A brief history of the human centric study of programming languages

Luke Church
Computer Laboratory
Cambridge University
luke@church.name

Alan F. Blackwell
Computer Laboratory
Cambridge University
Alan.Blackwell@cl.cam.ac.uk

Abstract

The study of programming languages focussing on the needs of the programmer has been a subject of intellectual enquiry for at least the past 50 years. We draw one through this history that is likely to be of particular interest to the Psychology of Programming audience highlighting recurring tensions of internal validity, generalisability and practical utility. We suggest this longer term perspective is useful for informing contemporary debates.

1. Introduction

The question of how to design languages for programming computers has been a matter of interest for a number of different disciplines in the past. Engineers working in compiler construction have considered the impact of the design of the languages on the technology needed to run programs (e.g. Waite and Goos 1984). Mathematicians have studied the ways in which the structure of programs affects the statements we can make about their behaviour (e.g. Pierce 2002). Psychologists and interaction design researchers have studied the design from a human perspective. In their landmark paper (Newell and Card 1985) observe that:

"Now programming languages are obviously symmetrical, the computer on one side, the programmer on the other. In an appropriate science of computer languages, one would expect that half the effort would be on the computer side, understanding how to translate the languages into executable form, and half on the human side, understanding how to design languages that are easy or productive to use."

This paper offers an overview and reflections on the human focussed, scientifically inclined, study of programmers and their behaviour. It is not a historical overview of the literature in psychology of programming - readers interested in that overview can refer to (A. F. Blackwell, Petre, and Church 2019). Alternatively, a more light-hearted historical introduction to the field can be found in (A. F. Blackwell 2017).

2. The origins of programming

During the early history of electronic computing, the only way of interacting with a computer was to write, or control the execution of, a program. For more than a century, the study of the interaction with a computer was effectively the study of programming, from Ada Lovelace's famous 1843 notes on the programming of Babbage's Analytical Engine (Menabrea and of Lovelace 1843) to the 1947 report by Goldstine and Von Neumann on Planning and coding of problems for an electronic computing instrument (Goldstine and von Neumann, n.d.).

Although nothing corresponding to the modern user interface had been imagined, it is notable that these pioneers already focused on problems of notation, and also that they considered the extent to which it would be feasible to specify different classes of problem using these notations. This means that early research on interaction with programs can still be informative when discussing the properties of modern programming environments (see e.g. (Arawjo 2020)). Furthermore, due to the importance of programming in early computing, the study of the usability of programming was formative in the study of the usability of computing more generally. In the first part of this review we will describe the evolution of the perspectives and techniques that researchers have used to try and improve the usability of programming, as well as the venues in which this work was disseminated.

From the perspective of subsequent consolidation of cognitive science as an interdisciplinary field, we can see that early contributions often involved implicit cognitive claims about what people find easy or hard when programming, followed by the implications for design of those claims. Aspects of human performance and experience have been part of the argument from the beginning.

3. Early cognitive claims (1968-1973)

An early example of a cognitive claim made by a computer scientist came from Edsger Dijkstra. In a now famous letter to the Communications of the ACM, he stated his position that 'Go To Statement Considered Harmful' (Edsger W. Dijkstra 1968) . This work makes essentially cognitive arguments such as:

'*My second remark is that our intellectual powers are rather geared to master static relations and that our powers to visualize processes evolving in time are relatively poorly developed.*'

Dijkstra then goes on to consider the implications for design of these claims such as:

'For that reason we should do (as wise programmers aware of our limitations) our utmost to shorten the conceptual gap between the static program and the dynamic process, to make the correspondence between the program (spread out in text space) and the process (spread out in time) as trivial as possible.'

The broader position that is outlined in the 'goto considered harmful' letter is that the structure of programs can be described by a number of 'independent co-ordinates' inherent to sequential processes by which progress in execution can be measured. Dijkstra argues that the 'unbridled use of the go to statement has an immediate consequence that it becomes terribly hard to find a meaningful set of coordinates in which to describe the process progress.'

The argumentation is presented as an expert experience report "For a number of years I have been familiar with the observation that the quality of programmers is a decreasing function of the density of go to statements in the programs they produce.", it does not present evidence to support this claim, or a definition of what 'quality of programmers' might be, or even a description of what the programmers being considered were doing. However on the basis of this experience report and several cognitive hypotheses, Dijkstra felt confident in recommending broad design changes to the majority of programming languages:

‘*I became convinced that the go to statement should be abolished from all "higher level" programming languages (i.e. everything except, perhaps, plain machine Code).*’

In 1971, (Evershed and Rippon 1971) follow a similar form of argumentation, considering their experience of the usability characteristics of various aspects of ALGOL and FORTRAN. On the basis of these characteristics they go on to make recommendations for a variety of design changes. They include an economic argument, which can be seen as a precursor of end-user programming: ‘with the esoteric content of programming eliminated, a much broader section of the population will become potential computer users’.

The scope of Evershed and Rippon’s work is broader than Dijkstra’s earlier work; it independently discusses several language features, however it doesn’t propose an overarching psychological hypothesis like Dijkstra’s ‘independent coordinates’.

Evershed and Rippon, as well as Dijkstra, argue from their experience rather than from documented empirical evidence. When (Sime, Green, and Guest 1973) published the results of a controlled experiment, it represented both a new approach to studying programming languages as well one of the first applications of what would become known as Human Computer Interaction (HCI) methods (Katz, Petre, and Leventhal 2001).

In their work, Sime, Green and Guest compare the behaviour of programmers when presented with two different forms of a conditional construct, the if-then-else form (named NEST) and the branch-to-label form (named JUMP). The authors constructed two micro-languages using the NEST and JUMP forms respectively. They then conducted a controlled experiment, in which participants were given a simple program description in English which they needed to translate into one form, and then after a week to translate the same English description into the other form. The experimenters recorded the number of problems that the participants did not correctly solve the first time, as well as the time to complete the tasks.

