Appendix A: HOPL IV History, Call for Papers, Reviewing, and Shepherding

RICHARD P. GABRIEL (poet, writer, computer scientist), California

The past is a foreign country: they do things differently there.
—Leslie Hartley, *The Go-Between* (1953)

Every 12–15 years ACM and SIGPLAN organize a conference and publication venue to gather historical papers about proven important programming languages, programming language families, language features, design themes, and other strong influences on the direction of programming language design, implementation, and usage. HOPL papers are not like ordinary research papers—they are oral histories, historical investigations, origin stories, and explorations. They have more in common with essays than with research papers. Therefore, submissions are not reviewed using the same criteria as research papers; submissions are not treated as final products; and reviewing is not on a short cycle in a hands-off manner.

1 HOPL IS DIFFERENT

Computer science research has an observed lifecycle: germs of ideas, ordinary research[1] (theoretical, practical, or empirical), reporting results, and reflection. Most SIGPLAN conferences and journals address ordinary research by providing venues to report the results. Some SIGPLAN conferences (Onward!, for example) address the germs of ideas, and some (Onward! Essays) address reflection. However, these outlier conferences are not generally considered the prime SIGPLAN conferences. The History of Programming Languages series might be the exception.

The current History of Programming Languages conference as reported in PACMPL breaks many traditional research publication rules. In addition to issuing the usual Call for Papers to as broad an audience as possible, HOPL IV also sent out "letters of encouragement"—because for HOPL, a call for papers alone is expected to produce almost no submissions. Instead of casting a wide net and then choosing great work and great submissions, HOPL IV, like its predecessors, has tried to identify historically important topics and sought authors most likely to write the best, most thorough paper.

To put this in perspective, consider the programming language Fortran. A compendium of histories of important programming languages must include a paper about Fortran. And who would have been better qualified to write about how Fortran came about than its creator John Backus? It would have been silly to put out a call for papers about the history of programming languages and *hope* that John Backus would decide to submit a paper about Fortran. Therefore, he was invited to do so for HOPL I.

Instead of imposing a low acceptance rate to ensure quality, HOPL aims to achieve 100% acceptance by supplying sufficient editorial and copyediting assistance to ensure a very high level of quality. Beginning with HOPL IV, these helpers have been called "shepherds," the idea being that a shepherd helps the authors of a paper prepare that paper to its best potential.

Instead of relying on the fact that "there is always next year" or "there is always another similar conference coming up," HOPL must assume that if the needed paper is not published for

[1] Ordinary in the sense of what is considered current practice.

this conference or journal, it will never be published—this because there are no other history of programming language conferences, and the next edition of HOPL is more than a decade distant.

Instead of imposing strict genre, style, and structural requirements to ensure that readers will be able easily to grasp the structural aspects of a paper, HOPL imposes no such requirements and has no limitation on page counts. Some HOPL papers are book-length. This is because the nature of each subject and the capabilities of each author are unique, and the contributions are to be judged by historical standards, not conventional scientific standards. Nevertheless, HOPL has always maintained a high standard of scholarship.

Instead of assuming papers are written for an audience reading in the next year or two, HOPL assumes that the primary readers will be the language designers of the next 20 years, and historians or scholars 20 to 100 years in the future.

Instead of assuming that authors will know how to write a history paper, HOPL provides an extensive set of guidelines and questions for authors before drafting begins to help those authors produce a high-quality, scholarly paper.

Instead of assuming that the members of the Review Committee will know already the standards for review and how to review history-paper submissions, HOPL assumes they do not and provides instructions and training to Reviewers.

Every HOPL conference has had a professional historian as a consultant and/or editor.

Finally, and most surprisingly, instead of valuing novelty as the most important criterion for acceptance, (fresh) novelty[2] in HOPL papers is forbidden.

2 THE HISTORY OF THE HISTORY OF PROGRAMMING LANGUAGES

There have been three prior History of Programming Languages conferences: HOPL I in 1978, HOPL II in 1993, and HOPL III in 2007. The differences from ordinary programming-language conferences were established in the first HOPL and have continued with some revisions since.

2.1 The History of Programming Languages I

Jean E. Sammet (1928–2017) had a fondness for programming languages, history, and scholarship. She developed FORMAC and was one of the designers of COBOL. She also served ACM in various capacities, including Chair of SICSAM (later SIGSAM), Chair of SIGPLAN, and Vice President and then President of ACM. In 1969 she wrote an influential book on programming languages, *Programming Languages: History and Fundamentals* [Sammet 1969]. In the 1970s she decided to hold a conference in which a key person in the creation of a programming language talked about how the language was created. Sammet sought and received the support of SIGPLAN and also served as General and Program Committee Chair; John A. N. Lee served as Administrative Chairman and designed the conference logo (Figure 1).

The first ACM SIGPLAN History of Programming Languages Conference was held June 1–3, 1978, in Los Angeles, California. The keynote speaker was Grace Murray Hopper, who identified herself as originally not a programmer but a "coder" and discussed what it was like to code programs in the early 1940s [Hopper 1981].

Preliminary versions of all the papers were published in a special issue of *SIGPLAN Notices* [Wexelblat 1978]. The final conference proceedings, including not only all the papers but also transcripts of audio recordings of the conference sessions, were captured in the monograph *History of Programming Languages*, edited by Richard L. Wexelblat [Wexelblat 1981]. The consulting historian was Henry Tropp.

[2]Novelty in the sense of new technical results in the subject area.

Fig. 1. The HOPL I Logo. (Courtesy ACM.)

The Program Committee set specific and general criteria for inclusion of a language. The specific requirements were that the language must have been created, and in use, by 1967; must remain in use in 1977; and must have had considerable influence on the field of computing. All papers would be by invitation.

The general criteria were usage, influence on language design, overall impact on the computing environment, novelty (of the language and its ideas), and uniqueness. The Program Committee decided on 13 languages (14 papers because of an American / European ALGOL divide) (Table 1).

The Program Committee wanted thoroughly researched and documented papers, and they believed that non-historians needed guidance to write good history, so they drafted 82 specific questions to guide authors and assist in identifying important facts and ideas [HOPL 1981]. The headings for those guidelines and questions were as follows:

(1) Background
  (a) Basic Facts about Project Organization and People
  (b) Costs and Schedules
  (c) Basic Facts about Documentation
  (d) Language / Systems Known at the Time
  (e) Intended Purposes and Users
  (f) Source and Motivation
(2) Rationale of Content of the Language
  (a) Environment Factors
  (b) Functions to be Programmed
  (c) Language Design Principles
  (d) Language Definition
  (e) Concepts about Other Languages
  (f) Influence of Non-technical Factors
(3) A Posteriori Evaluation
  (a) Meeting of Objectives
  (b) Contributions of Languages
  (c) Mistakes or Desired Changes
  (d) Problems
(4) Implications for Current and Future Languages
  (a) Direct Influence
  (b) Indirect Influence

Table 1. HOPL I Authors and Papers

Author	Paper
John Backus	The History of FORTRAN I, II, and III
Alan J. Perlis	The American Side of the Development of ALGOL
Peter Naur	The European Side of the Last Phase of the Development of ALGOL
John McCarthy	History of LISP
Jean E. Sammet	The Early History of COBOL
Douglas T. Ross	Origins of the APT Language for Automatically Programmed Tools
Jules I. Schwartz	The Development of JOVIAL
Geoffrey Gordon	The Development of General Purpose Simulation System (GPSS)
Kristen Nygaard Ole-Johan Dahl	The Development of the SIMULA Language
Charles L. Baker	JOSS: JOHNNIAC Open-Shop System
Thomas E. Kurtz	BASIC
George Radin	The Early History and Characteristics of PL/I
Ralph E. Griswold	A History of the SNOBOL Programming Language
Kenneth E. Iverson Adin D. Falkoff	The Evolution of APL

The first HOPL used discussants for some papers. A discussant is someone who puts the paper in perspective, shines a different light on the topic, or remarks on things not written in the paper but which are relevant. For HOPL I the remarks of discussants were captured in the monograph.

The first HOPL had what we now call "shepherds"; they were called "Language Coordinators." Each paper was assigned two Language Coordinators, of whom at least one was also a member of the Program Committee. The monograph describes the list of them as follows:

> The Language Coordinators who worked to help get the papers done, done well, and done on time were …

Each paper had its corresponding Language Coordinators named.

