The Next 700 Programming Language Manuals

Jeremy Singer
School of Computing Science
University of Glasgow

September 16, 2011

Research Problem: ***To optimize the production of programming language documentation, based on a study of its evolution over the past 60 years, using manual and automated techniques.***

1 Summary

The first programming manual [WWG51] was produced in 1951 by Wilkes, Wheeler and Gill. It was a carefully written scientific report, providing an account of distilled experience for the tiny global community of programmers [CK11]. Now 60 years later, we wonder whether there is still a need to write traditional programming manuals at all [SH01]? If there is, can we crowd-source the production by enabling early adopters to draft the documentation, and subsequently leaving the text to evolve online in a Wikipedia style [KCP+07]?

Documentation for programming languages is particularly important, in relation to other software documentation, e.g. that of an application or a user library. It is difficult for a programmer to engage in exploratory development without at least a basic understanding of the language-specific concepts introduced in the manual. In this project, we will investigate issues like: What does a useful manual need to tell us? How long is it? How formally should it be written? Does the documentation quality affect the popularity of the language? We know that once a programming language acquires a critical mass of programmers, the number of published textbooks increases dramatically. However, are there simple steps that a language inventor can take in order to make this popularization more likely?

In this PhD research, we propose to examine how programming language documentation has changed over time. We will investigate techniques that are effective, and those that are not. Our goal is to assess how programming language manuals could evolve in the future to be more relevant for the next generation of programmers.

2 Investigator Profile

Jeremy Singer is a lecturer at the School of Computing Science (SoCS) in the University of Glasgow (UoG). He is part-funded by the Scottish Informatics and Computer Science Alliance (SICSA), to conduct research in the *Complex Systems Engineering* strand of SICSA's research portfolio. This strand involves novel systems, engineered to meet the needs of industry and society, operating robustly in potentially hostile environments. Singer's research interests harmonize with this theme. They include:

- mathematical models for memory management [SBWC07, SBLW07, SJBL10]
- managed runtime environments for many-core platforms [SK08a, KSL10, ISK+10, SKBL11]
- static and dynamic compiler analyses [Sin05, Sin06, SBL+10]

Prior to joining UoG, Singer was a post-doctoral research associate in Steve Furber's Advanced Processor Technologies group at the School of Computer Science, University of Manchester. In this internationally renowned group, Singer's research focus was on Java runtime systems for multi-core architectures. He has published in ACM and IEEE conferences and journals in these areas.

Singer obtained his doctorate in 2006 from the Computer Laboratory, University of Cambridge. His dissertation [Sin06] concerns optimizing compilation and program analysis, particularly the static single assignment form. Some of the algorithms proposed in his thesis have since been implemented in the increasingly significant LLVM compiler[1].

Singer is a Steering Committee member of the UK Memory Management Network[2], a strong consortium of UK academics and industrial partners engaging in memory management research.

Singer is currently involved in the supervision and support of three doctoral students—two based at UoG and

[1] `http://llvm.org/`
[2] `http://www.mm-net.org.uk`

the other at the School of Informatics in the University of Edinburgh. He chaired the 2011 SICSA PhD conference[3], which was attended by 150 doctoral students from across Scotland.

3 Host Institution

The proposed research is to be carried out in the School of Computing Science (SoCS) at UoG.

Since 1957, when it became the first university in Scotland to have an electronic computer, Glasgow has built a reputation for the excellence of its Computer Science research. The 2010 QS university rankings place UoG SoCS in the top 100 Computer Science departments in the world[4]. The 2008 Research Assessment Exercise rates UoG SoCS in the UK's top ten, with 30% of research graded at 4* (world leading) and 50% at 3* (internationally excellent).

With about 30 academic staff, 70 research students and 35 research fellows, SoCS hosts many research projects funded by sources such as EPSRC, EU, Royal Society, and industrial partnerships. Grants and fellowships currently total more than £6M.

The vibrant research culture of UoG SoCS strongly encourages cross-disciplinary collaboration between the research groups, and across the various Colleges of the university.