The experiment clearly identified that the NEST form is a statistically significant improvement over the JUMP form, resulting in increased task completion rate and fewer semantic errors where the program the participant submitted for execution did the wrong thing.

This experimental approach is borrowed from cognitive psychology. It follows the methodology of considering a particular property that is common in a number of languages, constructing a micro-language that stresses the property and conducting an empirical experiment for measuring the effect of those properties. The authors hope that

‘*By devising micro-languages exemplifying particular features of interest, knowledge can be gained which allows more informed decisions to be made when designing new languages.*’ - (Sime et al. 1973).

Green and others expanded on this strategy in subsequent work on e.g. the delimitation of scopes (Sime, Green, and Guest 1977). This arc of work not only set the foundation for later work in HCI, but also established an empirical perspective on the evaluation of candidate language features.

Just as Evershed and Rippon were hopeful of increasing the access to computers for a broader section of the population, Sime et al. were concerned with the importance of studying the usability of computers and of programming systems by ‘non-specialists’. In a practice that would be followed by many others, the population of non-specialists they used as test participants were university students.

4. Broadening interest (1973-1986)

After the publication of the Sime et al’s work there was a growth of interest in the application of empirical techniques, especially the relatively informal Software Psychology society convened by Ben Shneiderman (Shneiderman 1986). In his book *Software Psychology*, Shneiderman (Shneiderman 1980) outlines many perspectives on the problem of creating software. He defines software psychology, a term he attributes to Tom Love, as “the study of human performance in the using computer and information systems” - but concentrates on software development rather than software use.

Shneiderman outlines a number of the methodologies in use for this research including introspection, detailed analysis of recordings of activities, called protocol analysis, discussion of code corpora such as (Knuth 1971), and controlled experiments. Shneiderman also deconstructs the idea of ‘programming’ as an activity, instead describing it as being composed of the tasks of Learning, Design, Composition, Comprehension, Testing, Debugging, Documentation and Modification.

Much of the book is concerned with what would become the study of software engineering, including organisational and managerial concerns. The material concerned with the design of language features is primarily discussed in Chapter 4 - ‘Programming Style’. It documents experiments concerned with the impact of comments on comprehensibility, (unpublished work by Peter Newsted) and apparently contradictory results on the effect of variable naming schemes ((Weissman and University of Toronto. Computer Systems Research Group 1973)), null results on the effects of indentation on comprehension of FORTRAN programs ((“RELATING INDIVIDUAL DIFFERENCES IN COMPUTER PROGRAMMING PERFORMANCE TO HUMAN INFORMATION PROCESSING ABILITIES” 1977)) and a extended discussion of choices of control structures.

At the end of this chapter, Shneiderman’s advice for designers of languages is that they *‘should recognize the that specific features may have a statistically significant effect on performance and should thoroughly test alternative proposals’* ((Shneiderman 1980), p90)

It is clear from the number of studies referred to in this section that by 1980 there was a growing community of research around programming language usability, and whilst advice about specific features such as jumps or blocks could be given, the broad advice at this point was that it was important to test the usability properties of the various choices in languages.

The work between 1973 and 1980 was criticised by (R. E. Brooks 1980) and (Sheil 1981). Brooks raised concerns that the heterogeneous nature of the total population of programmers results in experiments needing extremely large samples to achieve significance. He highlights a specific source of variation associated with the use of timing measures, observing that the time to solve the problem is interleaved in a non-trivial way with the time to understand the question, and that the effect size is diluted by the participants needing to understand concerns not strictly relevant to the test, for example understanding a library used in the stimulus material, when the purpose of the experiment was actually instruction flow.

Brooks also mentions a range of issues with the external validity of the experimental designs; he is concerned that a common practice to reduce variation, using students as participants, results in studies where the sample is not representative of the larger population of programmers. Brooks observed that the programs in the stimulus materials at the time were all under 500 lines, compared to contemporary commercial programs which were in excess of a million lines. He questions the validity of the experimental measures used, suggesting that timing studies may not be representative of a real value property. For example, measuring only development time may exclude considerations of quality, and debugging exercises may not successfully entail that the participant understands the program and so on.

The broad direction of this criticism is that designing an experiment which has a statistically significant effect is hard, and even when it is achieved, it can be difficult to know if the observed effect generalises to the practice of programming and software engineering.

Brooks is outlining, in 1980, one of the central motivating concerns of our work: knowing how to use information about programmers to improve design in a way that has validity outside of the experimental context. Brooks' response to these problems was to advocate the development of models of the cognitive processes involved in programming to understand and manage the methodological issues:

"*What approaches, then, show promise? Any successful characterization of the program-programmer interaction will probably be based on a model of the process or processes used by a programmer in interacting with a program. The development of such theories or models of the cognitive processes involved in programming is, therefore, likely to be a prerequisite to progress on these methodological issues.*" (R. E. Brooks 1980)

In a lengthier critique (Sheil 1981) outlines not only concerns about methodology but also about the lack of impact of the psychological research on the computing community. Sheil begins by outlining the state of the art in 1981:

"*As practiced by computer science, the study of programming is an unholy mixture of mathematics (e.g. (Edsger Wybe Dijkstra 1997)), literary criticism (e.g. (Kernighan and Plauger 1974)) and folklore (e.g. (F. P. Brooks 1975)). However, despite the stylistic variation, the claims that are made are all basically psychological; that is, that programming done in such and such a manner will be easier, faster, less prone to error, or whatever*" (Sheil 1981), [citation format updated]

Sheil's argument, however, is not that these alternative styles are problematic, it is that the direct psychological study of languages as it was performed in 1981 was problematic.

