanenberg, S., Kaijanaho, A.-J., Burnett, M., Turbak, F., ... Kaijanaho, A.-J. (2016). Usability of Programming Languages: Special Interest Group (SIG) Meeting at CHI 2016. In *Proceedings of the 2016 chi conference extended abstracts on human factors in computing systems* (pp. 1104–1107).

Newell, A., & Card, S. (1985). The Prospects for Psychological Science in Human-Computer Interaction. *Human-Computer Interaction*, *1*(3), 209–242.

Newell, A., & Rosenbloom, P. S. (1981). Mechanisms of skill acquisition and the law of practice. *Cognitive skills and their acquisition*, *1*, 1–55.

Pike, R. (2010). The Expressiveness of Go. In *Jaoo.*

Saariluoma, P., & Sajaniemi, J. (1989). Visual information chunking in spreadsheet calculation. *International Journal of Man-Machine Studies*, *30*(5), 475–488.

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

Shneiderman, B. (1993). 1.1 direct manipulation: a step beyond programming languages. *Sparks of innovation in human-computer interaction*, *17*, 1993.

Stefik, A., Hanenberg, S., McKenney, M., Andrews, A., Yellanki, S. K., & Siebert, S. (2014). What is the Foundation of Evidence of Human Factors Decisions in Language Design? An Empirical Study on Programming Language Workshops. In *Proceedings of the 22nd international conference on program comprehension* (pp. 223–231). New York, NY, USA: ACM. Retrieved from `http://doi.acm.org/10.1145/2597008.2597154` doi: 10.1145/2597008.2597154

Victor, B. (2012). Learnable programming: Designing a programming system for understanding programs. *URL: http://worrydream. com/LearnableProgramming*.

Visser, E. (2013). Understanding Software through Linguistic Abstraction. *Science of Computer Programming*.

Weinberg, G. M. (1971). The psychology of computer programming. , *932633420*.