A fox not a hedgehog: What does PPIG know?

Luke Church
Computer Laboratory
University of Cambridge
luke@church.name

Mariana Mărășoiu
Computer Laboratory
University of Cambridge
mariana.marasoiu@cl.cam.ac.uk

Abstract

We outline a thematic history of the Psychology of Programming Interest Group based on a coding of [~400] publications. We highlight the changing interests of the community, draw out trends and discuss missing topics. We compare and contrast with [~50] publications from the PLATEAU community. We find that, fox-like, PPIG has a very broad coverage of a wide range of topics. We characterise trends in these and go on to discuss missing topics and areas for future work.

1. Introduction

How people go about programming and the how to designing systems that support them is of growing interest once more to both the HCI and technical communities.

Victor (2012) helped restart the collective imagination of building live, usable, programming systems. Work on live programming has continued within the software engineering communities at LIVE at ICSE[1] and ECOOP[2], within the live coding scene via TOPLAP[3], the Dagstuhl on Live Coding (Blackwell et al., 2014) and the new conference series ICLC[4]. There are also other meetings interested in specific programming technologies, such as the workshop on block based languages (Turbak et al., 2015).

Further, as programming has become a required element of the curriculum in the United Kingdom there has been substantial interest via the Computing At Schools project (Peyton Jones, 2015).

Simultaneously, there has been a growing interest in the experience that developers have via workshops such as LIXD[5], and the new PX[6] workshop at ECOOP. There was recently a well attended group at the Special Interest Group of CHI 2016 on the Usability of Programming Languages (Myers et al., 2016).

Matters that concern the PPIG and PLATEAU (Sunshine et al., 2009) communities are on the rise again. At the same time, what constitutes knowledge within the communities is increasingly contested. For example, a substantial theme of discussion within the SIG-CHI was related to the work of (Stefik et al., 2014), where they conclude that little of the work in the PPIG and PLATEAU groups is about programming language design and meets their selected standard for empirical research (1.1% and 14.3% of papers respectively, within the constraints explained in their paper).

This leads to the question: if these communities haven't had much to say in the past about high quality randomized trials of programming language design, what do they talk about? In this work we explore this question.

[1] http://liveprogramming.github.io/2013/
[2] http://2016.ecoop.org/track/LIVE-2016
[3] http://toplap.org
[4] http://iclc.livecodenetwork.org
[5] http://www.lorentzcenter.nl/lc/web/2013/563/description.php3?wsid=563&venue=Snellius
[6] http://2016.ecoop.org/track/PX-2016

2. Analytical Methodology

2.1 The Corpora

We analyse papers available in digital form published at the annual workshops of the Psychology of Programming Interest Group (PPIG) and at the Workshop on Evaluation and Usability of Programming Languages and Tools (PLATEAU). Table 1. shows the number of papers for each conference that were analysed, organized by year.

	'92	'93	'94	'95	'96	'97	'98	'99	'00	'01	'02	'03	'04	'05	'06	'07	'08	'09	'10	'11	'12	'13	'14	'15
PPIG	12	-	6	21	20	11	18	19	18	22	18	19	22	26	21	20	20	16	17	22	17	-	24	11
PLATEAU																		8	10	11	5	-	10	12

Figure 1. Number of papers published and analysed from each conference, grouped by year.
There are 400 papers from PPIG and ?? from PLATEAU.

We make no distinction between ‘work in progress’, ‘graduate consortium submissions’ and ‘full papers’, but didn’t include the work in progress workshops of PPIG. The purpose of this analysis is to reflect on the communities interests, not quality control.

2.2 Data analysis

We used thematic analysis methods (Braun & Clarke, 2006) which we adapted to document analysis in order to generate topics. The analysis proceeded as follows. One author read through the titles of the papers in the corpora and extracted a set of research topics, which were discussed with the other author. These topics were then organized in larger themes by both authors. In the next step, the authors then independently read through the text of the papers with the dual purpose of assigning topics and generating new topics when necessary. Each paper was assigned multiple topics.

Since the coding scheme evolved in time, the corpora was searched again for some of the newly added codes as an extra check to ensure that the papers were labeled correctly. Papers found this way were reread and assigned potentially assigned the new code.

We only considered a code for the paper if it was the central theme, e.g. a language, for a programming language: where its features were discussed? Was it used in empirical or analytical studies described in the paper? We didn’t consider it as a topic if it was mentioned in passing, or if it was mentioned as the language that the system described was implemented in.

At this point an inter-rater reliability score for a 5% sample was computed giving a Kappa of 0.84, indicating broad agreement between the reviewers.

In order to improve the analysis, we then discussed the differences in the codes to reach consensus. Typically, we found this was resolved by adding codes to papers and occasionally by refining the coding scheme itself. Some of the outcomes of the discussions are included in a later section. Whilst this processes decreases the objectivity, it also decreases the likelihood of accidental oversights. In practice the authors found that accidental omission was by far the most common reason for the addition of new codes to a paper during the discussion phase.

2.3 Notes on the coding

We elected to engage in this coding activity rather than to use the keywords that the authors have tagged their work with due to many papers not having keywords and to allow a somewhat independent reflection on what the papers were about.

We initially intended to use Latent Dirichlet Allocation (LDA) (Blei et al., 2003) to extract topics automatically. This technique was inspired by the work of (Greenberg et al., 2015) in their analysis of the programming languages literature. However, the application of LDA resulted in poor topics for the PPIG corpus. Consequently we elected to identify and code the topics manually.

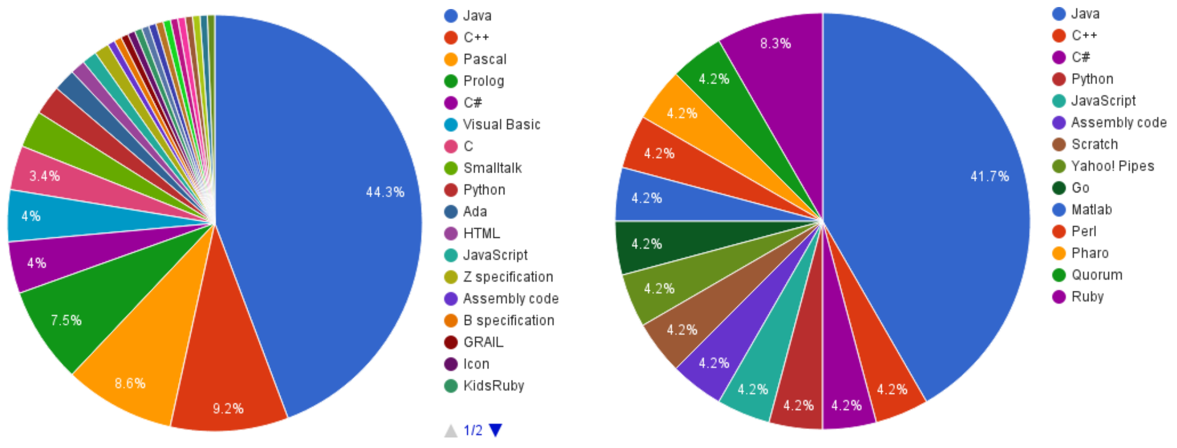
Both authors have developed and used open coding schemes in several analyses in past, but were struck by the difficulty of coding the PPIG papers. Over the 23 years covered in the analysis, the PPIG community does not seem to have converged on any particular strategy of narrative structure, and the lexicon for describing topics is still somewhat undecided. Moreover, the range of technologies, aspects and domains covered within the conference is very broad.