2.2 The History of Programming Languages II

In 1990 SIGPLAN gave the go-ahead for the Second History of Programming Languages Conference (HOPL II). John A. N. Lee was Conference Chair and Jean Sammet was Program Committee Chair. HOPL was revised to include contributed as well as invited papers, and the scope was expanded to include the *evolution* of languages, the history of language *features* and *concepts*, and papers on *classes of languages* such as application-oriented and paradigm-oriented languages.

Preliminary ideas must have been documented by 1982 and the language must have been in use or being taught by 1985. As with HOPL I, the committee provided authors a set of Content Guidelines [HOPL II 1996] and a set of "Questions on the Early History of a Language" [Hudson 1996a] (both based on material from HOPL I) plus three new sets of questions [Hudson 1996b,c,d] to deal with the three new types of papers. A call for papers was issued in December of 1990 with notification to potential authors that their papers would go through two rounds of refereeing with a major rewrite probably needed between them.

The conference's Consulting Historian, Michael S. Mahoney, wrote an essay "What Makes History" [Mahoney 1996b], which was sent to authors along with his reviews of their submissions. In the essay Mahoney exhorted HOPL authors to go beyond "events and people in chronological

Table 2. HOPL II Authors and Papers

Author	Paper
Charles H. Lindsey	A History of ALGOL 68
Niklaus Wirth	Recollections About the Development of Pascal
Per Brinch Hansen	Monitors and Concurrent Pascal: A Personal History
William A. Whitaker	Ada–The Project: The DoD High Order Language Working Group
Guy L. Steele Jr. Richard P. Gabriel	The Evolution of Lisp
Alain Colmerauer Philippe Roussel	The Birth of Prolog
Richard E. Nance	A History of Discrete Event Simulation Languages
Jean E. Sammet	The Beginning and Development of FORMAC
Barbara Liskov	A History of CLU
Alan C. Kay	The Early History of Smalltalk
Ralph E. Griswold Madge T. Griswold	History of the Icon Language
Donald R. Colburn Charles H. Moore Elizabeth E. Rather	The Evolution of Forth
Dennis Ritchie	The Development of the C Programming Language
Bjarne Stroustrup	A History of C++

order" to examine what people knew at that time, the (natural) language that they used, and what the problems seemed like to the people as those people were when they came upon those problems. In his conference remarks "Making History" [Mahoney 1996a], he pointed out that "being right is not an explanation"—at the time the work was done, no one knew what would become "the right thing."

HOPL II was held April 20–23, 1993, in Cambridge, Massachusetts. Frederick P. Brooks Jr. gave a keynote address on "Language Design as Design" [Brooks 1996]. Preliminary versions of all the papers, along with a complementary set of "brief language summaries"of about two pages each, were published in a special issue of *SIGPLAN Notices* [Wexelblat 1993]. The complete proceedings, including not only the papers but also transcripts of audio recordings of the conference sessions (but omitting the "brief language summaries"), were published in 1996 as a book, *History of Programming Languages*, edited by Thomas J. Bergin and Richard G. Gibson [Bergin and Gibson 1996a]. (Some of the material from the conference was omitted from the paper book for lack of space, but is available in the version in the ACM Digital Library, and a transcript of remarks made at the conference banquet was published separately [Bergin and Gibson 1996b].) The authors and papers are listed in Table 2.

HOPL II had an associated graphic that could be called a "logo" (Figure 2, on the left). It consists of a quill pen that appears to have just written some code in a "standard" programming language. This graphic appeared on the front cover (large), spine (small), and back cover (small) of the Bergin/Gibson book. The final program contained a larger graphic that included the quill pen and code sample, and one could refer to it as a "badge" (Figure 2, on the right).

HOPL II also assigned discussants, this time to each paper. Their remarks are in the proceedings.

Fig. 2. The HOPL II Logo (left) and the HOPL II badge (right). (Courtesy ACM.)

The second HOPL also had what we now call "shepherds" but called them "Language Experts." Here is how they were acknowledged in the proceedings:

> To assist the authors, each PRC [Program Review Committee] chair assigned a technical expert in the subject to help each author with the paper by reviewing the various drafts. In some sense, these experts are the unsung heroes of the conference, and so it is appropriate to list and thank them.

In addition,

> [W]e asked Michael S. Mahoney, a professor with significant expertise and experience in historiography—and specifically the history of computing—to provide guidance to the Program Committee and the authors. He reviewed *all* the papers and provided specific assistance to any author who requested it.

The Language Experts and the Program Committee Chair also supervised the development of the presentations, requiring that authors submit all presentation slides ahead of time for review. Thus HOPL II was thoroughly curated.

The organizers of HOPL II were concerned that a paper about the history of a programming language would be a paper about that programming language; therefore, they assigned to each paper a person or a couple of people to produce a language summary. Here is how this was described in the proceedings:

> We wanted the authors to concentrate on writing a history paper and not on describing the language. Therefore we asked knowledgeable individuals to provide a very short introduction to each language and these "language summaries" appeared in the preprints.

The Language Experts and Language Summary authors for each paper were named.

2.3 The History of Programming Languages III

In 2004, Barbara Ryder and Brent Hailpern started planning HOPL III, to be held in June 2007. This edition would have an open call for papers, and authors were advised to detail the early history or evolution of a specific programming language. Preliminary ideas for the language must have been documented by 1996 and the language must have been in use by 1998. Michael S. Mahoney was again the conference's Consulting Historian. In the Call for Papers, the Chairs wrote the following:

> As with its predecessors, HOPL-III will produce an accurate historical record of programming language design and development. To achieve this goal, the Program Committee will be working closely with prospective authors to help ensure that the all the [*sic*] papers

Fig. 3. The HOPL III Logo. (Courtesy ACM.)

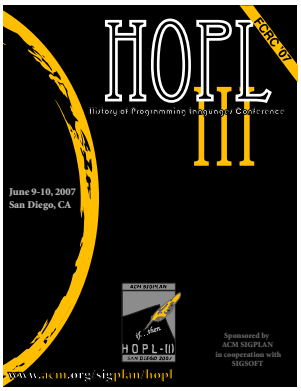

Fig. 4. The HOPL III Program Cover. (Courtesy ACM.)

are of high quality. As with HOPL-I and II, there will be **two rounds** of reviewing. The first round will select the papers for the conference. During the second round, the program committee will work with authors to polish the selected papers and ensure that they meet the requirements for both technical accuracy and historical completeness.

We will be providing prospective authors with a detailed set of Author Guidelines and submission information at the HOPL-III website. Because of the complex nature of the history of programming languages, there will be no a priori upper bound on the length of submitted papers—authors should strive for completeness.

Authors should submit a 1 page abstract by July 8, 2005. Full papers for the first round are due on August 15, 2005, with notification to authors in January 2006. Second round submissions will be due in August 2006, with notification to authors in January 2007. Camera ready copy will be due in March 2007.

The Author Guidelines were a revision of those for HOPL II, and included Content Guidelines [HOPL III 2007a] and two sets of questions on Early History [HOPL III 2007b] and Language Evolution [HOPL III 2007c]; the sets of questions on "Language Feature or Concept" and "Class of Languages" were dropped.

HOPL III used a small variant of the HOPL II badge as its logo (see Figure 3), but also created a vivid conference program cover (see Figure 4) that included the badge near the bottom.

Table 3. HOPL III Keynote Snippets

Guy Steele Starts	Dick Gabriel Ends
"Fifty in fifty." Simply a quirk? Fifty topics or languages? Fifty years of language work In fifty minutes? Clever guess, Yet subtler considerations lurk. To clarify one reason why: After we've finished this keynote speech, Dick will have made—and so will I— Fifty remarks of exactly fifty words each.	So there you have it: A tandem talk—a random walk, Less survey than picaresque. We marvel at the potpourri, The comely, homely, and grotesque; We revel in variety, In purpose, mischief, art, and craft. We hope you've learned, improved your wit— But if naught else, you've cried, or laughed.

HOPL III was held June 7–9, 2007, in San Diego, California, co-located with the Federated Computing Research Conference (FCRC). Guy L. Steele Jr. and Richard P. Gabriel presented a keynote entitled "50 in 50" [Steele and Gabriel 2007b].[3] This keynote started with Guy Steele and ended with Dick Gabriel reciting the starting and ending snippets shown in Table 3.