Sheil points out several underlying patterns: that results that might appear strong for novices quickly vanish as the participants become more experienced, that it is easy to measure the effect of pathological designs but multiple practical alternatives are often difficult to distinguish, and that it is difficult to separate out the ways in which a treatment, such as the introduction of static typing, which

Sheil argues entails the introduction of structural typing, affects the subject performance. Sheil continues enumerating other methodological challenges to the experimental designs and reporting of the experiments published at that point, the strongest critique however is in the general handling and deployment of evidence in support of design.

"*Yet, as Shneiderman sadly notes in a retrospective of this work, "Flowchart critics cheered our results as the justification of their claims, while adherents found fault and pronounced their confidence in the utility of flowcharts in their own work" (Shneiderman 1980). It is in response to reactions like this that the use of psychology in computer science debates was earlier characterized as "ammunition.*" (Sheil 1981)

Sheil is concerned that, as the actually empirically supported conclusions are too weak, there is a tendency to overreach, blurring what is known with what is conjecture.

"*Another consequence of this dependency on computing is that behavioural researchers tend, possibly in an attempt to make their work appeal to computer scientists, to generalize far beyond their data… [discussing (Shneiderman 1980)] Detailed discussions of experimental results are interleaved with totally (empirically) unsupported opinions on programming style. Much of this material would be quite legitimate, intuitively based argument in a computer science debate. However its presentation as part of a discussion of empirical research completely blurs the distinction between data and intuition, inviting readers to reject data that do not support their preconceptions. This makes the entire empirical enterprise moot*". - (Sheil 1981)

And

"*The absence of a critical review process, coupled with the very considerable difficulty of research in this area and the constant tendency to drift into intuitively based argument and generalize far beyond what has been established, has created a pseudopsychology of programming*" (Sheil 1981)

Similar to Brooks, Sheil argues that the most pressing need is to establish a theory of programming skill.

"*The experimental investigation of such factors as the style of conditional notation is premature without some theory which gives some account of why they might be significant factors in programmer behavior.*" (Sheil 1981)

In a retrospective in 1986, (Curtis 1986) argues that these methodological critiques were coupled with a problem of the better designed studies tending to come after decisions had already been taken:

'*Computer science was little interested in weak empirical justifications for directions it had already taken, such as structured programming. Computer scientists cared more for deductive proofs than for the rejection of null hypotheses.*' (Curtis 1986)

Looking back, Curtis sees 1981 as a turning point, that in response to Sheil and Brooks future research of the field adopted a cognitive psychological approach rather than a human factors approach. However, Curtis somewhat laments that cognitive psychology was chosen. At this point in time, the first workshop on the Empirical Studies of Programmers was organised, and it was here where Curtis was presenting his retrospective.

5. Community of study: Empirical Studies of Programmers (1986 - 1997)

Although the most prominent spinout from the Software Psychology society was the ACM CHI series ((Shneiderman 1986)), a smaller group was convened to continue the specific focus on programming. Empirical Studies of Programmers (ESP)[1] first met in June 1986 in Rosslyn VA ((Shneiderman 1986)). The preface of the proceedings outlines the purpose of the research as guiding interventions to improve practice:

'*Broadly speaking, the basic assumption of researchers who study programmers is this: By understanding how and why programmers do a task, we will be in a better position to make prescriptions that can aid programmers in their task. For example, if we can understand how a maintainer, say, goes about comprehending a program, we should be in a good position to recommend changes in documentation standards that would enable the maintainer to more effectively glean from the documentation the necessary information. Similarly, recommendations for software tools and education should also follow.*' ((Soloway and Sitharama Iyengar 1987), pvii).

They outline a number of the challenges of carrying out experiments in this area, many of which persist to the current day. One is for the research to be multidisciplinary, requiring 'a healthy degree of sophistication in both programming and psychology in order to recognize what are the important research issues'. They also acknowledge a limitation of the work contained in the proceedings: that the research was examining 'programming in the small' whilst million line programs could be found in industry and 'software of such magnitude has not as yet received significant attention by researchers in the field'.

Curtis ((Curtis 1986)) shares this concern about the ecological validity of the studies carried out, suggesting that the series might be called "Empirical Studies of Student Programmers" or that the work will continue to be retrospective in focus on 'demonstrating already established cognitive phenomena' (p257), raising doubts as to the justification for experimental studies as a cost-effective method of influencing design.

Curtis argues for increased ecological validity in studies, considering not only the individual's cognition but also the broader organisational context in which programming takes place. (Soloway 1986) concludes the proceedings discussing an agenda for the continued importance of programming-in-the-small to identify 'baseline-issues', that is to catalogue the important and recurrent behaviours of programmers as well as to develop confidence in the research methodologies. He also goes on to suggest that controlled studies may 'not be that useful for initially studying programming-in-the-large. This type of methodology requires carefully worked out hypotheses be developed first, before the experiment.', going on to state that 'Almost by definition, programming-in-the-large violates the basic premises for a controlled study.' ((Soloway 1986), p266). Instead he suggests that alternative techniques such as 'talking-aloud' may be a richer source of data for theory building.

[1] http://www.ppig.org/news/2006-06-01/whatever-happened-empirical-studies-programmers

Even at this very early stage of the field a number of techniques and questions are outlined. For example, (Onorato and Schvaneveldt 1986) considers, in the proceedings for the first Empirical Studies of Programmers workshop, the difference in exploratory behaviour for communicating how to do something between programmers and non-programmers. This kind of question is now important in discussions of how to support 'end-user programmers' in an organisational context, as identified in both MacLean et al's pioneering Buttons project ((MacLean et al. 1990)) and Nardi's seminal text ((Nardi 1993)).