This intellectual diversity acts as a core strength of the community but, as we shall see, raises some challenges in the development of theory. Let's start by looking at the clearly identifiable themes and patterns.

3. Languages, People, Analyses

3.1 Languages

Looking first at which languages are studied, we see a fairly similar distribution between PPIG and PLATEAU. Java dominates both conferences, and the broader family of C-style languages even more so. There are a wider range of programming languages considered in PPIG than PLATEAU, but this be accounted for by their relative age.



(A) (B)

Figure 1. Languages discussed at PPIG (A) and PLATEAU (B)

Looking at how interest in the languages studied at PPIG evolved over time, we see a definite transition in [YEAR] when Java replaced a waning interest in Pascal and Prolog. Other languages such as C++ and Visual Basic show continued but lower intensity interest.

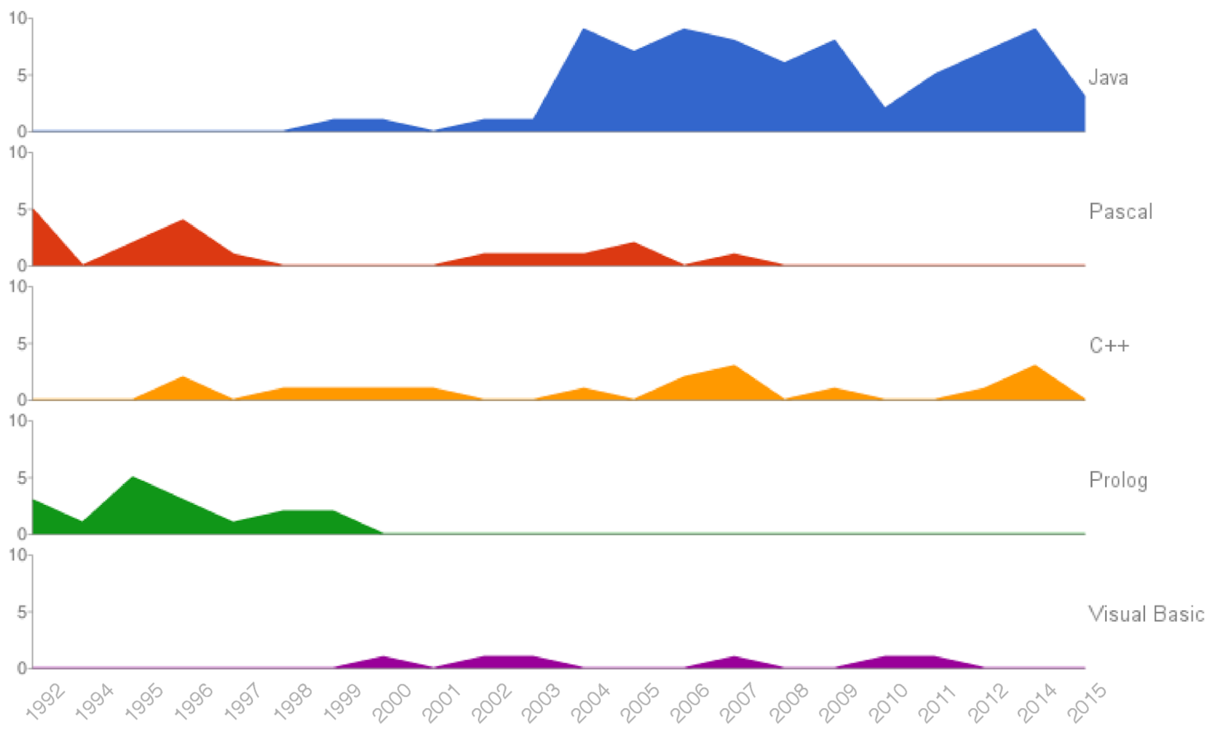


Figure 2. Languages discussed at PPIG over time

There isn't an easily available baseline to compare these to. There are a number of different schemes for assessing language popularity[7] . The TIOBE index[8], which lists languages by popularity, seems in general agreement with the Figure above, except with a considerably higher emphasis on C.

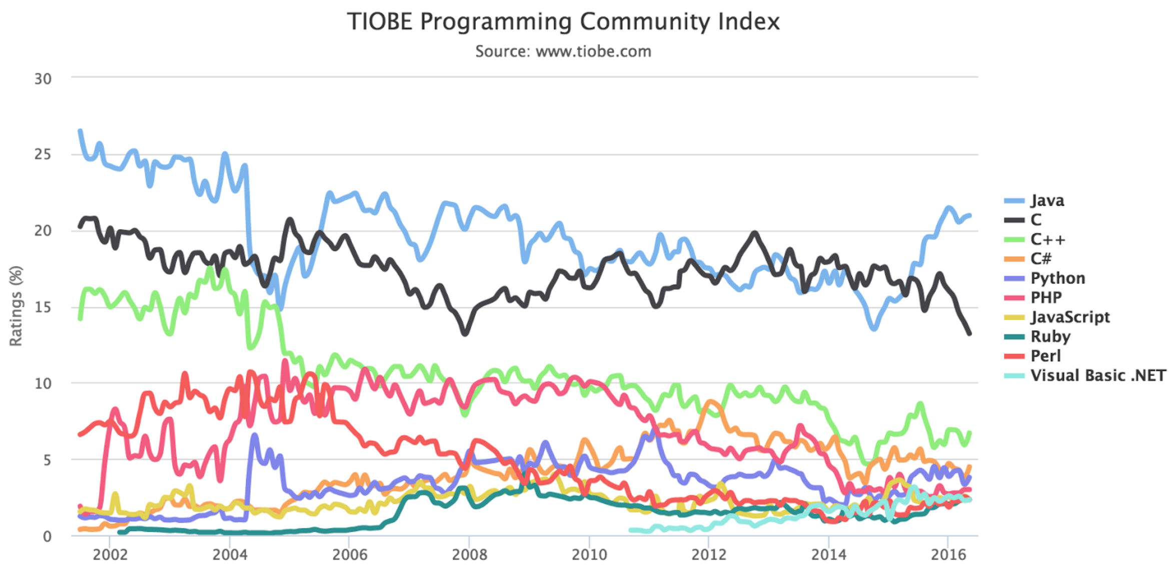


Figure 3. The TIOBE index of languages over time

[7] https://en.wikipedia.org/wiki/Measuring_programming_language_popularity

[8] http://www.tiobe.com/tiobe_index?page=programminglanguages_definition

3.2 Who are the programmers?

Looking at the programmers who were being studied at PPIG, we can categorise the papers broadly as either studies of professional or expert programmers (those whose primary occupation is programming), of end-user programmers, (those who are programming to support some other primary occupation), and of novices and those learning to program.