The keynote was a multimedia, performance-artish extravaganza that included music, verse, songs, slide artwork, and movies composed and produced especially for the presentation. The intent was to leave the audience with a brief, vivid impression of just one or two distinctive aspects of as many languages as they could. It ended with a long, slowly presented sequence of photos of those researchers and engineers whose work was mentioned in the talk but who had passed away. The purpose of the talk was to show that there were many more and much stranger programming languages than mainstream conferences and journals typically talk about, while also paying due attention to the development and evolution of important ideas in well-known languages. (As a point of reference: as of 2007, Diarmuid Pigott had catalogued 8,512 programming languages created and used after Grace Murray Hopper's A-0 compiler [Bergin 2007; Pigott 1995–2006, footnote 17], and as of May 2020, that catalogue now contains 8,945 entries [Pigott and Axtens 1995–2020].)

The proceedings were published in the ACM Digital Library [Ryder and Hailpern 2007]. The authors and papers are listed in Table 4. The Language Experts were thanked as follows:

> We must also thank the language experts who helped with the extensive paper reviews …

They were, however, not associated in the proceedings with the papers they worked with. HOPL III also did not have any discussants, though the possibility of using discussants was discussed.

3 THE HISTORY OF PROGRAMMING LANGUAGES IV

For HOPL IV, as for all the HOPLs, the process started with the creation or development of a programming language, programming mechanism, or programming-language idea by an individual or group. That artifact must have proven itself significant over a decade-long period. The creators or stewards of that artifact must be capable of writing a history of it. For HOPL IV, 62 such artifacts were identified by the Program Chairs using a variety of criteria. After the conventional call for papers was sent out, the designers and implementation teams for these 62 artifacts were sent "letters of encouragement" in Autumn 2017 (see Figure 5).

[3] This talk was reprised by invitation at several other conferences, including OOPSLA [Steele and Gabriel 2007a], JAOO [Steele and Gabriel 2008], and YOW! [Steele and Gabriel 2010]).

Table 4. HOPL III Authors and Papers

Author	Paper
William R. Cook	AppleScript
Roberto Ierusalimschy Luiz Henrique de Figueiredo Waldemar Celes	The Evolution of Lua
Niklaus Wirth	Modula-2 and Oberon
Bjarne Stroustrup	Evolving a Language in and for the Real World: C++ 1991–2006
David Harel	Statecharts in the Making: A Personal Account
Joe Armstrong	A History of Erlang
Ken Kennedy Charles Koelbel Hans Zima	The Rise and Fall of High Performance Fortran: An Historical Object Lesson
Lawrence Snyder	The Design and Development of ZPL
David Ungar Randall B. Smith	Self
Bent Bruun Kristensen Ole Lehrmann Madsen Birger Møller-Pedersen	The When, Why, and Why Not of the BETA Programming Language
Andrew P. Black Norman C. Hutchinson Eric Jul Henry M. Levy	The Development of the Emerald Programming Language
Paul Hudak John Hughes Simon Peyton Jones Philip Wadler	A History of Haskell: Being Lazy with Class

As for past HOPL conferences, no page limit was imposed—several of the papers presented in this issue are book-length. The call for papers noted the following:

> The criteria for the programming languages considered appropriate for HOPL IV are:
>
> (1) The programming language came into existence before 2009, that is, it was designed and described at least 11 years before HOPL IV (2020).
>
> (2a) The programming language has been widely used since 2011 either (i) commercially or (ii) within a specific domain. In either case, "widely used" implies use beyond its creators.
>
> (2b) Certain research languages had great influence on widely used languages that followed them. If the research language was used by more than its own inventors, and the paper clearly traces its influence on other languages in wider use, then it may be considered appropriate for discussion at HOPL IV.

The Program Chairs for HOPL IV along with HOPL IV Historian Mark Priestley prepared, for the use of authors preparing submissions, both a revised set of Content Guidelines [Steele and Gabriel 2018] and four extensively revised sets of Questions for Authors [Steele et al. 2018a,b,c,d] (reviving the two sets of questions from HOPL II that had been dropped for HOPL III). The introduction to the Content Guidelines for HOPL IV stated the following:

> A HOPL IV paper that is about a single specific language should detail the early history or evolution of that language and the motivation for creating that new language; ideas about the language should have been documented by 2009, and the language should have been widely used by 2011. A HOPL IV paper that addresses a language or language family already described in a previous HOPL conference should provide substantial new detail about language evolution, development and standardization activities, new dialects and implementations, significant publications, applications, user groups, and other technical and social consequences. A HOPL IV paper about a more general historical theme or trend should take a cross-language perspective to discuss and analyze some issue or design feature that has affected the historical development of programming language design, implementation, and usage over a span of at least twenty years.

Because most of the authors encouraged to submit a paper were not accustomed to writing history papers or even essays, and because most of the reviewers were not accustomed to reviewing history papers or even essays, two sets of training materials were written for authors and reviewers: "HOPL IV Reviewing Principles" [Gabriel et al. 2018] and "HOPL Shepherding" [Gabriel et al. 2019], both by Richard P. Gabriel, Mark Priestley, and Guy L. Steele Jr. The call for papers noted the following:

> The Program Committee will work closely with prospective authors to ensure that both the content and presentation of the papers are of high quality. There will be two rounds of careful reviewing. The first round will select the papers for the conference (conditional acceptance); the second round will polish the papers and ensure that they meet the requirements for technical accuracy, historical completeness, and clarity. The Program Chairs may also ask outside experts to provide additional reviews. For each selected paper, a member of the Program Committee will be assigned as a "shepherd" to ensure that intermediate drafts are carefully revised and written clearly, and that the recommendations of reviewers are addressed.

HOPL IV did not produce language summaries nor did it include discussants. And at least one language the organizers had a strong wish to receive a paper for was not written up—Java.

HOPL IV used a very intensive form of shepherding. Because in many cases targeted authors of HOPL IV papers were the only people who could write the hoped-for paper, and because for some of the authors it was "now or never," the Program Chairs considered it essential that these authors get as much help as reasonably possible. A HOPL shepherd is a combination of editor, copyeditor, content advisor, writing coach, and typesetter. In addition, a shepherd ensures the reviewers' comments are addressed. A shepherd's job includes the following:

> … pruning, shaping, clarifying, tidying inconsistencies of tense and pronouns and location and tone, noticing all the sentences that could be read in two different ways, dividing awkward long sentences into short ones, putting the writer back on the main road if he has strayed down a side path, building bridges where the writer has lost the reader by not paying attention to his transitions, questioning matters of judgment and taste. A ⟨shepherd's⟩ hand must also be invisible. Whatever he adds in his own words shouldn't sound like his own words; they should sound like the writer's words. [Zinsser 2006, pp. 299–300]

Some shepherds focused more on making sure reviewers' comments were addressed and addressed well, while others could arguably be considered co-authors. Where a shepherd landed on this spectrum depended on the paper and the authors in consultation with their assigned shepherds. Some papers had two shepherds.

Dear ⟨*names*⟩:

As Program Co-chairs, we (⟨*chairname1*⟩ and ⟨*chairname2*⟩) would like to strongly encourage you to consider writing a paper about ⟨*language or topic*⟩ for HOPL-IV: The Fourth ACM SIGPLAN History of Programming Languages Conference, to be held in June 2020.

[*Custom sentence(s), such as*: ⟨*language or topic*⟩ has been around since ⟨*year*⟩, has not yet appeared in HOPL, is now used widely for ⟨*purpose*⟩, and is one of the top ⟨*number*⟩ languages on the September 2017 TIOBE Index.] A paper on this topic would be of great interest to conference attendees and to future readers interested in the history of programming languages. We believe that you have primary knowledge on this topic and may be able to tell a clear and complete story. (We did our best to identify the relevant set of people to whom to send this letter; please feel free to forward it to others who might be appropriate co-authors.)

[*Other information or comments specific to the proposed language or topic.*]

HOPL is held only infrequently; the first three were in 1978, 1993, and 2007. HOPL-V will likely be held sometime after 2030. We believe that HOPL-IV in 2020 will be the best opportunity for you to tell the story of ⟨*language or topic*⟩.

HOPL typically publishes "the final word" on programming languages, and examines not only technical content but historical development and relationships to other languages and projects. Papers are typically longer than standard conference papers; most HOPL paper are at least 40 pages. Because of the complex nature of the history of programming languages, there is no upper bound on the length of submitted papers. We would prefer that authors strive for completeness rather than worry about page limits.

[*If appropriate*: We are aware that one of you is a member of the HOPL-IV program committee. The HOPL-IV policy (as stated in the Call for Papers and explained to prospective PC members when they were invited to serve) is: There will be no restriction on submissions from PC members. The program chairs/general chairs may not submit papers. PC papers will be held to a higher standard than other papers. The criterion for acceptability of a PC paper is a clear accept.]