Similarly, (Spohrer and Soloway 1986) describe a project to understand high-frequency bugs, showing that there are bugs that occur very commonly in novice programs (such as off-by-one errors), but also that these are not the result of novices misunderstanding a language feature. They conducted this experiment by gathering a corpus using a rudimentary form of instrumented tooling. They augmented the operating system of the computer their participants were using (a VAX 750), and obtained a copy of each syntactically correct program submitted for compilation. They refer to this data as on-line protocols.

By the conclusion of the first workshop of the Empirical Studies of Programmers, many of the core methodological tensions (e.g. internal validity vs; ecological validity, and what form of evidence needs to be in place to have an effective impact on language design) were already established.

Over the course of the subsequent workshops these issues continued to be explored with the balance shifting towards the study of professionals (by ESP 5, 70% of subjects in studies were professionals, compared to 21% in ESP 1). The series concluded with what would have been ESP 8. The submissions were included in a special issue of the International Journal of Human-Computer Studies (IJHCS Volume 54, No 2, Feb 2001). As the Editorial of this final issue suggests (Katz, Petre, and Leventhal 2001), the need for research into the Empirical Studies of Programmers had grown as wider populations engaged in programming ranging from the growing business applications, to educational and end-user programming fields. However, after this 8th workshop, some of the core researchers moved into other communities.

The role of a US based workshop considering the empirical investigation of professional programmers has been taken up by the Workshop on Evaluation and Usability of Programming Languages and Tools (PLATEAU) series discussed below. Human factors and HCI aspects have also been strongly represented at the Visual Language and Human Centric Computing (VL/HCC) IEEE symposium series, especially since the addition of "HCC" to the original focus on descriptions of novel kinds of visual syntax, which had an implicit concern with improving usability while seldom actually testing this ((A. F. Blackwell 1996)).

6. PLATEAU (2009 - present)

PLATEAU started in 2009 with the goal of considering and improving the efficiency of programmers by improving the usability of the languages and tools they develop with (Anslow, Markstrum, and Email: 2009). It was explicitly aimed at improving the visibility of the work on Human Computer Interaction to the programming language community. As such, PLATEAU tends to meet co-located with major software engineering conferences such as ICSE or OOPSLA/SPLASH.

PLATEAU has taken a software engineering perspective on the questions that ESP was interested in. It has a similar focus on professional and expert study (60% of studies that explicitly reported their sample, were of professional programmers compared to 23% at PPIG, discussed below).

The work that is published at PLATEAU includes corpus analyses, for example (Pritchard 2015) models the distribution of error messages from a system for novices learning to program as a precursor to systematically improving their usability. They compare their system for writing Python (Pritchard and Vasiga 2012) with data from Blackbox, which records data from the BlueJ Java editor (Brown et al. 2014).

PLATEAU also published empirical investigations into tools usage such as (Sadowski and Yi 2014)'s qualitative study into how developers at Google use tools for concurrent race detection, (Kabáč, Volanschi, and Consel 2015)'s controlled experiments with four professionals evaluating the ease of learning of DiaSuite, or (Campusano et al)'s controlled experiment investigating preference, developer productivity and comprehension in live programming robots.

Work at PLATEAU is often focussed around evaluation techniques such as (Hanenberg and Stefik 2015)'s call to build a community standard for the design of controlled experiments. (Kurtev, Christensen, and Thomsen 2016) outline a methodology for supporting the design of iterative or incremental improvements to programming languages. They observe that the 'evaluation' step of the incremental design of languages is often prohibitively expensive, especially if large scale empirical studies are used. Instead they propose a 'discount usability evaluation' method conducting a simple test observing participants completing basic tasks and categorising the problems they experienced by severity.

As well as these discussions and publications of empirical results on how programming is done, PLATEAU also provides a venue for higher level discussions such as (Ko 2016)'s description of the socio-technical roles of programming languages.

As would be expected given the focus on professional users, the PLATEAU community has in general been focussed on conventional uses of programming. This is in contrast to the sibling community of ESP and PLATEAU in Europe, PPIG, which has often provided a venue for the discussion of avant garde uses of programming such as bricolage programming (McLean and Wiggins 2010), choreography (Church, Rothwell, and Downie 2012) and education via robotics (Martin and Hughes 2011).

8. What works - the 'language wars' (2014 - present)

In 1981 Sheil criticised the construction of a 'pseudoscience of programming' through a combination of a lack of theory development, poor experimental practice and systematic overreach from weak empirical positions. Since 2014 there has been a resurgence of critique along a similar line, notably from Stefik and Hanenberg, that the standards of evidence used in programming research is inadequate, especially in contrast to the use of Randomised Control Trials (RCTs).

In the earlier critique, Sheil was primarily concerned with the integrity and scientific standing of programming language research, a concern that Stefik and Hanenberg also share. In ((Andreas Stefik et al. 2014)) they describe a systematic review of the papers in PPIG and PLATEAU, coded according to their own scheme for categories and using criteria from the campaign for evidence-based policy in education research in the USA known as the What Works Clearinghouse. As described in a critical evaluation of that campaign (an evaluation which the author claims was suppressed by the federal funders of WWC), in asking why it had attracted so much controversy, 'The answer to these questions can be summed up in two words: "math wars."' (Schoenfeld 2006).

The goal of the WWC in USA educational policy had been to ensure that educational initiatives resulted in quantifiable improvements, applying the same logic as the use of RCTs in publicly-funded health interventions. Although driven by a recognisable political agenda, education researchers are well aware of the futility of attempting RCTs in children's education, for factors that include the primarily political and social drivers of educational outcomes (e.g. race, gender, social deprivation) that political actors hope not to draw attention to when advocating curriculum change or technological intervention as a less costly panacea.