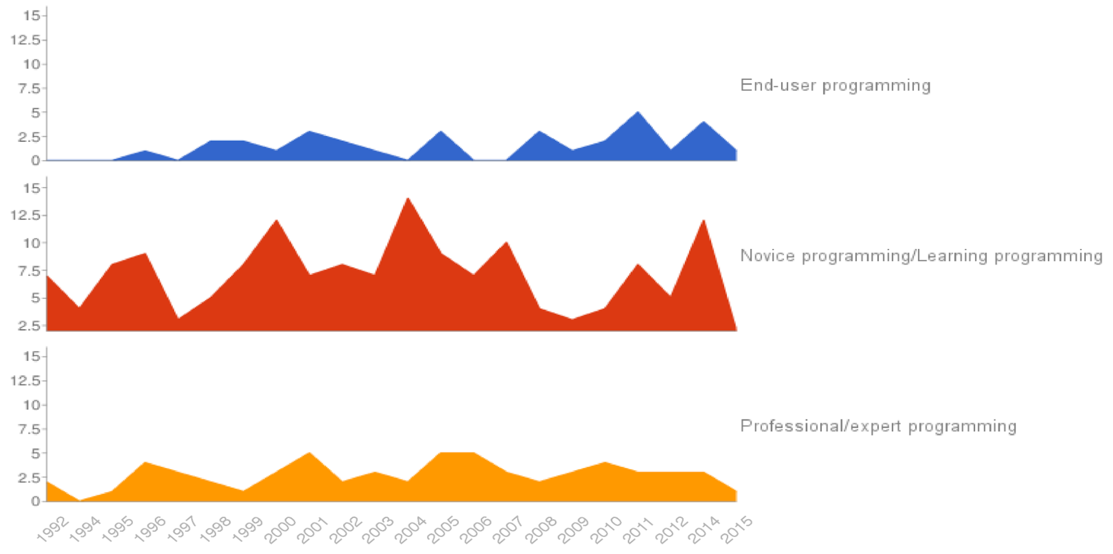
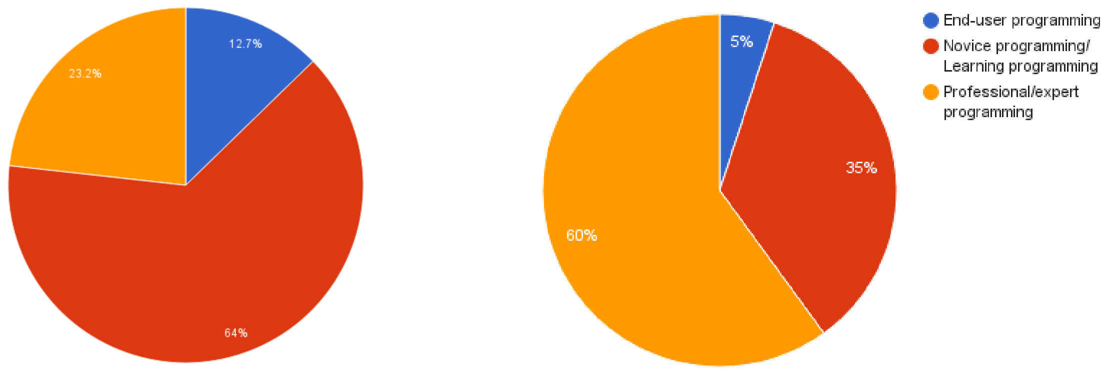


Figure 4. Interest in different programmer types over time

Papers were labeled as any of the three categories only when it was explicit which population was being studied.



(A) *(B)*

Figure 5. The distribution of programmer types being studied at PPIG (A) and PLATEAU (B)

Contrasting the two conferences, we can see that PPIG has been more interested in novice programming and PLATEAU more in professional programmers.

3.3 What techniques are used for discussing these programmers?

We broadly categorised the techniques used into analytical, empirical, qualitative and qualitative. These are non-exclusive labels, meaning that a study can use a combination of these techniques and receive a combination of labels. We can see that analyses within these communities is predominantly empirical (231 papers and 38 papers for PPIG and PLATEAU) as opposed to analytical (29 papers and

5 papers), with a balance of qualitative methods (108 papers and 18 papers) and quantitative methods (138 papers and 24 papers).

The communities start to vary in the techniques they are using for performing the analysis. Considering analytical techniques, there are insufficient PLATEAU papers in this category to make further separation useful. However, within the PPIG community, there are several analytical and theoretical frames that have been used repeatedly, as can be seen in the figure below.