The complete Call for Papers may be viewed at `http://hopl4.sigplan.org`. We strongly recommend that prospective authors examine papers presented at previous HOPL conferences to get an idea of appropriate length, content, and organization. The website `http://research.ihost.com/hopl` has information about past HOPL conferences and links to their proceedings.

We will say this up front: writing a successful paper for HOPL is a lot of work. (We know this from experience: we did it ourselves for HOPL-II.) Every paper goes through at least two full rounds of reviews, guided by an assigned "shepherd" from the program committee and with input from multiple program committee members, over the course of eighteen months. It is usually helpful to have co-authors, both to spread the load and to help make sure that details are not overlooked.

But maybe you don't want to write such a paper, or you're too busy with other things, or feel you aren't quite the right person for the job; in that case, can you please suggest to us who might be the right person(s)? It's important that historical information come from primary, original project participants, but sometimes it's better to have a separate writer pull the story together.

And we need to say one more thing up front: This is not an "invitation" to publish a paper that, once written, will be guaranteed publication. Every submitted paper must go through the same rigorous review and shepherding process, which may or may not result in accepting the paper. However, this *is* a letter of "strong encouragement"; we think that you participated in a very interesting part of the history of programming languages and that you may well have a great story to tell.

Initial drafts are needed about eleven months from now (August 31, 2018), so you have some time to think about it and maybe talk to potential co-authors. But if you want to talk to us about the possibilities or ask questions right away, please don't hesitate to contact us at `hopl4@sigplan.org`.

Yours,
⟨*chairname1*⟩
⟨*chairname2*⟩
Program Co-chairs, HOPL-IV

Fig. 5. Template used for each "letter of encouragement" sent to potential authors

Fig. 6. The HOPL IV Logo. (Courtesy Richard P. Gabriel.)

For HOPL IV, several shepherds dedicated more than two months of their time[4] to helping authors—a couple dedicated more than a month to each of several authors.

At the first (face-to-face) Review Committee meeting in September 2018, this training material was presented and a first pass of inspection was made of the 22 submitted drafts. The result was an initial set of recommendations for revision to be completed in advance of the second (also face-to-face) Review Committee meeting in January 2019.

At that January 2019 Review Committee meeting, 19 of the 22 submissions were conditionally accepted pending shepherding. Second drafts for all 19 papers were submitted by August 2019. After another four-month review period, the program committee conditionally accepted all 19 papers with specific stated requirements for final revisions.

Shepherding continued through the four-month review period and extended until after final acceptance—that is, until the end of April 2020—to aid preparation of final copy for publication.

In late 2019, PACMPL Editor-in-Chief Philip Wadler noted that HOPL papers required much more precise and thorough citations and references than was then-common practice. He commissioned a small group to come up with detailed recommendations for proper HOPL citations. The recommendations, along with other document preparation and typesetting guidelines, became the "Revised HOPL IV Style Guide" [Gabriel et al. 2020].

The organizers of HOPL IV adapted part of the HOPL II and HOPL III logos into the one they created afresh (Figure 6). It consists of a feather drawn (by Julie Steele[5]) and colored (by Jo Lawless[6]) specifically for HOPL IV, and the pen/inkwell portion of the quill pen is the personal logo of Richard Gabriel (designed for him by Rebecca Rikner). From this logo, Steele and Gabriel created a series of badges to be used for advertising, each echoing the HOPL II badge and HOPL III logo (Figure 7). The twelve badges contain code snippets (in font American Scribe) from twelve different programming systems. The intent was to illustrate concisely the wide variety of notations and concerns.

The HOPL IV Program Chair who served as Associate Editor for this issue thanked the shepherds and historian as follows:

> I would like to thank the reviewers, shepherds, and historian for their hard work; the high quality of the articles in this issue is also the result of these largely unsung volunteers. They have provided very useful feedback to the authors, in many cases spending hundreds of hours on their assignments, helping the authors to improve their work.

Moreover, the shepherd or shepherds for each paper are listed just below the author or authors on the title page of that paper.

[4]Two months of full-time effort, sometimes extending over the course of a year.
[5]Guy Steele's daughter.
[6]Richard Gabriel's wife.

Fig. 7. The HOPL IV Badges. The reader may find it amusing and/or edifying to match each of the 12 badges to one of these 11 languages (one of the languages is represented by two badges): Algol 60, Algol W, APL, C, C++, COBOL, Fortran 90, Lisp 1.5, Pascal, Perl, TEX. (Courtesy Richard P. Gabriel and Guy L. Steele Jr.)

4 FINAL REMARK

When we look at the frontier of research, we want to look for what is novel, what might become something wonderful—we invite people to give us their best. HOPL faces the other direction: it looks at what has become wonderful and asks its creators to tell us the story. The HOPL process is designed to help tell those stories well.

REFERENCES

Thomas J. Bergin and Richard G. Gibson (Eds.). 1996a. *History of Programming Languages II.* Association for Computing Machinery (ACM Press), New York, NY, USA. 831 book pages. 978-0-201-89502-5 https://doi.org/10.1145/234286 Proceedings of the Second ACM SIGPLAN History of Programming Languages Conference (HOPL-II), Cambridge, Massachusetts, USA, 20–23 April 1993. This entire book is available in the ACM Digital Library.

Thomas J. Bergin and Richard J. Gibson. 1996b. Conference Report: Supplemental Material from HOPL II. *SIGPLAN Notices* 31, 11 (Nov.), 9–20. 0362-1340 https://doi.org/10.1145/240964.1198155 Includes a transcript of remarks at the HOPL II conference banquet by Bernie Galler, Robert F. Rosin, Charles Lindsey, Jean Sammet, Joel Moses, Klaus (Niklaus) Wirth, Dick Nance, and Rich Miller.

Thomas J. (Tim) Bergin. 2007. A History of the History of Programming Languages. *Communications of the ACM* 50, 5 (May), 69–74. 0001-0782 https://doi.org/10.1145/1230819.1230841

Frederick P. Brooks. 1996. Keynote Address: Language Design as Design. In *History of Programming Languages II*, Thomas J. Bergin and Richard G. Gibson (Eds.). Association for Computing Machinery (ACM Press), New York, NY, USA, 4–16. 978-0-201-89502-5 https://doi.org/10.1145/234286.1057806 Transcript (including reproductions of slides) of HOPL II keynote talk given on 20 April 1993.

Richard P. Gabriel, Mark Priestley, and Guy L. Steele Jr.. 2018. HOPL IV Reviewing Principles. 13 Feb. 2018. 21 pages. https://hopl4.sigplan.org/getImage/orig/HOPLReviewing.pdf (also at Internet Archive 26 Oct. 2019).

Richard P. Gabriel, Mark Priestley, and Guy L. Steele Jr.. 2019. HOPL IV Reviewing Principles. Feb. 2019. 9 pages. https://hopl4.sigplan.org/getImage/orig/HOPLShepherding.pdf (also at Internet Archive 26 Oct. 2019).

Richard P. Gabriel, Guy L. Steele Jr., and Mark Priestley. 2020. Revised HOPL IV Style Guide. 18 Feb. 2020. 33 pages. https://hopl4.sigplan.org/getImage/orig/Revised+HOPL+IV+Style+Guide.pdf (also at Internet Archive 19 Feb. 2020).

HOPL. 1981. General Questions Asked of All Authors. In *History of Programming Languages*, Richard L. Wexelblat (Ed.). Academic Press, New York, NY, USA, 722–728. 978-0-12-745040-7 https://doi.org/10.1145/800025 Contained in Appendix B, in the back matter.

HOPL II. 1996. Content Guidelines for Authors. In *History of Programming Languages II*, Thomas J. Bergin and Richard G. Gibson (Eds.). Association for Computing Machinery (ACM Press), New York, NY, USA, 835. 978-0-201-89502-5 https://doi.org/10.1145/234286 Contained in Appendix B, in the back matter.

HOPL III. 2007a. Content Guidelines for Authors. In *Proceedings of the Third ACM SIGPLAN Conference on History of Programming Languages (HOPL-III)*, Barbara Ryder and Brent Hailpern (Eds.). Association for Computing Machinery, New York, NY, USA, BM-1–BM-2. 9781595937667 https://doi.org/10.1145/1238844 Contained in the Appendix, in the back matter.

HOPL III. 2007b. Questions on the Early History of a Language. In *Proceedings of the Third ACM SIGPLAN Conference on History of Programming Languages (HOPL-III)*, Barbara Ryder and Brent Hailpern (Eds.). Association for Computing Machinery, New York, NY, USA, BM-3–BM-12. 9781595937667 https://doi.org/10.1145/1238844 Contained in the Appendix, in the back matter.