The desire for more clear cut quantifiable evidence in programming language design was also associated with a particular approach to programming education, hoping to demonstrate that Stefik's Quorum language (2017) had a more solid scientific basis than the more popular Scratch. Scratch had been introduced to schools from the more arts-based tradition of live, creative coding, rather than a focus on teaching conventional syntax, and this was a matter of concern for computer scientists who, as had occurred earlier with Dijkstra's polemic against BASIC, were concerned that the more easily-learned graphical syntax would damage students' understanding. Stefik originally implemented Quorum (and its predecessors) to support the needs of students with disabilities, but developed this into a campaign to make Quorum the first language for which every feature could be justified with a scientific study.

Advocates of the evidence-based approach, meeting the "what works" standards, focus on technical properties of programming language design, for example explicitly labelling the discussion of self-efficacy, tooling support, education or program comprehension as not related to programming language design ((Andreas Stefik et al. 2014), p228). Instead they concentrate on properties such as type systems, syntax and API design. In a direct challenge to established methods in Psychology of Programming, Stefik and colleagues concluded that only 1.1% of PPIG papers were both related to programming language design (by the above definition), and also met the standards proposed by the What Works Clearinghouse. They report that the corresponding number for Plateau is 14.3% and 16.7% for ESP.

They argue that this represents a fundamental weakness in programming language design research. However, they go on from there to suggest that this weak evidential base for language design may be an important cause of long-standing controversy that they describe as 'language wars' (Andreas Stefik and Siebert 2013)), which they argue have had substantial, negative, social consequences caused by the effort of creating, learning and adopting multiple languages. As an alternative they advocate for methodologies that contribute "what works" style evidence, such as RCT's of specific features. Examples of this approach based on RCTs are found in (Uesbeck et al. 2016), (Endrikat et al. 2014), (Andreas Stefik and Siebert 2013).

In the math wars in education research that the "language wars" phrase alludes to, the political drivers reflect conservative advocacy of labour utility, as imagined through the rote classroom "labour" of skill acquisition, practice drills, and quantitative assessment of the numbers of uniformly correct answers, as contrasted with a vision of education that emphasises creative experience and diversity of assessment. In education, these two poles are associated with alternative scientific agendas - on the creative side, qualitative and interpretive research that is politically informed, while on the utilitarian side, research emphasises quantitative assessment and RCTs to verify reductionist cognitive or perceptual models.

The same dynamics can be seen in programming language research, where a utilitarian labour focus emphasises the number of correct actions taken by the programmer implementing a well-defined specification, rather than programming as an exploratory, creative and diverse experience. As programming language research becomes more human-centric, in the recognition that the language is a user interface and that some account needs to be taken on the user, these two alternative perspectives have led to the language wars campaigners suggesting that programming language design should prefer some HCI research methods, but not others.

Large scale software application deployments do use RCT-like A/B tests to compare interface design alternatives, where there is a clear productivity measure that can be used by the company (e.g. numbers of sales or click-throughs). We note that these methods are especially relevant to *incremental* optimisation of programming language designs. This is in contrast to research such as that in the Visual Languages or Live Coding communities that focuses on novel paradigms, new styles of representation, and applications beyond traditional waterfall-style software engineering. Of course, the focus on novelty within VL/HCC may indeed miss opportunities for optimisation of existing language designs, which often have points of detail that could be improved through application of RCTs. A point of particular relevance for incremental RCTs is decisions that relate to the surface syntax of programming languages and environments. There are many details of syntax that have been chosen on an arbitrary basis, without clear evidence for the choice made. When supported by empirical evidence, we are able to determine, for example, whether a special assignment operator corrects frequent misconceptions that arise from misuse of the equals sign ((Mc Iver, n.d.)).

The natural desire of computer scientists to produce quantitative accounts of human behaviour ((A. F. Blackwell 2022)), combined with the particular kind of political drivers that expect educational policy to produce a mechanically trained, yet disempowered, workforce ((Hicks 2017)), combine in the software industry with those large companies whose business model require measurement and control of user's attention ((Zuboff 2019), (Seaver 2022)). The result for the psychology of programming has been a constant struggle to take a well-informed approach to design guidance.

8. Cognitive dimensions (1989 - present)

The paper describing the broad layout of the language wars (Andreas Stefik and Hanenberg 2014) raises a number of questions that overlap with our research interests. Perhaps the largest point of difference is that they are particularly upset about the widespread use of Thomas Green's Cognitive Dimensions of Notations framework ((Green 1990), (Green and Petre 1996), (Hadhrawi, Blackwell, and Church 2017)). The emphasis of CDs on the importance of the tool and environment suggested that reductive controlled comparison of syntax choice as the primary scientific agenda in language design might not actually be the most important question to investigate. The emphasis of CDs on the need for different solutions to different problems, insisting that a language should only be evaluated in relation to a particular profile of activities, also seemed contrary to the desire to identify objectively "best" features.

(A. Stefik and Hanenberg 2017) write:
For example, one common approach that we think lacks merit is the so-called cognitive dimensions of notations framework, a set of design principles conceived by Thomas R.G. Green in 1989 and expanded in a 1996 article. According to Google Scholar this influential article has been cited some 500 times, but Green's theory wasn't based on sound empirical evidence--by 1989 there had only been seven programming language design studies.

Emphasis on citation counts as a measure of scientific quality is not one that we would advocate, especially since the publication describing Stefik's own Quorum language ((Andreas Stefik and Ladner 2017)) has only been cited 23 times, which is less influential than we would hope for "the first language to use human-factors evidence from both field data and randomized controlled trials in its design" (the "language wars" paper has been cited many more times than the language itself, which certainly does not do justice to the original scientific or design objectives of Quorum).