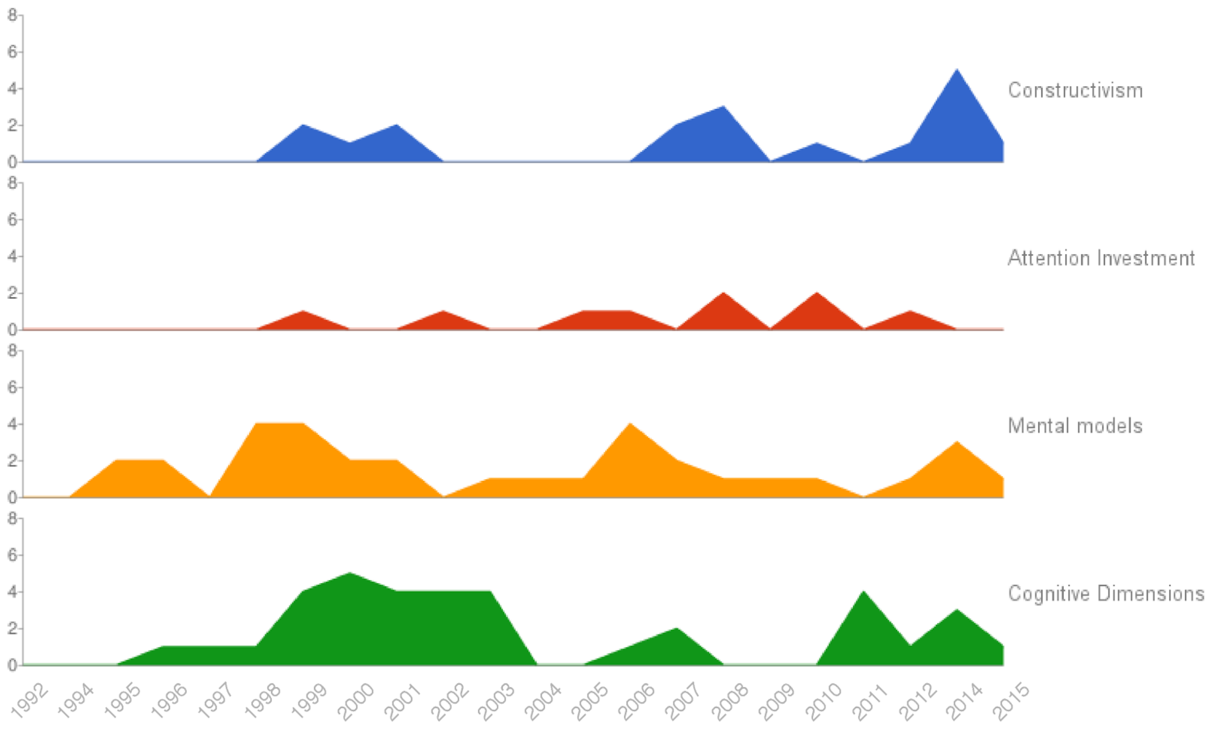


Figure 6. Theoretical frames and Analytical techniques at PPIG over time

This shows that some analytical and theoretical approaches fluctuate over time (e.g. Cognitive Dimensions, Constructivism) whereas others such as the Mental Models perspective on programming remain relatively steady.

Similarly, we can look at the empirical techniques that are used at PPIG:

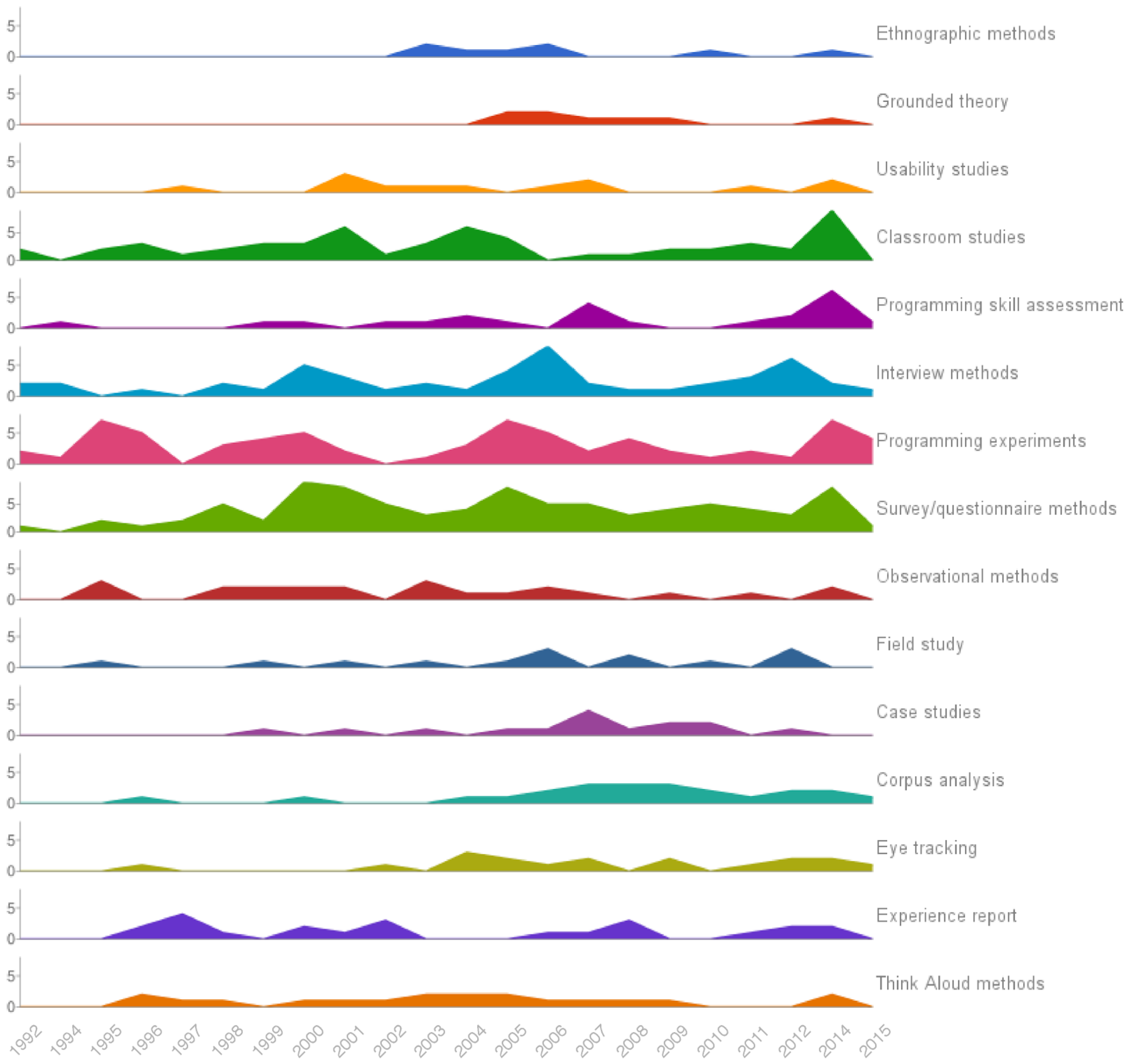


Figure 7. Empirical techniques at PPIG over time

We can see that new methods get added (e.g. Grounded theory, eye tracking or corpus analysis). These new techniques don't displace existing ones but compliment them resulting in an increased range of techniques. PLATEAU has similar characteristics, but with slightly more restricted techniques. The community has focused on programming experiments, corpus analyses, surveys, usability studies and experience reports but, so far, there are no publications using eye tracking studies, grounded theory or ethnographic techniques.