HOPL III. 2007c. Questions on the Evolution of a Programming Language. In *Proceedings of the Third ACM SIGPLAN Conference on History of Programming Languages (HOPL-III)*, Barbara Ryder and Brent Hailpern (Eds.). Association for Computing Machinery, New York, NY, USA, BM-13–BM-20. 9781595937667 https://doi.org/10.1145/1238844 Contained in the Appendix, in the back matter.

Gracy Murray Hopper. 1981. Keynote Address. In *History of Programming Languages*, Richard L. Wexelblat (Ed.). Academic Press, New York, NY, USA, 7–20. 978-0-12-745040-7 https://doi.org/10.1145/800025.1198341 Transcript of HOPL keynote talk given on 1 June 1978.

Randy Hudson (Ed.). 1996a. Questions on the Early History of a Language. In *History of Programming Languages II*, Thomas J. Bergin and Richard G. Gibson (Eds.). Association for Computing Machinery (ACM Press), New York, NY, USA, 836–841. 978-0-201-89502-5 https://doi.org/10.1145/234286 Contained in Appendix B, in the back matter.

Randy Hudson (Ed.). 1996b. Questions on the Evolution of a Programming Language. In *History of Programming Languages II*, Thomas J. Bergin and Richard G. Gibson (Eds.). Association for Computing Machinery (ACM Press), New York, NY, USA, 841–844. 978-0-201-89502-5 https://doi.org/10.1145/234286 Contained in Appendix B, in the back matter.

Randy Hudson (Ed.). 1996c. Questions on the History of a Class of Languages. In *History of Programming Languages II*, Thomas J. Bergin and Richard G. Gibson (Eds.). Association for Computing Machinery (ACM Press), New York, NY, USA, 846–847. 978-0-201-89502-5 https://doi.org/10.1145/234286 Contained in Appendix B, in the back matter.

Randy Hudson (Ed.). 1996d. Questions on the History of a Language Feature or Concept. In *History of Programming Languages II*, Thomas J. Bergin and Richard G. Gibson (Eds.). Association for Computing Machinery (ACM Press), New York, NY, USA, 845–846. 978-0-201-89502-5 https://doi.org/10.1145/234286 Contained in Appendix B, in the back matter.

Michael S. Mahoney. 1996a. Making History. In *History of Programming Languages II*, Thomas J. Bergin and Richard G. Gibson (Eds.). Association for Computing Machinery, New York, NY, USA, 24–26. 978-0-201-89502-5 https://doi.org/10.

1145/234286.1057808 Transcript of remarks by the conference historian at the opening session.

Michael S. Mahoney. 1996b. What Makes History? In *History of Programming Languages II*, Thomas J. Bergin and Richard G. Gibson (Eds.). Association for Computing Machinery, New York, NY, USA, 831–832. 0201895021 https://doi.org/10.1145/234286.1057848

Diarmuid J. Pigott. 1995–2006. HOPL: An Interactive Roster of Programming Languages (website). http://hopl.murdoch.edu.au/home.prx (also at Internet Archive 9 July 2011).

This site lists 8512 languages, complete with 17837 bibliographic records featuring 11064 extracts from those references. It is in effect a family tree of languages with 5445 links, making it not only the biggest programming language family tree around, but also one of the largest idea-genealogical projects undertaken.

The site is in the form of a Gernsback machine, a hypernavigable multidimensional facet tree for which the languages, systems, organisations and people are instantiations, and which permits exploration by stepping, flipping, and zooming in and out on any of the points of data.

Diarmuid J. Pigott and Bruce Axtens. 1995–2020. Online Historical Encyclopaedia of Programming Languages (website). http://hopl.info/ (also at Internet Archive 10 April 2020).

An online roster and genealogy of 8945 programming languages from the 18th century to the present, featuring 7,800 influence links and over 11,000 citations.

This site is concerned with the idea-historical treatment of the development of programming languages as a means of human expression and creation.

Barbara Ryder and Brent Hailpern (Eds.). 2007. *Proceedings of the Third ACM SIGPLAN Conference on History of Programming Languages (HOPL-III)*. Association for Computing Machinery, New York, NY, USA. 9781595937667 https://doi.org/10.1145/1238844 San Diego, California, USA, 9–10 June 2007. This entire book is available in the ACM Digital Library.

Jean E. Sammet. 1969. *Programming Languages: History and Fundamentals*. Prentice-Hall, Englewood Cliffs, New Jersey, USA. 785 book pages. 978-0-13-729988-1

Guy L. Steele Jr. and Richard P. Gabriel. 2007a. 50 in 50. In *OOPSLA Companion '07: Companion to the Proceedings of the 22nd ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications* (Montréal, Quebec, Canada). Association for Computing Machinery, New York, NY, USA, 721. 978-1-59593-865-7 https://doi.org/10.1145/1297846.1297850

Guy L. Steele Jr. and Richard P. Gabriel. 2007b. Presentation: "50 in 50" (video recording). In *HOPL III: Proc. Third ACM SIGPLAN Conference on History of Programming Languages* (video recording) (San Diego, California). Association for Computing Machinery, New York, NY, USA. 978-1-59593-766-7 https://doi.org/10.1145/1238844.1411839 Invited keynote (86 minutes).

Guy L. Steele Jr. and Richard P. Gabriel. 2008. 50 in 50. In *JAOO Conference* (Aarhus, Denmark, 1 Oct.). JAOO, Aarhus, Denmark. http://jaoo.dk/aarhus-2008/presentation/50+in+50 (also at Internet Archive 7 March 2009). Invited keynote.

Guy L. Steele Jr. and Richard P. Gabriel. 2010. YOW! 2010 50 in 50 Keynote Guy Steele and Richard Gabriel (video recording). Vimeo. 6 Dec. 2010. NON-ARCHIVAL https://vimeo.com/25958308 Invited talk, YOW! 2010 Conference, Brisbane, Australia (75 minutes).

Guy L. Steele Jr. and Richard P. Gabriel. 2018. Questions for Authors: Early History (website document). 13 Feb. 2018. https://hopl4.sigplan.org/track/hopl-4-papers#Content-Guidelines-for-Authors (also at Internet Archive 29 Feb. 2020). Reprinted in this issue of PACMPL.

Guy L. Steele Jr., Richard P. Gabriel, and Mark Priestley. 2018a. Questions for Authors: Class of Languages (website document). 13 Feb. 2018. https://hopl4.sigplan.org/track/hopl-4-papers#Questions-for-Authors-Class-of-Languages (also at Internet Archive 29 Feb. 2020). Reprinted in this issue of PACMPL.

Guy L. Steele Jr., Richard P. Gabriel, and Mark Priestley. 2018b. Questions for Authors: Early History (website document). 13 Feb. 2018. https://hopl4.sigplan.org/track/hopl-4-papers#Questions-for-Authors-Early-History (also at Internet Archive 29 Feb. 2020). Reprinted in this issue of PACMPL.

Guy L. Steele Jr., Richard P. Gabriel, and Mark Priestley. 2018c. Questions for Authors: Feature or Concept (website document). 13 Feb. 2018. https://hopl4.sigplan.org/track/hopl-4-papers#Questions-for-Authors-Feature-or-Concept (also at Internet Archive 29 Feb. 2020). Reprinted in this issue of PACMPL.

Guy L. Steele Jr., Richard P. Gabriel, and Mark Priestley. 2018d. Questions for Authors: Language Evolution (website document). 13 Feb. 2018. https://hopl4.sigplan.org/track/hopl-4-papers#Questions-for-Authors-Language-Evolution (also at Internet Archive 29 Feb. 2020). Reprinted in this issue of PACMPL.

Richard L. Wexelblat. 1978. Preprints. In *ACM SIGPLAN History of Programming Languages Conference* (Los Angeles, California, USA, 1–3 June 1978). Association for Computing Machinery, New York, NY, USA, v–xviii, 1–310. *SIGPLAN Notices* 13, 8 (Aug. 1978). 0362-1340 https://doi.org/10.1145/960118

Richard L. Wexelblat (Ed.). 1981. *History of Programming Languages*. Academic Press, New York, NY, USA. 795 book pages. 978-0-12-745040-7 https://doi.org/10.1145/800025 Proceedings of the first ACM SIGPLAN History of Programming

Languages Conference (HOPL), Los Angeles, California, USA, 1–3 June 1978. This entire book is available in the ACM Digital Library.