Furthermore, UoG SoCS is a leading player in the Scottish Informatics and Computer Science Alliance (SICSA[5]), which is a research pool supported by the Scottish Funding Council. The SICSA objective is to develop and extend Scotland's position as a world leader in Informatics and Computer Science research. The ethos is for the various Scottish Computer Science centres to work co-operatively rather than competitively, by providing mutual support and sharing facilities, by working closely with industry and government and by appointing and retaining world-class staff and research students.

4 Proposed Investigation

The evolution and development of programming languages was predicted early on by Landin [Lan66]. It has been extensively studied in retrospect, e.g. [Ber07, Mau02, AV97, PZ95]. As a profession, we feel the need to explore, characterize and reflect on the overall trends in programming language evolution. The significant majority of academic effort concentrates on the languages themselves, i.e. their syntax and semantics. To a lesser extent, we study the compiler technology that supports these languages.

However in this investigation, we propose to look at parts of the **wider eco-system of a programming language**, particularly considering *documentation*. For our purposes in this proposal, documentation refers to descriptions of a programming language and comprises materials like:

1. published textbooks and manuals
2. informal reports and lecture notes
3. formal standardization documents
4. archives of support messages generated by the community
5. other relevant textual material.

We intend to consider the perspective of a developer *using* the language, rather than a designer *creating* the language. Two extreme examples of documentation philosophy are ML and Perl. ML has a formal mathematical definition document [MTHM97], which is the official specification of the core language, and is regarded as unalterable. On the other hand, Perl takes a more pragmatic approach. The website `http://perldoc.org` is the online repository of the official language standard documentation, and is constantly changing. Although O'Reilly reference books e.g. [WCO00] by Larry Wall (the inventor of Perl), are treated as canonical, they are likely to be out-of-date in relation to the dynamic website.

Some language communities (e.g. Algol 68) begin with a formal definition for experts [VWMPK68, vWMP+76] and later realize the need for an informal textbook for ordinary users [LvdM77]. Other languages (e.g. Java) are born with an informal definition [AGH00] which is followed by theoretical frameworks (e.g. [IPW01, DE99]) to fix up the problems that are eventually discovered.

For many languages the *only* extant definition is the textbook, such as BCPL [RWS80] and pre-standardized C [KR78]. The *de facto* definition is the compiler, but in the early days, the compiler and textbook author were generally the same person(s).

The observations that inform our research are based on the fact that programming languages have evolved over the last 60 years [Kay84]. We are interested in corresponding changes in the manner in which languages were documented (and therefore adopted) over the same time period.

Many factors may contribute to the evolution of documentation and developer behaviour. Elzen and Mackenzie have studied the sociological impact of a broadening base of computer users, from the perspective of supercomputer architecture [EM94]. We hope to perform a similar study from the perspective of programming language design, deployment and usage. The International Standards Organization has developed

[3] `http://www.sicsaconf.org/`

[4] `http://www.topuniversities.com/university-rankings/world-university-rankings/2011/subject-rankings/engineering/computer-science`

[5] `http://www.sicsa.ac.uk`

prescriptive guidelines for the production of programming language documentation [ISO97]. The increasingly widespread availability of compute power enables a more rapid edit/compile/debug cycle. This leads to the modern ideas of exploratory (or agile) programming [B+01]. Further, integrated development environments offer online documentation, live syntax correction, etc. Internet connectivity enables social networking for programming language support, via facilities like newsgroups, mailing lists, wikis and chat forums. Web archives host online code repositories for most major languages, to allow developers to cut and paste, or at least use as a source of inspiration when creating new programs. Examples include `http://sourceforge.net` and `http://github.com` for open-source projects, or even `http://www.pastebin.org` for smaller snippets of code. Code search engines such as `http://code.google.com` are growing in popularity too. The combination of these technologies gives rise to the concept of *postmodern programming* [NB02, NB04] which posits that instead of creating a new program to solve a particular problem, programmers should simply search for an existing program and modify it as necessary.

So, the particular research questions to tackle in this investigation are:

1. How have programming language documentation features (in terms of content, style and media) changed over time?
2. How does the provision of programming language documentation correlate with the success of those languages, in terms of linguistic adoption and usage by developers?

5 Research Plan

The research will take place over three years, with deliverables **DY.n** noted at the end of each year.

5.1 Year 1

We will commence with case studies of some exemplar languages. We intend to select at least one representative language from each decade. Our initial set of candidate languages is below. These particular languages are selected because of the links that the principal investigator and collaborators have with the corresponding language designers, which should give us excellent access to the full range of documentation material.