The language wars authors were not the first to have criticised the intentions and scientific significance of Cognitive Dimensions. Green's work at the MRC Applied Psychology Unit, had been intended precisely to use empirical human factors techniques as a basis for engineering design, along with much other research that made the MRC-APU one of the founding centres of HCI research internationally ((Craik 1944), (Reynolds and Tansey 2003)). It was after 20 years of work on this problem that Green came to realize individual controlled experiments were not a practical or effective source of design guidance for the designers of new interactive systems. The intentions of Cognitive Dimensions were precisely to avoid the "death by detail" that made it infeasible to address design problems in this way, instead offering a theoretically informed "broad brush" vocabulary with which designers could be encouraged to discuss decisions and trade-offs that were cognitively relevant.

Some younger researchers have perceived this response to the need for "broad brush" design guidance as a lack of scientific rigour. In particular, those coming from human-factors engineering or business productivity contexts have hoped that problems of design could be reduced to more quantified performance formulas or objective observations of what is right. This was the driver for the "Physics of Notations" proposed by business school lecturer Daniel Moody ((Moody 2009)) - a critique that is even more highly cited than the above quote, even more critical of the supposedly scientific status of CDs (keeping in mind that CDs was always intended to be a resource for designers, not a scientific theory), and also driven by the desire for objective and measurable criteria. As it turns out, much of Moody's alternative "physics" was no less subjective than the CDs themselves, since supposedly "physical" facts such as semantic interpretation are always dependent on the observer. The Physics of Notations has become far more widely cited than CDs, and certainly has greater appeal to quantitative PL researchers, but meta-analysis of the many studies citing this work show that they are just as lacking in scientific rigour and replicability as Moody felt that Green had been ((Linden and Hadar 2018)).

The desire for a physics-based set of design principles for PLs, or a laboratory controlled trial for choosing the objectively best features, is typical of first-wave HCI ((Bødker 2006)), with its emphasis on human factors as an optimisable component within an engineering system. Perhaps CDs might be considered a part of the turn toward context that was characteristic of second-wave HCI. The possibility of focusing on a more diverse variety of creative experiences, including artistic practices as well as utilitarian ones, is a focus more characteristic of third-wave HCI. Blackwell and Fincher's suggestion that CDs might be regarded as a pattern language describing Patterns of User Experience (PUX) ((A. F. Blackwell and Fincher 2010)), offers a third-wave alternative to the design orientation of the field ((A. Blackwell 2015)), following first-wave Physics of Notations and second-wave original CDs.

9. Conclusion: Lack of consensus on how to improve human centric programming

As the breadth of the discussion above shows, in the 50 years since Dijkstra's speculation on the socio-cognitive implications of the goto statement, no consensus has arisen as to where and how to go about improving programming language usability.

Existing communities lend varying degrees of support to a collection of different methods: PLATEAU supports empirical methods for discussing professional software engineers, ICLC focuses on practice-based research for supporting artist programmers, and PPIG accepts contributions in a wide variety of forms ranging from participant ethnographies, to design discussions, theoretical framework and philosophical speculations.

There is an ongoing tension within the community as to what the scientific status of the research is, and what techniques are appropriate to address these problems. Even more so, there are tensions between what are the right questions for the field to try and address.

This brief history of the study of human centric programming language design has provided the background to those tensions, to be used as a reference for the ongoing work in our group, looking at what kinds of questions programming language designers face, what techniques are currently available for answering them, and what is missing

10. Acknowledgements

The authors would like to thank the community for many years of interesting discussions

11. References

Anslow, Craig, Shane Markstrum, and Emerson Murphy-Hill Email: 2009. "Evaluation and Usability of Programming Languages and Tools (PLATEAU) PLATEAU." https://ecs.wgtn.ac.nz/foswiki/pub/Main/TechnicalReportSeries/ECSTR10-12.pdf.

Arawjo, Ian. 2020. "To Write Code: The Cultural Fabrication of Programming Notation and Practice." In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, 1–15. CHI '20. New York, NY, USA: Association for Computing Machinery.

Blackwell, Alan. 2015. "Patterns of User Experience in Performance Programming." In . Zenodo.