Richard L. Wexelblat. 1993. Preprints. In *Second ACM SIGPLAN History of Programming Languages Conference* (Cambridge, Massachusetts, USA, 20–23 April 1993). Association for Computing Machinery, New York, NY, USA, 1–370. *SIGPLAN Notices* 28, 3 (March 1993). 0362-1340 https://doi.org/10.1145/154766 These preprints include fourteen "brief language summaries" that do not appear in the final proceedings.

William Zinsser. 2006. *On Writing Well: The Classic Guide to Writing Nonfiction* (30th anniversary edition). HarperCollins, New York, NY, USA. 978-0060891541

Appendix B: HOPL IV Content Guidelines for Authors and HOPL IV Questions for Authors

GUY L. STEELE JR., Oracle Labs, USA, *Associate Editor*
RICHARD P. GABRIEL (poet, writer, computer scientist), California

B

The HOPL IV Program Chairs prepared five documents to assist authors in the preparation of their initial submissions: the Content Guidelines for Authors (an updated version of guidelines provided for HOPL II and HOPL III) and four sets of Questions for Authors (modeled on similar sets of questions provided for HOPL II and HOPL III, but updated, expanded, and specialized for each of the four classes of papers to be considered). These five documents are presented here.

CONTENT GUIDELINES FOR AUTHORS

The first three History of Programming Languages conferences established high technical and editorial standards. We trust that your contribution will help HOPL-IV maintain or surpass these standards. These guidelines are intended to help you develop the appropriate content for your contribution to HOPL-IV.

Appropriate Languages

The criteria for the programming languages considered appropriate for HOPL-IV are:

(1) The programming language came into existence before 2009, that is, it was designed and described at least 11 years before HOPL-IV (2020).
(2a) The programming language has been widely used since 2011 either (i) commercially or (ii) within a specific domain. In either case, "widely used" implies use beyond its creators.
(2b) Certain research languages had great influence on widely used languages that followed them. If the research language was used by more than its own inventors, and the paper clearly traces its influence on other languages in wider use, then it may be considered appropriate for discussion at HOPL-IV.

Please be sure to include within your paper a clear indication of how the subject material satisfies these criteria—not only (1), but also either (2a)(i) or (2a)(ii) or (2b). This information can certainly be provided indirectly as part of the overall text. For instance, some of the criteria are dates; it suffices to include the dates as part of the historical narrative.

Categories of Submissions

There are four kinds of papers that may be considered for HOPL-IV:

- *Early History* of a specific language, typically including the story of its initial design and first implementation(s)
- Later *Evolution* of a specific language (usually a language already treated in an earlier HOPL conference)
- A cross-language examination of a specific *Feature or Concept*
- Consideration of a *Class of Languages* having a common theme or purpose

These categories overlap to some extent; for example, Evolution of a single initial language may result in a Class of Languages that share some Feature or Concept. It is not necessary to label

your submission as belonging to a specific category; rather, the categories are intended as a tool to help you organize the story you want to tell and to make sure that relevant information is not overlooked.

Questions for Authors

For each category we provide a set of questions that you should consider when writing your paper (to see them, click on the links in the list shown above). The questions point to the kind of information that people want to know about the history of programming languages.The four sets of questions overlap to some extent; even within a single set, the same question, or very similar questions, may be asked in different contexts. Please draft your paper in light of these different emphases and contexts. It may be appropriate for you to examine more than one set of questions. Conversely, keep in mind that there may be important aspects of your specific story that the questions fail to address.

Your paper should try to answer as many questions as possible, but it is understood that you might not be able to address every one of them. Because history can unfold in so many different ways, some of the questions may be clearly irrelevant to your particular topic, or your particular point of view. The information requested might no longer be available, but please remember that the very fact of unavailability may be important to report as well. Several questions are of the form "How did something affect something else?"; it may be important for the historical record to assert that "it didn't."

A question set is intended to suggest the content, not the form, of your paper. (In particular, your paper should *not* be in question-answer format!) The questions are organized into topics and subtopics for your convenience during your research; this structure is not meant to serve as an outline for your paper. (Topics, subtopics, and questions are also numbered and lettered for convenience of reference.) Please feel free to use whatever form and style for your paper seems most appropriate and comfortable to you.

We recommend that you read two pages of advice from Michael S. Mahoney about "What Makes History?" This will help you to adopt an appropriate historical perspective when writing.

Examples

The Program Committee strongly suggests that you examine at least one paper from each of the proceedings of HOPL-I, HOPL-II, and HOPL-III to see what earlier contributors did. Links to all prior HOPL papers (within the ACM Digital Library) may be found here. We also suggest here examples of papers in each category:

- Early History: COBOL or FORTRAN or LISP or ALGOL 68 or Erlang or Haskell or High Performance Fortran (HPF)
- Evolution: Evolution of Lisp
- Feature or Concept: Monitors and Concurrent Pascal
- Class of Languages: Discrete Event Simulation Languages

QUESTIONS FOR AUTHORS: EARLY HISTORY

The principal objective of a contribution in this category is to record the history of the origins, early development, and initial deployment of a single programming language.

I. BACKGROUND

These questions are directed primarily to the origins of the project and to the people involved and how they were organized.

1. **Motivation**
   (a) What (or who) was the real origin of the idea to develop this language? (This question may have more than one good answer; the choice may shape the perspective of your paper. Be sure to consider whether there are multiple perspectives.)
   (b) Who was the prime mover for the development effort?
   (c) What was the primary motivation in developing the language (e.g., research, task assigned by management)?
   (d) Did the motivation change over time before the language was released? After the language was released?
2. **Languages and Systems Known at the Time**
   (a) What other specific languages were known to you and/or other members of the development group at the time the work started? Which other languages did any of you learn about as the work progressed? Was such learning in intentional pursuit of information to guide the design or development of the new language? How much did you know about these languages and in what ways (e.g., as users, from having read unpublished and/or published papers, informal conversations, from documentation, from reading the code for the implementation, from having participated in the design or implementation)? Please try to distinguish between what you, the author, knew at the time and what the other members of the project knew at the time.
   (b) Were these other languages regarded as formal inputs to consider in your own language development, or did they merely provide background? What was it about these languages that you wanted to emulate (e.g., syntax, capabilities, internal structure, application area, etc.)?
   (c) How influenced were you by these languages? Put another way, how much did the prior language backgrounds of you and other members of the group influence the early language design? Whether the answer is "a lot" or "a little," why did these other languages have that level of influence?
   (d) Was there a primary source of inspiration for the language, and if so, what was it? Was the language modeled after this (or any other predecessors or prototypes)?
   (e) How did information about each of these other languages or systems become available to the people it influenced?
   (f) Were any specific new features borrowed from, inspired by, or created as an alternative to specific features in other languages or systems? Why?
   (g) Were any proposed or plausible features omitted or removed in response to experience with other languages or systems? Why?
   (h) Was the design, development, or deployment process modeled on that used for another language or system? Why?
   (i) Was the design, development, or deployment process intentionally done differently from that used for another language or system? Why?

3. **Basic Facts about Activities, Organization, and People**
   (a) Under what organizational auspices (if any) was the language developed (e.g., name of company, department/division in the company, university, consortium or collaboration, open-source project, standardization effort)? Be as specific as possible about organizational units and subunits involved.
   (b) What, if any, was the nature of any cooperation or competition among multiple organizations or organizational subunits?
   (c) Were there problems or conflicts within the organization(s) in getting the project started? If so, please indicate what these were and how they were resolved.
   (d) What was the source of funding (e.g., research grant, company R&D, company production units, government contract)?
   (e) Who were the people on the project and what was their relationship to the author(s) (e.g., team members, subordinates, managers)? To the largest extent possible, name all the people involved, including part-timers, when each person joined the project, and what each person worked on and when. Indicate the technical experience and background of each participant, including formal education. Please be as specific as possible regarding names, titles, and dates.
   (f) How did the roles of various individuals change during the course of the activity?
   (g) In what ways did geographic factors affect what was done, who participated, how they were organized, and how they communicated?
   (h) In what ways did organizational or political factors affect what was done, who participated, how they were organized, and how they communicated?
   (i) In what ways did computerized or networked facilities affect what was done, who participated, how they were organized, and how they communicated?
   (j) In what ways did the availability (or lack) of other resources affect what was done, who participated, how they were organized, and how they communicated?
   (k) Were the design and implementation done together, by a single person or team? Alternatively, were there separate design and implementation steps or phases, and if so, were the design and the implementation done by the same people or different (possibly overlapping) teams?
   (l) Was the design done originally as a research project, as a development project, as a committee effort, as an open-source effort, as a one-person effort with some minor assistance, or what? Was it a side effect of some other effort that happened to produce a language as a subgoal, but the language itself ended up being important in its own right?
   (m) Was the implementation done originally as a research project, as a development project, as a committee effort, as an open-source effort, as a one-person effort with some minor assistance, or what?
   (n) Was there a separate documentation team, or was documentation written by designers and/or implementors?
   (o) Was testing ("quality assurance") done by a separate team? By the designers or implementors?
   (p) Were there any persons whose roles were "marketing" or "user liaison" or "user training"? If so, who fulfilled such roles, and how?
   (q) Was there a defined leader to the group? If so, what was his or her exact position (and title) and how did he or she get to be the leader (e.g., appointed "from above," self-starter, volunteer, elected)?