- Autocode (1950s/60s)
- BCPL (1960/70s)
- C++ (1980/90s)
- Haskell (1990s)
- F# (2000s)

The initial task involves *collecting and categorizing* the available documents for each language and assessing their relevance. We will experiment with some automated metrics and some manual judgement of documentation. The main concern is the requirement to to apply *common measures to documentation* from different languages and different periods, for a fair comparison between them.

We intend to apply *discourse analysis* [Gee11] to the documents, in order to see how technical phraseology has changed over time. We will also use other standard text-based statistical measurements, e.g. [Jon08]. This will be in close collaboration with the English Language department at UoG, which has significant expertise in this area.

We aim to construct a *taxonomy* of documentation options. We will investigate how documentation for different languages fits into various parts of the taxonomy.

Deliverables

D1.1 Corpus of proglangs documentation e-text for analysis

D1.2 Study of changes in text-based metrics over time, on the corpus

D1.3 Taxonomy of documentation options (book, wiki, forum, ...)

5.2 Year 2

In this middle stage of the project, we hope to conduct interviews with people, and read through further documentation material. *Language designers* will provide insight regarding how the manuals and documentation were originally produced for a programming language. (Note that the pioneers of each of the languages mentioned above are fairly accessible for interviews, and have given previous statements about their language history, strategy and philosophy.) *Language developers* who produced programs in these languages will provide insight regarding how the manuals and documentation were used for education and day-to-day programming.

Although these interviews might result in highly unstructured data, we hope to identify cross-cutting themes and underlying trends. The interview might be carefully structured in the form of a questionnaire, to enable more quantitative analysis of the responses. There may be some role for *Grounded Theory* here [GS67]. Specifically, we hope to discover trends that show how various policies about documentation affect the adoption of a programming language.

We will examine the differences between various languages. For instance, the 'documentation' for some lan-

guages merely consists of a collection of source code examples (e.g. Google go [Goo11]). Also, there are some cases where the documentation precedes the language implementation (e.g. PL/I).

To mitigate the risk that we might not be able to arrange many interviews, we have a fallback option to analyse existing public interviews and statements from language designers e.g. [Bia09, Jon03, BG96, Wex93]. However we accept that most of this material is not particularly focused on documentation issues.

Deliverables

D2.1 Publication of interview transcriptions

D2.2 Study of trends in communication between language designers and language users, through the various documentation media

5.3 Year 3

In this final year of the project, we will gather together all the material. We aim to synthesize a systematic categorization of practice regarding programming language documentation. We will use the source material to answer the research questions posed above. Finally, we hope to provide recommendations for best practice for developers of new languages.

Deliverables

D3.1 Assessment of the effectiveness of various documentation approaches

D3.2 Guidelines for future documentation practice

D3.3 PhD dissertation

6 Research Methods

The proposed research is not experimental. Instead, what we intend to do largely consists of *analysis* and *interpretation* of historical material. While these are not standard research practices in Computer Science, we feel that they would be worthwhile for the particular domain area of programming language documentation. We will enlist the help of suitably experienced collaborators (see Section 7) to ensure we have the appropriate mix of domain-specific knowledge and research experience.

Our work will involve the collection of a range of materials relating to specific programming languages. We intend to *archive and curate* this material in such a way that it might be useful to other researchers in the future. This will include the raw artifacts such as recordings or transcriptions of interviews where possible.

7 Collaboration

We expect this research to be highly collaborative, due to its interdisciplinary nature. Below we list four partners with whom we already have strong links, and who are explicitly interested in working on this specific project. Throughout the work, we expect to strengthen these partnerships and forge new links. This is highly likely to happen within existing research networks such SICSA, the Analysis, Slicing and Transformation Research Network (ASTReNet[6]), and the Psychology of Programming Interest Group (PPIG[7]).

Paul Cockshott is a Reader at UoG SoCS. He will act as a project advisor, and a secondary supervisor for the PhD student. Cockshott's background is in programming language design and implementation. He is the maintainer of the Vector Pascal compiler infrastructure [Coc02].