3.4 Which aspects of computation have been studied?

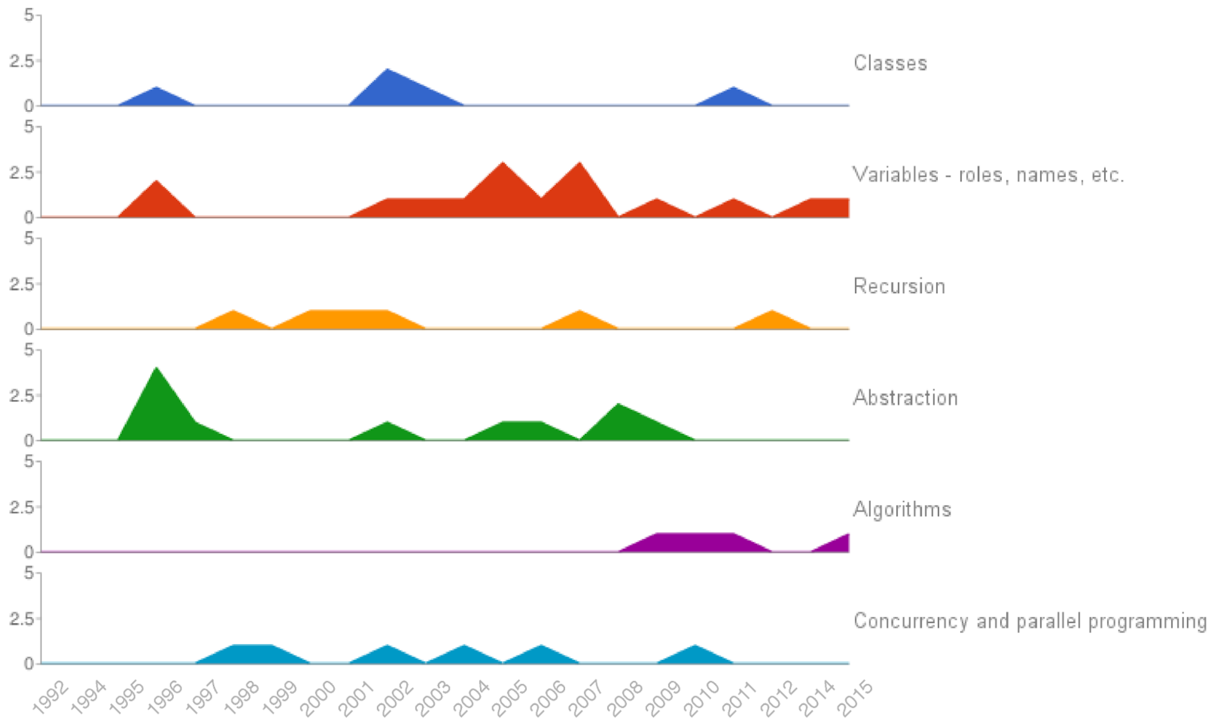


Figure 8. Aspects of computation studied at PPIG over time

Considering computational artefacts the difference between PPIG and PLATEAU is most evident. In the 6 years of PLATEAU, more papers which have been primarily concerned with concurrency and parallelism (7) have been published than in the 22 years of PPIG proceedings. However, in the same time, PPIG has been considerably more concerned with the roles and naming of variables (16 papers) compared to PLATEAU where this has not been a substantial area of study.

As (Blackwell & Morrison, 2010) point out, professional programmers tend to concentrate on semantics and control flow, whereas variable names are a primary concern of end user programmers and those relatively new to programming. As such, the continued interest in variables at PPIG may correspond to a continued interest in end-user programmers and novice programmers as discussed earlier.

3.5 Which aspects of programming have been studied?

As we see, there has been an ongoing interest in program comprehension, debugging and problem solving at PPIG, with some testing and software maintenance publications as well.