(r) Was there a *de facto* leader different from the defined leader? If so, who was this leader and what made this person the *de facto* leader (personality, background, experience, a "higher authority," something else)? Was one person the "manager" and another the "technical lead"? Or did more than one person fill either or these roles? Was technical leadership divided according to multiple design or implementation areas?
(s) Were there consultants from outside the project who had a formal connection to it? If so, who were they, how and why were they chosen, and how much help were they? Were there also informal consultants? If so, please answer the same questions.
(t) Did the participants in the project view themselves primarily as language designers, as implementors, or as eventual users? If there were some of each working on the project, indicate the split as much as possible. How did this internal view of the people involved affect the initial planning and organization of the project?
(u) Did the language designers know (or believe) that they would also have the responsibility for implementing the first version? Whether the answer is "yes" or "no," was the technical language design affected by this?
(v) Did the language designers know (or believe) that they would also have the responsibility for maintaining subsequent versions? Whether the answer is "yes" or "no," was the technical language design affected by this?

4. **Costs and Schedules**
   (a) Was there a budget? Was it adequate? Did the budget provide a fixed upper limit on the costs? If so, how much money was to be allocated and in what ways? What external or internal factors led to the budget constraints? Was the money formally divided between language design and actual implementation? If so, in what way?
   (b) Was there a fixed deadline for completion of the project? Was the project divided into phases and did these have deadlines? How well were the deadlines met?
   (c) What is the best estimate for the amount of human resources involved (i.e., in person-years)? How much was for language design, for documentation, for implementation, and for other functions?
   (d) What is the best estimate of the costs prior to putting the first system in the hands of the first users? If possible, show as much breakdown on this as possible.
   (e) If there were cost and/or schedule constraints, how did that affect the language design and in what ways?
   (f) What other resource constraints affected the schedule?
   (g) Did any external constraints (such as customer requirements or competition with another organization) influence the schedule?
5. **Basic Facts about Documentation**
   (a) What are the significant publications that arose from design, development, testing, and deployment? For each provide:
      (a1) Full bibliographic citation
      (a2) Names of authors (if not part of the reference)
      (a3) Intended audience (e.g., user, implementer, language researcher)
      (a4) Format (e.g., manual, general trade book, standards document, conference paper, web page, blog entry)
      (a5) Availability
   (b) In the planning stage, was there consideration of the need for documentation of the work as it progressed? If so, was it for internal communication among project members or external monitoring of the project by others, or both?

(c) What types of documentation were decided upon?
(d) Did any sort of intentional specification precede implementation, or was an initial implementation followed by documentation?
(e) If documentation and implementation were found to be in conflict, was one generally considered more definitive than the other? If so, which one, and why? Did this change over time?
(f) Was a test suite developed to verify the implementation? If so, was it used to perform regression tests on the compiler or other parts of the implementation? Was the test suite made public? To what extent was the test suite considered part of the (internal or external) specification or documentation?
(g) To the largest extent possible, cite both dates and documents for the following (including internal papers and web sites which may not have been released outside of the project) by title, date, and author.
    (g1) Initial idea
    (g2) First documentation of initial idea
    (g3) Preliminary specifications
    (g4) "Final" specifications (i.e., those which were intended to be implemented)
    (g5) "Prototype" running (i.e., as thoroughly debugged as the state of the art permitted, but perhaps not all of the features included)
    (g6) "Full" language compiler (or interpreter) was running
    (g7) Usage on real problems done by the developers
    (g8) Usage on real problems done by people other than the developers
    (g9) Documentation by formal methods
    (g10) Paper(s) in professional journals or conference proceedings
    (g11) Please identify extensions, modifications and new versions
    (g12) Independent implementations by other groups or organizations

    For each of these stages, where relevant, indicate the level of formality of the specifications. Were specifications entirely in prose (in English or some other "natural language"), or were any formalisms used (examples include BNF, regular expressions or other ways of marking optional or repeated elements in the syntax, syntax diagrams (aka "railway charts"), attribute grammars, mathematical equations, algebraic specifications, denotational semantics, the Vienna definition language, compiler-construction tools such as parser generators, and proof assistants)? Were the formalisms directed primarily to the explanation of syntax, or was formalism also used in the explanation of semantics (that is, program behavior)?
(h) If there were independent implementations, did they rely entirely on specifications and documentation, or did they read or borrow from an existing code base and/or communicate directly with the original development team?
(i) Please reflect on the project documents as historical sources. Throughout your paper, identify when you are relying on these documents for information, and when you are relying on memory or other sources. Are there places where memory and documentation conflict? Do you now regard the available documentation as a complete and faithful record of the history, or can you identify gaps or incorrect information in the documents?
(j) It may be appropriate to reproduce a limited number of original documents or photographs in your paper. Which ones are the most relevant for telling the story?

6. **Intended Purposes and Users**
   (a) For what application area was the language designed? What type of problems was it intended to be used for? Traditional labels such as "business data processing" or "scientific applications" may be too vague; please be as specific as possible and appropriate in describing the application area.
      (a1) Was a specific statement of application area or purpose drafted when the project started? If so, did it change over the course of the project?
      (a2) Was the apparent application area of some other language used as a model?
   (b) For what types of users was the language intended (e.g., experienced programmers, mathematicians, business people, novice programmers, non-programmers)? Was there any conflict within the group on this? Were compromises made, and if so, were they made for technical or non-technical reasons?
   (c) Why develop a new language? Why were existing languages deemed sufficiently inadequate for the purpose that a new effort was justified?
      (c1) Was the design developed from whole cloth in response to the stated application area or purpose?
      (c2) Was the intent to create a language "just like language X, except for thus-and-so specific improvements"?
      (c3) Was the intent to create a synthesis of the best ideas from two or more other languages?
      (c4) Was the language created in order to exemplify or promulgate a specific theoretical idea?
      (c5) Was the language initially the result of "just hacking around," perhaps by grafting a new idea or two onto an existing language?
   (d) How much of the language design effort went into application-specific features, and how much of it went into "structural issues" or "software engineering issues" such as modularity, namespace management, debugging support, or features coders of libraries might need as opposed to application programmers?
   (e) What equipment was the language originally intended to be implemented on? Wherever possible, cite specific machine(s) by manufacturer(s) and model numbers, or alternatively, give broad descriptions relevant to the time period with specific examples. Consider this example from the HOPL-I paper about COBOL:

      > COBOL was designed from its inception to be used on large computers (by 1959 standards, although by 1978 standards they would be considered minicomputers), intended for business data processing use. In particular, machines for which COBOL was certainly being considered were the UNIVAC I and II, the IBM 705 and 709, the Honeywell 800, RCA 501, Sylvania MOBIDIC, [and] Burroughs B-5000.

      Was machine independence a significant design goal, albeit within this class of machines?
7. **Distribution**
   (a) Was the language specification (information needed to create a complete implementation) made available to users? Was it considered proprietary? Was it provided for free or for sale? What distribution media were used?
   (b) How was the language implementation made available to users? Was it considered proprietary? Was it provided for free, for lease or subscription, or for sale? What

distribution media were used? Were users able to run it on their own hardware, or only as part of a service on hardware they did not own?
(c) How was the language documentation (information needed to use the language, which may or may not include the specification and may or may not include tutorial material) made available to users? Was it considered proprietary? Was it provided for free or for sale? What distribution media were used?
(d) Did the answers to any of the preceding questions change over time?

II. RATIONALE OF THE CONTENT OF THE LANGUAGE

These questions are directed to the technical content of the language design and implementation effort and are intended to stimulate thought about various factors that affect most language design efforts. Not all the questions are relevant for every language. They are intended to suggest areas that might be addressed in each paper. This is not necessarily an exhaustive list of relevant questions.