https://doi.org/10.5281/zenodo.19315.
Blackwell, Alan F. 1996. "Metacognitive Theories of Visual Programming: What Do We Think We Are Doing?" In *Proceedings of the 1996 IEEE Symposium on Visual Languages*, 240. VL '96. USA: IEEE Computer Society.
———. 2017. "6,000 Years of Programming Language Design: A Meditation on Eco's Perfect Language." In *Conversations Around Semiotic Engineering*, 31–39. Cham: Springer International Publishing.
———. 2022. "Wonders without Number: The Information Economy of Data and Its Subjects." *AI & Society*, January. https://doi.org/10.1007/s00146-021-01324-8.
Blackwell, Alan F., and Sally Fincher. 2010. "PUX: Patterns of User Experience." *Interactions* 17 (2): 27–31.
Blackwell, Alan F., Marian Petre, and Luke Church. 2019. "Fifty Years of the Psychology of Programming." *International Journal of Human-Computer Studies* 131 (November): 52–63.
Bødker, Susanne. 2006. "When Second Wave HCI Meets Third Wave Challenges." In *Proceedings of the 4th Nordic Conference on Human-Computer Interaction: Changing Roles*, 1–8. NordiCHI '06. New York, NY, USA: Association for Computing Machinery.
Brooks, Frederick P. 1975. *The Mythical Man-Month : Essays on Software Engineering*. Reading, Mass.: Addison-Wesley Pub. Co.
Brooks, Ruven E. 1980. "Studying Programmer Behavior Experimentally: The Problems of Proper Methodology." *Communications of the ACM* 23 (4): 207–13.
Brown, Neil Christopher Charles, Michael Kölling, Davin McCall, and Ian Utting. 2014. "Blackbox: A Large Scale Repository of Novice Programmers' Activity." In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, 223–28. SIGCSE '14. New York, NY, USA: ACM.
Church, L., N. Rothwell, and M. Downie. 2012. "Sketching by Programming in the Choreographic Language Agent." *Proceedings of the*. https://pdfs.semanticscholar.org/17c3/e3b7530bc25c9e45fa4f7430a3fc54ba3db4.pdf.
Craik, K. J. W. 1944. "Medical Research Council Unit for Applied Psychology." *Nature* 154: 476–77.
Curtis, Bill. 1986. "By the Way, Did Anyone Study Any Real Programmers?" In *Papers Presented at the First Workshop on Empirical Studies of Programmers on Empirical Studies of Programmers*, 256–62. Norwood, NJ, USA: Ablex Publishing Corp.
Dijkstra, Edsger W. 1968. "Letters to the Editor: Go to Statement Considered Harmful." *Communications of the ACM* 11 (3): 147–48.
Dijkstra, Edsger Wybe. 1997. *A Discipline of Programming*. 1st ed. Upper Saddle River, NJ, USA: Prentice Hall PTR.
Endrikat, Stefan, Stefan Hanenberg, Romain Robbes, and Andreas Stefik. 2014. "How Do API Documentation and Static Typing Affect API Usability?" In *Proceedings of the 36th International Conference on Software Engineering*, 632–42. ICSE 2014. New York, NY, USA: ACM.
Evershed, D. G., and G. E. Rippon. 1971. "High Level Languages for Low Level Users." *Computer Journal* 14 (1): 87–90.
Goldstine, H. H., and J. von Neumann. n.d. "Planning and Coding of Problems for an Electronic Computing Instrument. Part II, Vol." *The Institute for Advanced Study Princeton, New*.
Green, T. R. G. 1990. "Cognitive Dimensions of Notations. People and Computers V: Proc. British Computer Society HCI'89 Conference." Cambridge University Press.
Green, T. R. G., and M. Petre. 1996. "Usability Analysis of Visual Programming Environments: A 'Cognitive Dimensions' Framework." *Journal of Visual Languages & Computing* 7 (2): 131–74.
Hadhrawi, Mohammad, Alan F. Blackwell, and Luke Church. 2017. "A Systematic Literature Review of Cognitive Dimensions." In *Proceedings of the 28th Annual Workshop of the Psychology of Programming Interest Group (PPIG 2017)*, 13.
Hanenberg, Stefan, and Andreas Stefik. 2015. "On the Need to Define Community Agreements for

Controlled Experiments with Human Subjects: A Discussion Paper.” In *Proceedings of the 6th Workshop on Evaluation and Usability of Programming Languages and Tools*, 61–67. PLATEAU 2015. New York, NY, USA: ACM.

Hicks, Mar. 2017. *Programmed Inequality: How Britain Discarded Women Technologists and Lost Its Edge in Computing*. MIT Press.

Kabáč, Milan, Nic Volanschi, and Charles Consel. 2015. “An Evaluation of the DiaSuite Toolset by Professional Developers: Learning Cost and Usability.” In *Proceedings of the 6th Workshop on Evaluation and Usability of Programming Languages and Tools*, 9–16. PLATEAU 2015. New York, NY, USA: ACM.

Katz, Irvin R., Marian Petre, and Laura Leventhal. 2001. “Editorial: Empirical Studies of Programmers.” *International Journal of Human-Computer Studies* 54 (2): 185–88.

Kernighan, Brian W., and P. J. Plauger. 1974. *Elements of Programming Style*. New York, NY, USA: McGraw-Hill, Inc.

Knuth, Donald E. 1971. “An Empirical Study of FORTRAN Programs.” *Software: Practice & Experience* 1 (2): 105–33.

Ko, Andrew J. 2016. “What Is a Programming Language, Really?” In *Proceedings of the 7th International Workshop on Evaluation and Usability of Programming Languages and Tools*, 32–33. ACM.

Kurtev, Svetomir, Tommy Aagaard Christensen, and Bent Thomsen. 2016. “Discount Method for Programming Language Evaluation.” In *Proceedings of the 7th International Workshop on Evaluation and Usability of Programming Languages and Tools (plateau 2016)*. Association for Computing Machinery. http://people.cs.aau.dk/~bt/PLATEAU2016/Preprint-PLATEAU2016-KurtevChristensenThomsen.pdf.

Linden, D. Van Der, and I. Hadar. 2018. “A Systematic Literature Review of Applications of the Physics of Notation.” *IEEE Transactions on Software Engineering*, 1–1.

MacLean, Allan, Kathleen Carter, Lennart Lövstrand, and Thomas Moran. 1990. “User-Tailorable Systems: Pressing the Issues with Buttons.” In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 175–82. CHI ’90. New York, NY, USA: Association for Computing Machinery.

Martin, C., and J. Hughes. 2011. “Robot Dance: Edutainment or Engaging Learning.” *Proceedings of the 23rd Psychology of Programming*. http://www.ppig.org/papers/23/14%20Martin.pdf.

Mc Iver, Linda. n.d. “The Effect of Programming Language on Error Rates of Novice Programmers.” https://pdfs.semanticscholar.org/ac30/ee4129122006bbe2c1af6a935d958c416eb4.pdf.

McLean, A., and G. Wiggins. 2010. “Bricolage Programming in the Creative Arts.” *22nd Psychology of Programming Interest*. http://www.academia.edu/download/30255078/22nd-eup-2.pdf.

Menabrea, Luigi Federico, and Ada King Countess of Lovelace. 1843. *Sketch of the Analytical Engine Invented by Charles Babbage, Esq*. Richard and John E. Taylor.

Moody, Daniel. 2009. “The ``Physics'' of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering.” *IEEE Transactions on Software Engineering* 35 (6): 756–79.