Martin Richards is a retired Senior Lecturer at the Computer Laboratory, University of Cambridge. He was Singer's PhD supervisor 2001-2005. Richards is the inventor of the seminal BCPL language, for which he recently received an IEEE Computer Pioneer Award[8].

Derek M. Jones runs *Knowledge Software Ltd* which is a programming language consultancy firm. Jones has authored a comprehensive commentary on the C standard [Jon02]. He is a visiting professor at Kingston University and a member of IST/5 (the UK programming language standards body). His particular research focus is software development from a cognitive psychology point of view.

Jean Anderson is a researcher in the UoG English Language department. Singer and Anderson have collaborated on computer-aided analysis of Westminster parliamentary discourse. We aim to apply similar automated analysis tools to programming language manuals.

8 Expected Outcomes

The main intellectual outcome of the project will be an understanding of how programming language documentation practice (and hence adoption, indirectly) has changed over time. More concretely, we aim to provide best practice guidance for dissemination of information about new languages. We will provide a template portfolio of information required by developers when working with a new language.

We will report these research findings in high-profile CS outlets such as ACM TOPLAS and IEEE Annals of the History of Computing, or conferences such as PLDI, OOPSLA, and ICSE. We also expect to produce material

[6] http://www.dcs.kcl.ac.uk/staff/zheng/astrenet/

[7] http://www.ppig.org

[8] http://www.computer.org/portal/web/awards/richards-pioneer

that will be of interest to the more popular scientific press, such as CACM and New Scientist.

This research project is timely. It is only as we use tools of historical interpretation to look back and review past behaviour that we can take advantage of our perspective in the present. Santayana's remark is apt: 'Those who cannot remember the past are condemned to repeat it.' Our research proposal offers: (1) a unique and timely opportunity to collect and centralise first-hand evidence from those pioneers involved in creating the early high-level languages; (2) a review of the efficacy of various programming language documentation practices; and (3) recommendations for optimal documentation strategies for future languages.

As the software development community shifts to the many-core paradigm, programming languages are likely to become more complex. Developers will rely increasingly on their interaction with effective, accessible, and user-friendly documentation. This research aims to facilitate such documentation for programming languages of the future.

References

[AGH00] K. Arnold, J. Gosling, and D. Holmes. *The Java programming language*. Addison-Wesley, 2000.

[AV97] Doris Appleby and Julius J. VandeKopple. *Programming Languages Paradigm and Practice*. McGraw-Hill, 2nd edition, 1997.

[B+01] Kent Beck et al. Manifesto for agile software development, 2001. http://agilemanifesto.org/.

[Ber07] Thomas J. (Tim) Bergin. A history of the history of programming languages. *Communications of the ACM*, 50:69–74, May 2007. http://doi.acm.org/10.1145/1230819.1230841.

[BG96] Thomas J. Bergin, Jr. and Richard G. Gibson, Jr., editors. *History of programming languages—II*. ACM, 1996.

[Bia09] Federico Biancuzzi. *Masterminds of Programming*. O'Reilly, 2009.

[CK11] Martin Campbell-Kelly. In praise of Wilkes, Wheeler, and Gill. *Communications of the ACM*, 54(9):25–27, 2011. http://dx.doi.org/10.1145/1995376.1995386.

[Coc02] Paul Cockshott. Vector pascal reference manual. *ACM SIGPLAN Notices*, 37:59–81, June 2002. http://doi.acm.org/10.1145/571727.571737.

[DE99] Sophia Drossopoulou and Susan Eisenbach. Describing the semantics of Java and proving type soundness. In Jim Alves-Foss, editor, *Formal Syntax and Semantics of Java*, volume 1523 of *Lecture Notes in Computer Science*, pages 41–80. Springer Berlin / Heidelberg, 1999. http://dx.doi.org/10.1007/3-540-48737-9_2.

[EM94] B. Elzen and D. Mackenzie. The social limits of speed: development and use of supercomputers. *Annals of the History of Computing, IEEE*, 16(1):46–61, 1994. http://dx.doi.org/10.1109/85.251854.

[Gee11] James Paul Gee. *An Introduction to Discourse Analysis: Theory and Method*. Routledge, 3rd edition, 2011.

[Goo11] Google. Effective go, 2011. http://golang.org/doc/effective_go.html.