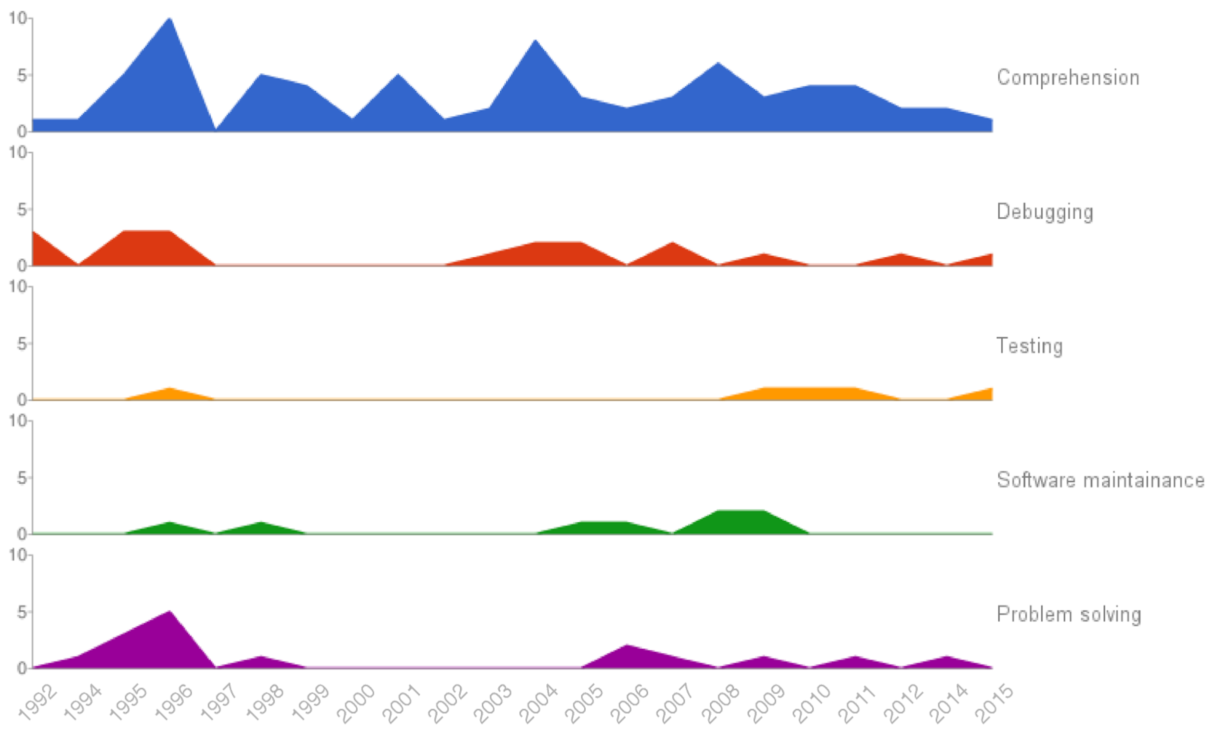


Figure 9. Aspects of programming studied at PPIG over time

3.6 Which types of technology have been studied?

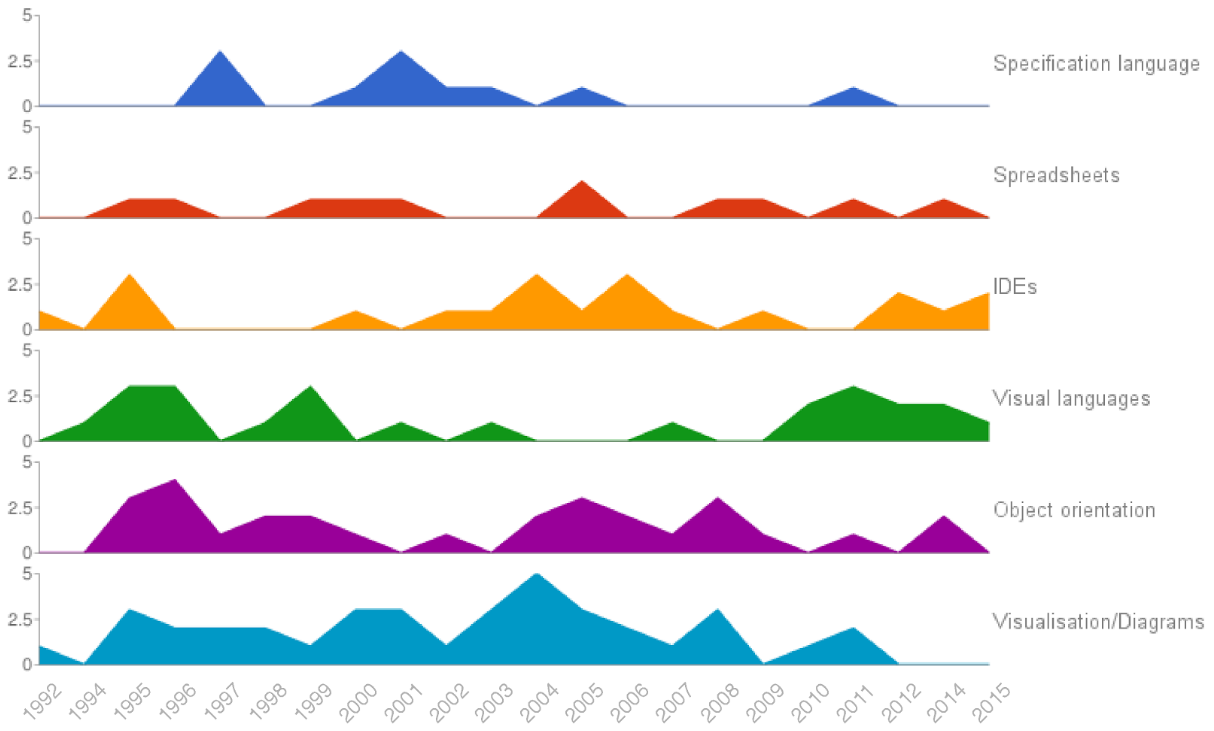


Figure 10. Programming technologies studied at PPIG over time

Here we can see the resurgence of interest in visual languages alluded to in the introduction. We also see from this and the studies of aspects above that the PPIG community is interested in programming systems, combinations of notations and the tools that people use to interact with them.

3.7 Which social interactions have been studied?

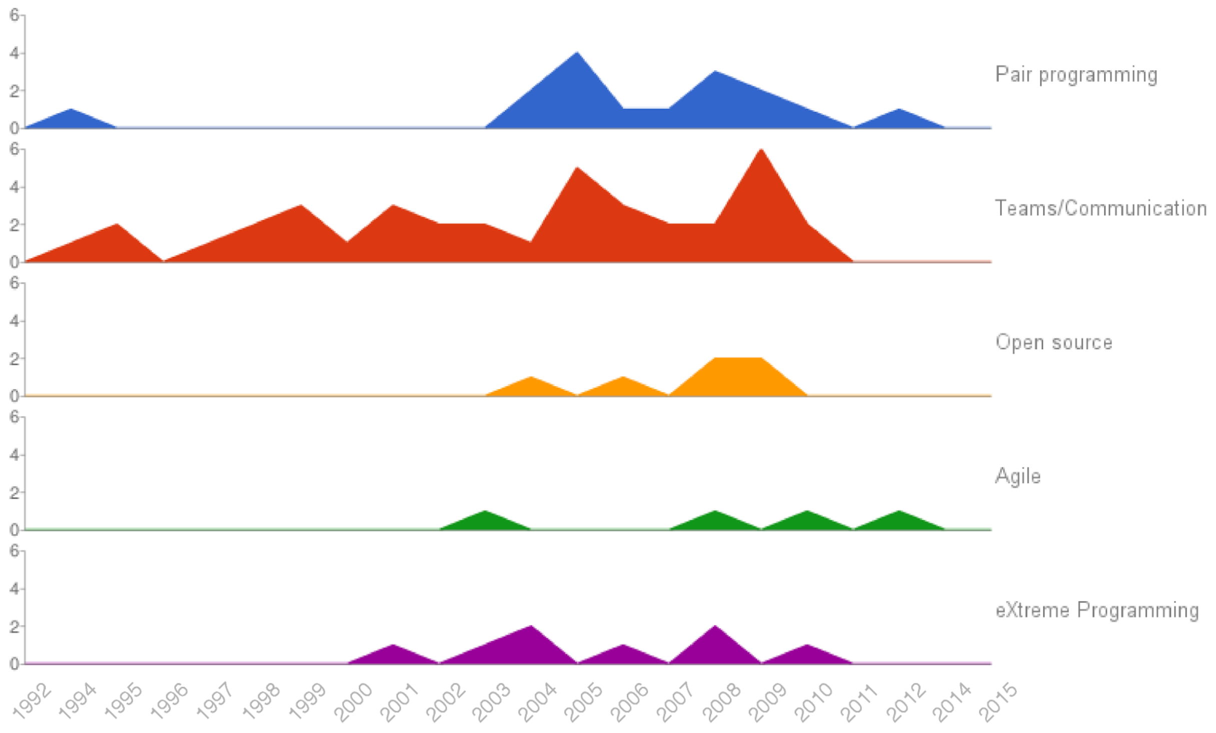


Figure 11. Social concerns studied at PPIG over time

This shows how some themes within the PPIG are relatively transitory and associated with investigating interests from the broader software engineering community.