Nardi, Bonnie A. 1993. *A Small Matter of Programming: Perspectives on End User Computing*. Cambridge, MA, USA: MIT Press.

Newell, Allen, and Stuart K. Card. 1985. “The Prospects for Psychological Science in Human-Computer Interaction.” *Hum. -Comput. Interact.* 1 (3): 209–42.

Onorato, Lisa A., and Roger W. Schvaneveldt. 1986. “Programmer/Nonprogrammer Differences in Specifying Procedures to People and Computers.” In *Papers Presented at the First Workshop on Empirical Studies of Programmers on Empirical Studies of Programmers*, 128–37. Norwood, NJ, USA: Ablex Publishing Corp.

Pierce, Benjamin C. 2002. *Types and Programming Languages*. 1st ed. The MIT Press.

Pritchard, David. 2015. “Frequency Distribution of Error Messages.” *arXiv [cs.SE]*. arXiv.

http://arxiv.org/abs/1509.07238.

Pritchard, David, and Troy Vasiga. 2012. “CS Circles: An In-Browser Python Course for Beginners.” *arXiv [cs.CY]*. arXiv. http://arxiv.org/abs/1209.2166.

“RELATING INDIVIDUAL DIFFERENCES IN COMPUTER PROGRAMMING PERFORMANCE TO HUMAN INFORMATION PROCESSING ABILITIES.” 1977. https://search.proquest.com/openview/c39f5fd5a80bb3296095014ef2d2b881/1?pq-origsite=gscholar&cbl=18750&diss=y.

Reynolds, L. A., and E. M. Tansey. 2003. *The MRC Applied Psychology Unit: V. 16*. Edited by L. A. Reynolds and E. M. Tansey. Wellcome Witnesses to Twentieth Century Medicine S. London, England: Wellcome Trust Centre for the History of Medicine at UCL.

Sadowski, Caitlin, and Jaeheon Yi. 2014. “How Developers Use Data Race Detection Tools.” In *Proceedings of the 5th Workshop on Evaluation and Usability of Programming Languages and Tools*, 43–51. ACM.

Schoenfeld, Alan H. 2006. “What Doesn’t Work: The Challenge and Failure of the What Works Clearinghouse to Conduct Meaningful Reviews of Studies of Mathematics Curricula.” *Educational Researcher* 35 (2): 13–21.

Seaver, Nick. 2022. *Computing Taste: Algorithms and the Makers of Music Recommendation*. University of Chicago Press.

Sheil, B. A. 1981. “The Psychological Study of Programming.” *ACM Comput. Surv.* 13 (1): 101–20.

Shneiderman, Ben. 1980. *Software Psychology: Human Factors in Computer and Information Systems (Winthrop Computer Systems Series)*. Winthrop Publishers.

———. 1986. “No Members, No Officers, No Dues: A Ten Year History of the Software Psychology Society.” *SIGCHI Bull.* 18 (2): 14–16.

Sime, M. E., T. R. G. Green, and D. J. Guest. 1973. “Psychological Evaluation of Two Conditional Constructions Used in Computer Languages.” *International Journal of Man-Machine Studies* 5 (1): 105–13.

———. 1977. “Scope Marking in Computer Conditionals—a Psychological Evaluation.” *International Journal of Man-Machine Studies* 9 (1): 107–18.

Soloway, Elliot. 1986. “What to Do next: Meeting the Challenge of Programming-in-the-Large.” In *Papers Presented at the First Workshop on Empirical Studies of Programmers on Empirical Studies of Programmers*, 263–68. Ablex Publishing Corp.

Soloway, Elliot, and S. Sitharama Iyengar. 1987. *Empirical Studies of Programmers (Human-Computer Interaction Series)*. Intellect.

Spohrer, James G., and Elliot Soloway. 1986. “Analyzing the High Frequency Bugs in Novice Programs.” In *Papers Presented at the First Workshop on Empirical Studies of Programmers on Empirical Studies of Programmers*, 230–51. Norwood, NJ, USA: Ablex Publishing Corp.

Stefik, A., and S. Hanenberg. 2017. “Methodological Irregularities in Programming-Language Research.” *Computer* 50 (8): 60–63.

Stefik, Andreas, and Stefan Hanenberg. 2014. “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*, 283–99. Onward! 2014. New York, NY, USA: ACM.

Stefik, Andreas, Stefan Hanenberg, Mark McKenney, Anneliese Andrews, Srinivas Kalyan Yellanki, and Susanna Siebert. 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*, 223–31. ICPC 2014. New York, NY, USA: ACM.

Stefik, Andreas, and Richard Ladner. 2017. “The Quorum Programming Language (abstract Only).” In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. New York, NY, USA: ACM. https://doi.org/10.1145/3017680.3022377.

Stefik, Andreas, and Susanna Siebert. 2013. “An Empirical Investigation into Programming Language

Syntax." *Trans. Comput. Educ.* 13 (4): 19:1–19:40.
Uesbeck, Phillip Merlin, Andreas Stefik, Stefan Hanenberg, Jan Pedersen, and Patrick Daleiden. 2016. "An Empirical Study on the Impact of C++ Lambdas and Programmer Experience." In *Proceedings of the 38th International Conference on Software Engineering*, 760–71. ACM.
Waite, William M., and Gerhard Goos. 1984. *Compiler Construction*. Springer-Verlag New York.
Weissman, Larry, and University of Toronto. Computer Systems Research Group. 1973. *Psychological Complexity of Computer Programs: An Initial Experiment*. Computer Systems Research Group, University of Toronto.
Zuboff, Shoshana. 2019. *The Age of Surveillance Capitalism: The Fight for a Human Future at the New Frontier of Power: Barack Obama's Books of 2019*. Profile Books.