[GS67] Barney G. Glaser and Anselm L. Strauss. *The discovery of grounded theory: strategies for qualitative research*. Aldine, 1967.

[IPW01] Atsushi Igarashi, Benjamin C. Pierce, and Philip Wadler. Featherweight Java: a minimal core calculus for Java and gj. *ACM Transactions on Programming Languages and Systems*, 23:396–450, May 2001. http://doi.acm.org/10.1145/503502.503505.

[ISK+10] Nikolas Ioannou, Jeremy Singer, Salman Khan, Polychronis Xekalakis, Paraskevas Yiapanis, Adam Pocock, Gavin Brown, Mikel Lujan, Ian Watson, and Marcelo Cintra. Toward a more accurate understanding of the limits of the tls execution paradigm. In *Proceedings of the IEEE International Symposium on Workload Characterization*, 2010. http://dx.doi.org/10.1109/IISWC.2010.5649169.

[ISO97] ISO. Implementation of ISO/IEC TR 10176:1997 Information technology—guidelines for the preparation of programming language standards. Technical report, International Standards Organization, 1997.

[Jon02] Derek M. Jones. *The New C Standard A Cultural and Economic Commentary*. 2002. http://www.knosof.co.uk/cbook/cbook.htm.

[Jon03] Simon Peyton Jones. Wearing the hair shirt: a retrospective on Haskell. invited talk at POPL, 2003. http://research.microsoft.com/en-us/um/people/simonpj/papers/haskell-retrospective/HaskellRetrospective.pdf.

[Jon08] Derek M. Jones. Forms of language specification, Feb 2008. http://www.knosof.co.uk/vulnerabilities/langconform.pdf.

[Kay84] Alan Kay. Computer software. *Scientific American*, 251(3):41–47, 1984.

[KCP+07] Aniket Kittur, Ed H. Chi, Bryan A. Pendleton, Bongwon Suh, and Todd Mytkowicz. Power of the few vs. wisdom of the crowd: Wikipedia and the rise of the bourgeoisie, 2007. http://www.viktoria.se/altchi/index.php?action=showsubmission&id=41.

[KR78] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language*. Prentice Hall, 1978.

[KSL10] George Kovoor, Jeremy Singer, and Mikel Lujan. Building a Java map-reduce framework for multi-core architectures. In *Proceedings of the Third Workshop on Programmability Issues for Multi-Core Computers*, 2010. http://www.cs.man.ac.uk/~lujanmx/research/docs/kovoor_multiprog2010.pdf.

[Lan66] P. J. Landin. The next 700 programming languages. *Communications of the ACM*, 9:157–166, March 1966. http://doi.acm.org/10.1145/365230.365257.

[LvdM77] C. H. Lindsey and S. G. van der Meulen. *Informal introduction to ALGOL 68*. North-Holland, 1977.

[Mau02] W. Douglas Maurer. The comparative programming languages course: a new chain of development. *SIGCSE Bulletin*, 34:336–340, Feb 2002. http://doi.acm.org/10.1145/563517.563472.

[MTHM97] Robin Milner, Mads Tofte, Robert Harper, and David MacQueen. *The Definition of Standard ML, Revised Edition*. MIT Press, 1997.

[NB02] James Noble and Robert Biddle. Notes on postmodern programming. Technical Report CS-TR-02-9, Computer Science, Victoria University of Wellington, Mar 2002. http://www.mcs.vuw.ac.nz/comp/Publications/archive/CS-TR-02/CS-TR-02-9.pdf.

[NB04] James Noble and Robert Biddle. Notes on notes on postmodern programming: radio edit. In *Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, OOPSLA '04, pages 112–115, 2004. http://doi.acm.org/10.1145/1028664.1028710.

[PYS+10] Adam Pocock, Paraskevas Yiapanis, Jeremy Singer, Mikel Lujan, and Gavin Brown. Online non-stationary boosting. In *Proceedings of the International Workshop on Multiple Classifier Systems*, volume 5997 of *Lecture Notes in Computer Science*. Springer, 2010. http://dx.doi.org/10.1007/978-3-642-12127-2_21.

[PZ95] Terrence W. Pratt and Marvin V. Zelkowitz. *Programming Languages Design and Implementation*. Prentice Hall, 3rd edition, 1995.