3.8 Which factors influencing behaviour have been considered?

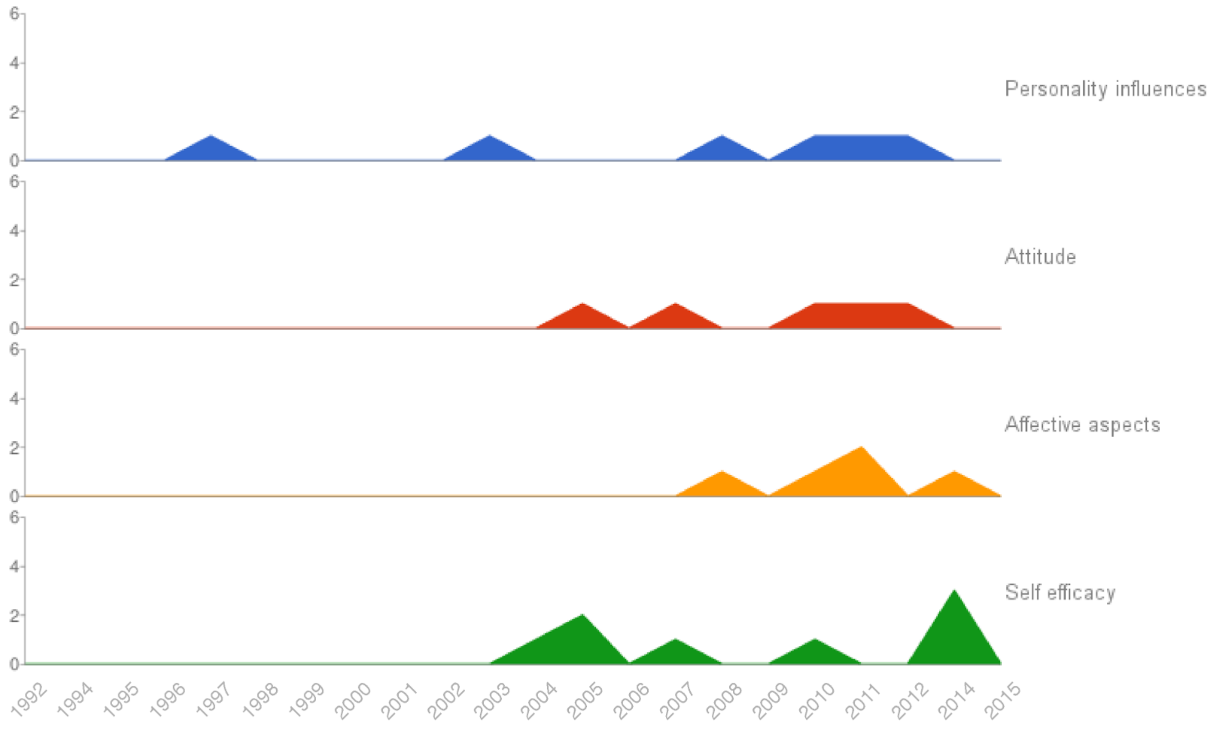


Figure 12. Behavioural concerns studied at PPIG over time

Relatively recently there has been a growing interest in non-cognitive psychological concerns at PPIG. These started with discussions of personality influences, but of late self-efficacy has become an increasing concern.

4. Discussion

4.1 PPIG and PLATEAU

Many of the differences between PPIG and PLATEAU stem from the audience being considered. PPIG has substantial End User Programming and educational interests, whilst PLATEAU is more concerned with professional and expert programmers. This leads to different artefacts being studied (e.g. IDEs vs. visual languages) and, to a certain degree, to different techniques. The most notable place where this has shaped the research agenda is PPIG's interest in variables rather than control flow constructs.

This difference in the study focus of the two cultures is institutionally useful - it allows the conferences to complement each other in the content they cover, whilst the overlap allows for effective exchanges of knowledge between the communities.

This then leads to the question of what hasn't been talked about much at PPIG. Note that we are not suggesting that these topics haven't been discussed at all, but instead suggesting that they might warrant more attention than they have received so far.

4.2 Missing groups of users

PPIG as a community has learnt a lot from studying a wide variety of users, including End User Programmers, live coders and even security workers (Biddle, 2014). However these are not the only groups of people who engage in programming-like behaviours (Blackwell, 2002).

There are other notable communities to study. For example, the communities that design 'high-integrity' systems have presumably made considerable progress along the axis of quality - what might we learn from them about the psychology of 'normal' programming?

Likewise, computer games often include substantial aspects of end user programming, such as the players mentally simulating 'what might be', creating artifacts that are repeatedly reused, and managing long range dependencies between their actions and their emergent outcomes in the game - all things that we find programmers doing. They do this with a user experience that is so compelling that people interact with it for its own sake. Studying both the large, available, gaming population and their analytical discourse (e.g. (Salen & Zimmerman, 2003)) might yield interesting learnings.

4.3 Missing psychologies and economies

As we have seen in the analysis above, the psychological techniques that PPIG has employed have primarily been cognitive in nature. Given that programming is a heavily cognitive activity, this is unsurprising. However, there are many other aspects of psychology that are of relevance. There are some good indications in this direction. For example, the application of Prospect theory (Kahneman & Tversky, 1979), which at least partially derives from mathematical psychology, resulted in Attention Investment, one of the award winning theories within the field. Similarly, Conway's Law, an adage that systems reflect the organisation that built them (Conway, 1968), can be viewed as a form of organisational psychology applied to programming.

Similarly, there may be more to learn from applying behavioural economics to understanding programmers. There is theoretical work missing on understanding why programming languages and features get adopted when they do. Moreover, there is relatively little work in understanding how notations evolve over time and how their users respond to these changes.

Finally, as (Bowker & Star, 2000) argue, abstractions have political consequences. How these political consequences affect the day to day work of programmers is poorly understood but very important in understanding the relationships between computers and society.

4.4 Theory formation

As the above results indicate, there is much more empirical work within PPIG than there is theory application. In Kuhnian terms (Kuhn, 1996), PPIG is still pre-scientific in its methodologies: there are few stable theories that are widely used to ground the experimentation and the ones we have are relatively poorly validated.

The problem with the lack of stable theory is that it becomes difficult to construct artefacts within a discipline that is continuously reinventing its own basis. This leads to two questions: what are good theoretical foundations that can be used, and how can they be used to extract understanding out of the considerable body of empirical work that is already published at PPIG. This could take the form of either attempting to empirically validate the dominant theories within PPIG (e.g. The Cognitive Dimensions of Notations). Projects like Blackbox (Brown et al., 2014) should provide data that can serve as an empirical basis, if we can understand how to analyze it effectively to ask the right questions.

4.5 Modern technologies

The languages that PPIG has extensively studied are old. Many new languages have been released and grown in adoption since Java. There are no doubt lessons that can be learnt in programming from, for example, Dart (Bracha & Bak, 2011), F#(Syme et al., 2007), Hack (Facebook, 2014), Go (Pike, 2009), Rust (Matsakis & Klock, 2014), Scala (Odersky et al., 2004), Swift (Inc, Apple, 2014). Some of these bring new debates and technical hypotheses (e.g. gradual typing can decrease premature commitment).

Similarly there are new technical trends, such as the widespread adoption of machine learning, that raise new notational and interaction challenges. Debugging machine learning systems is an open problem and one that PPIG is ideally situated to study.

These should be aided by progress in technology. As we highlight in (Church et al., 2016), very large scale computing power can now be economically applied to assist developers, if we can work out how to use it effectively. This offers a very useful opportunity for PPIG, asking the question - how can we use thousands of CPUs to assist developers?

Distributed systems not only offer a technical potential to assist developers, but they also represent a new domain for PPIG to study. Site Reliability Engineers (Beyer et al., 2016) represent a new audience with new roles and challenges.

It is not only new technologies within the domain of programming that pose interesting questions that PPIG is well placed to address. New, or re-imagined, interaction technologies such as Dialog Systems, Augmented Reality and the Internet of Things all carry challenges that could both inform and be informed by the perspectives that PPIG brings.

4.6 Fundamental challenges

There have been hints at PPIG suggesting new directions for the philosophy of computation, from viewing naming as a primary operation rather than a syntactic necessity (Church et al., 2012), to concerns about divergence in the worlds of the program and representation. However, as we suggest above, there is much more to be done at the intersection of politics, philosophy and the psychology of programming, and even more to be done by building and characterising actual programming languages designed from these insights. This is a fundamental challenge: turning the Psychology of Programming from a reflective community to an actively generative one.