[RWS80] Martin Richards and Colin Whitby-Strevens. *BCPL, the language and its compiler*. Cambridge University Press, 1980.

[SB06] Jeremy Singer and Gavin Brown. Return value prediction meets information theory. *Electronic Notes in Theoretical Computer Science*, 164(3):137–151, Oct 2006. http://dx.doi.org/10.1016/j.entcs.2006.07.016.

[SBL+10] Jeremy Singer, Gavin Brown, Mikel Luján, Adam Pocock, and Paraskevas Yiapanis. Fundamental nano-patterns to characterize and classify Java methods. *Electronic Notes in Theoretical Computer Science*, 253:191–204, 2010. http://dx.doi.org/10.1016/j.entcs.2010.08.042.

[SBLW07] Jeremy Singer, Gavin Brown, Mikel Luján, and Ian Watson. Towards intelligent analysis techniques for object pretenuring. In *Proceedings of the International Conference on Principles and Practice of Programming in Java*, pages 203–208, Sep 2007. http://doi.acm.org/10.1145/1294325.1294353.

[SBWC07] Jeremy Singer, Gavin Brown, Ian Watson, and John Cavazos. Intelligent selection of application-specific garbage collectors. In *Proceedings of the 6th International Symposium on Memory Management*, pages 91–102, Oct 2007. http://doi.acm.org/10.1145/1296907.1296920.

[SH01] Abigail J. Sellen and Richard H. R. Harper. *The Myth of the Paperless Office*. MIT Press, 2001.

[Sin05] Jeremy Singer. Static single information from a functional perspective. *Trends in Functional Programming*, 4:63–78, Jan 2005. http://homepages.inf.ed.ac.uk/stg/workshops/TFP/book/Singer/singer/ssifunc.pdf.

[Sin06] Jeremy Singer. Static program analysis based on virtual register renaming. Technical Report UCAM-CL-TR-660, University of Cambridge Computer Laboratory, Feb 2006. http://www.cl.cam.ac.uk/TechReports/UCAM-CL-TR-660.pdf.

[SJBL10] Jeremy Singer, Richard E. Jones, Gavin Brown, and Mikel Luján. The economics of garbage collection. In *Proceedings of the International Symposium on Memory Management*, pages 103–112, 2010. http://doi.acm.org/10.1145/1837855.1806669.

[SK08a] Jeremy Singer and Chris Kirkham. Dynamic analysis of Java program concepts for visualization and profiling. *Science of Computer Programming*, 70(2–3):111–126, Feb 2008. http://dx.doi.org/10.1016/j.scico.2007.07.006.

[SK08b] Jeremy Singer and Chris Kirkham. Exploiting the correspondence between micro patterns and class names. In *Proceedings of the Eighth IEEE International Working Conference on Source Code Analysis and Manipulation*, pages 67–76, 2008. http://dx.doi.org/10.1109/SCAM.2008.23.

[SKBL11] Jeremy Singer, George Kovoor, Gavin Brown, and Mikel Luján. Garbage collection auto-tuning for Java mapreduce on multi-cores. In *Proceedings of the International Symposium on Memory Management*, pages 109–118, 2011. http://doi.acm.org/10.1145/1993478.1993495.

[vWMP+76] A. van Wijngaarden, B.J. Mailloux, J.E.L. Peck, C.H.A. Koster, M. Sintzoff, CH Lindsey, L. Meertens, and RG Fisker. *Revised report on the algorithmic language ALGOL 68*. Springer-Verlag, 1976.

[VWMPK68] A. Van Wijngaarden, B. J. Mailloux, J. Peck, and C. H. A. Koster. Draft report on the algorithmic language algol 68. *ALGOL Bulletin*, pages 1–84, March 1968. http://dx.doi.org/10.1145/1064072.1064073.

[WCO00] Larry Wall, Tom Christiansen, and Jon Orwant. *Programming Perl*. O'Reilly, 3rd edition, 2000.

[Wex93] *SIGPLAN Notices*, 28(3), 1993. Record of the History of Programming Languages II conference.

[WWG51] M. V. Wilkes, D. J. Wheeler, and S. Gill. *The preparation of programs for an electronic digital computer*. Addison-Wesley, 1951.