5. Conclusion

In (Berlin, 1953), Berlin dichotomised thinkers and writers into two categories, those with one defining idea through which they see the world - the hedgehogs - and those that explain it through many different ideas - the foxes. Reflecting on the analysis above we must conclude that PPIG is a fox, not a hedgehog.

As the new excitement for the design of programming systems grows, we propose that PPIG should engage by expanding its horizons to study the new languages, domains, psychologies, philosophies and programmer populations - synthesizing this knowledge into new theoretical frames. In other words, PPIG is at its best as a fox and its scope for influence grows, the PPIG of the future needs to be foxier still.

6. Acknowledgements

We would like to thank the PPIG and PLATEAU communities, as well as the programming language research community at large for a fascinating set of material to study.

7. References

Berlin, I. (1953). *The Hedgehog and the Fox: An Essay on Tolstoy's View of History*. Weidenfeld & Nicolson.

Beyer, B., Jones, C., Petoff, J., & Murphy, N. R. (2016). *Site Reliability Engineering: How Google Runs Production Systems* (1 edition). O'Reilly Media, Inc, USA.

Biddle, R. (2014, June). *Beyond Usable Security*. Presented at the PPIG 2014 - 25th Annual Workshop, Brighton, UK. Retrieved from http://www.ppig.org/library/paper/beyond-usable-security

Blackwell, A. F. (2002). First steps in programming: A rationale for attention investment models. *Proceedings - IEEE 2002 Symposia on Human Centric Computing Languages and Environments, HCC 2002*, 2–10.

Blackwell, A. F., & Morrison, C. (2010). A logical mind, not a programming mind: Psychology of a professional end-user. In *PPIG 2010: Proceedings of the 22nd annual workshop of the psychology of programming interest group, September 19--22, 2010. Madrid, Spain* (pp. 175–184).

Blackwell, A., McLean, A., Noble, J., & Rohrhuber, J. (2014). Collaboration and learning through live coding (Dagstuhl Seminar 13382). *Dagstuhl Reports*, *3*(9), 130–168.

Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent Dirichlet Allocation. *Journal of Machine Learning Research: JMLR*, *3*, 993–1022.

Bowker, G. C., & Star, S. L. (2000). *Sorting Things Out: Classification and Its Consequences (Inside Technology)*. The MIT Press.

Bracha, G., & Bak, L. (2011, October). *Dart, a new programming language for structured web* *programming*. Presented at the GOTO Aarhus conference, Aarhus. Retrieved from http://gotocon.com/aarhus-2011/presentation/Opening%20Keynote:%20Dart,%20a%20new%20programming%20language%20for%20structured%20web%20programming

Braun, V., & Clarke, V. (2006). Using thematic analysis in psychology. *Qualitative Research in Psychology*, *3*, 77–101.

Brown, N. C. C., Kölling, M., McCall, D., & Utting, I. (2014). Blackbox: A Large Scale Repository of Novice Programmers' Activity. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education* (pp. 223–228). New York, NY, USA: ACM.

Church, L., Rothwell, N., & Downie, M. (2012). Sketching by Programming in the Choreographic Language Agent. *Cl.Cam.Ac.Uk*. Retrieved from http://www.cl.cam.ac.uk/~afb21/publications/PPIG-2012.pdf

Church, L., Söderberg, E., Bracha, G., & Tanimoto, S. (2016). Liveness becomes Entelechy: A scheme for L6. In *Proceedings of the Second International Conference on Live Coding (ICLC 2016)*. McMaster University, Canada.

Conway, M. E. (1968). How do Committees Invent? *Datamation*, *14*(5), 28–31.

Facebook. (2014). Hack. Retrieved June 7, 2016, from http://hacklang.org/

Greenberg, M., Fisher, K., & Walker, D. (2015). Tracking the Flow of Ideas through the Programming Languages Literature. In Thomas Ball and Rastislav Bodik and Shriram Krishnamurthi and Benjamin S. Lerner and Greg Morrisett (Ed.), *1st Summit on Advances in Programming Languages (SNAPL 2015)* (pp. 140–155). Dagstuhl, Germany: Schloss Dagstuhl--Leibniz-Zentrum fuer Informatik.

Inc, Apple. (2014). Swift. Retrieved June 7, 2016, from https://swift.org

Kahneman, D., & Tversky, A. (1979). Prospect Theory: An Analysis of Decision under Risk. *Econometrica: Journal of the Econometric Society*, *47*(2), 263–291.

Kuhn, T. S. (1996). *The Structure of Scientific Revolutions* (3rd edition). University of Chicago Press.

Matsakis, N. D., & Klock, F. S., II. (2014). The Rust Language. In *HILT '14 Proceedings of the 2014 ACM SIGAda annual conference on High integrity language technology* (Vol. 34, pp. 103–104). New York, NY, USA: ACM.

Myers, B. A., Stefik, A., Hanenberg, S., Kaijanaho, A.-J., Burnett, M., Turbak, F., & Wadler, P. (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). ACM.

Odersky, M., Altherr, P., Cremet, V., Emir, B., Maneth, S., Micheloud, S., … Zenger, M. (2004). *An overview of the Scala programming language* (No. IC/2004/64). École Polytechnique Fédérale de Lausanne. Retrieved from https://infoscience.epfl.ch/record/52656/files/ScalaOverview.pdf

Peyton Jones, S. (2015). *The dream of a lifetime: an opportunity to shape how our children learn computing*. Presented at the PPIG 2015 - 26th Annual Workshop, Bournemouth, UK. Retrieved from http://www.ppig.org/sites/default/files/2015-PPIG-26th-Peyton.pdf

Pike, R. (2009). *The Go Programming Language*. Presented at the Google's Tech Talks. Retrieved from https://redmine.bring.out.ba/attachments/3206/go_talk-20091030.pdf

Salen, K., & Zimmerman, E. (2003). *Rules of Play: Game Design Fundamentals*. MIT Press.

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.

Sunshine, J., Anslow, C., LaToza, T., Murphy-Hill, E., Sadowski, C., & Markstrum, S. (2009). Workshop on Evaluation and Usability of Programming Languages and Tools. Retrieved June 5, 2016, from https://sites.google.com/site/workshopplateau/

Syme, D., Granicz, A., & Cisternino, A. (2007). *Expert F# (Expert's Voice in .NET)* (1st Corrected ed. 2007. Corr. 6th printing 2007 edition). Apress.

Turbak, F., Bau, D., Gray, J., Kelleher, C., & Sheldon, J. (2015). Blocks and Beyond: Lessons and Directions for First Programming Environments. In *A VL/HCC 2015 workshop*. Atlanta. Retrieved from http://cs.wellesley.edu/~blocks-and-beyond/organizers.html

Victor, B. (2012). Inventing on principle.