1. **Environmental Factors**

To what extent was the design of the language influenced by:

(a) Program Size: Was it explicitly thought that programs written in the language would be large and/or written by more than one programmer? Or, conversely, was it explicitly thought that typical programs would be small and/or written primarily by one programmer? What features were explicitly included (or excluded) for this reason? If this factor wasn't considered, did it turn out to be a mistake? Were specific tools or development environments designed at the same time to support these choices?
(b) Program Libraries and/or Frameworks: Were "program libraries" (or "standard frameworks") envisioned as necessary or desirable, and if so, how much provision was made for them? Were "standard libraries" a major portion of the language design effort? If so, what libraries were especially important? Which libraries were novel, unusual, or essential to the "flavor" of the language? Alternatively, was it an important design goal of the language to be able to interoperate with existing libraries already coded in another language?
(c) Portability: How important was the goal of machine independence? What features reflect concern for portability? How well was this goal attained?
(d) User Background and Training: What features catered to the expected background of intended users?
    (d1) Were any notational or semantic features of the language intentionally modeled on notations or semantics not widely used in computer science but customary in an intended domain of application?
    (d2) How difficult did it prove to train users in the correct and effective use of the language, and was the difficulty a surprise? What changes in the language would have alleviated training problems? Were any proposed features rejected because it was felt users would not be able to use them correctly or appropriately?
    (d3) In retrospect, what features of the language proved to be difficult for programmers to use correctly? Did some features fall into disuse? Please identify such features and explain why they fell into disuse.
(e) Execution Efficiency: How did requirements for executable code size and speed affect the language design? Were programs in the language expected to execute on large or small computers (i.e., was the size of object programs and/or execution speed expected to pose a problem)? What design decisions were explicitly motivated by the concern (or lack of concern) for execution efficiency? Did these concerns turn out

to be accurate? How was the design of specific features changed to make it easier to optimize executable code?

(f) Target Computer Architecture: To what extent were features in the language dictated by the characteristics of the anticipated target computer (e.g., word size, floating-point hardware, instruction set peculiarities, application-specific instructions, register or cache structure, special-purpose co-processors and accelerators, parallelism, synchronization features, transactional memory)? Were there multiple targets, and if so, in what way did this affect the language design or implementation?

(g) Compilation Environment: To what extent, if any, did concerns about compilation efficiency affect the design? Were features rejected or included primarily to make it easier to implement compilers for the language or to ensure that the compiler(s) would execute quickly? In retrospect, how correct or incorrect do you feel these decisions were? What decisions did you make regarding use of the compiler run-time system?

(h) Programming Ease: To what extent was the ease of coding an important consideration and what features in the language reflect the relative importance of this goal? Did maintainability considerations affect any design decisions? If so, which ones?

(i) Execution Environment: To what extent did the language design reflect its anticipated use in a batch, timeshared, embedded, portable, mobile, wearable, kiosk, office, or networked environment? What features reflect these concerns?

(j) Multiple Implementations: Were there multiple implementations being developed at the same time as the later part of the language development? If so, was the language design hampered, improved, or influenced by this in any way?

(k) Standardization: In addition to (or possibly separate from) the issue of portability, what considerations were given to possible standardization? What types of standardization were considered, and what groups were involved and when?

(l) Networking/Parallel Environment: To what extent did the language design reflect its anticipated use in a networked- or parallel-execution environment? What features reflect these concerns? Was execution efficiency in a networked or parallel environment (as opposed to single-CPU efficiency) a concern?

(m) Interoperability: Was it important for programs written in this language to interoperate with code written in other languages, operating systems, or other tools? If so, which ones, and why? Did this require specific technical features in the new language?

(n) Documentation: Were features included to specifically to support documentation of programs coded in the language? If so, did they support documentation tied to the program representation, generation of separate documents (paper documents, files, web pages), or both? What tools and conventions were used or supported?

(o) Reliability: Were features included specifically to support detection or prevention of programming errors and/or proofs of program correctness? Were otherwise common or familiar features omitted in order to help prevent programming errors?

(p) Debugging: Were features included specifically to support the debugging process?

2. **Functions to be Programmed**

(a) How did the operations and data types in the language support the writing of particular kinds of algorithms or applications?

(b) What features might have been left out, if a slightly different application area had been in mind?

(c) What features were considered essential to properly express the kinds of programs to be written?

(d) What misconceptions about application requirements turned up that necessitated redesign of application-specific features before the language was actually released?
(e) To what extent were application-specific features incorporated directly into the language design, and to what extent was the expectation that application-specific features would be provided (perhaps as libraries) by coding in the language itself? Were any features of the language specifically intended to support or enable the latter?

3. **Language Design Principles**
   (a) What consideration, if any, was given to designing the language so that programming errors could be detected early and easily? Were the problems of debugging and testing considered? Were debugging and testing facilities deliberately included in the language?
   (b) To what extent was the goal of keeping the language simple considered important? What kind of simplicity was considered most important? What did your group mean by "simplicity"? Was there an explicit statement of simplicity that guided the design effort?
   (c) To what extent was the goal of keeping the language consistent considered important? What kind of consistency was considered most important? What did your group mean by "consistency"? Was there an explicit statement of consistency that guided the design effort?
   (d) Were there any other explicitly stated principles that guided the language design? If so, were any specific proposed features modified or omitted after coming into conflict with a stated principle?
   (e) What thought was given to make programs more understandable and how did these considerations influence the design? Was there conscious consideration of making programs "easy to read" versus "easy to write"? If so, which were chosen and why?
   (f) Did you consciously develop the data types first and then the operations, or did you use the opposite order, or did you try to develop both in parallel with appropriate iteration? Were data and operations combined into objects?
   (g) To what extent did the design reflect a conscious philosophy of how languages should be designed (or how programs should be developed)? What was this philosophy?
   (h) Did you intentionally model the language according to an existing style or school of thought (e.g., Algol-like, Lisp-like, macro-like, imperative, declarative, block-structured, object-oriented, functional, pure, or stack-based)? If so, why? In what ways does your resulting design reflect this style or school of thought, and in what ways does it differ, and why?
   (i) Were any slogans or catchphrases used as summaries or reminders of language design principles? Were they used internally or also externally? Were they used seriously or jocularly?
   (j) Did the developers of the language make any attempt to evaluate specific features, or the entire language design, objectively/empirically? Did the results lead to changes in the language design? Did such changes result in programs that were somehow better (e.g., shorter, more reliable, more efficient, easier to read, easier to maintain)?

4. **Language Specification**
   (a) What techniques for specifying languages were known to you? Did you use these or modify them, or did you develop new ones?
   (b) To what extent (if any) was the language itself influenced by the technique used for the specification?

(c) What formalisms and/or tools were used in the specification process?

5. **Concepts About Other Languages**
   (a) Were you consciously trying to introduce new concepts? If so, what were they? Do you feel that you succeeded? (Conversely, was it a design goal to *avoid* introducing new concepts? If so, why?)
   (b) If you were not trying to introduce new concepts, what was the justification for introducing this new language? (Such justification might involve technical, personal, political, or economic factors.)
   (c) To what extent did the design consciously borrow from previous language designs or attempt to correct perceived mistakes in other languages?
6. **Influence of Non-technical Factors**
   (a) How did time and cost constraints (as described in the Background section) influence the technical design?
   (b) How did the size and structure of the design group affect the technical design?
   (c) How did the size and structure of the implementation group affect the implementation?
   (d) How did the size and structure of the documentation group affect the documentation?
   (e) Did the membership of any of these groups change over time, and if so, how did such changes affect the design, implementation, or documentation efforts?
   (f) Provide any other information you have pertaining to ways in which the technical language design was influenced or affected by non-technical factors.

III. A POSTERIORI EVALUATION

These questions are directed to assessing the language project from today's standpoint.

1. **Meeting of Objectives**
   (a) How well do you think the language met its original objectives?
   (b) What is the size and nature of the current user community? Do the users think the language has met its objectives? Do the users think the language has met their needs? If not, what do users think could be done, or should have been done, to further improve the language?
   (c) How well do you think the computing community (as a whole) thinks the objectives were met?
   (d) How much impact did portability (i.e., machine independence) have on acceptance by users?
   (e) How much impact did other specific aspects of the language (such as execution efficiency, compilation efficiency, ease of programming, or specific libraries) have on acceptance by users?
   (f) Did the objectives change over time? If so, how, when, why, and in what ways did they change?
   (g) Was the language eventually put to unexpected uses, or adopted by an unexpected user community? If so, what were the results? Did the language change further in response to these new uses? How might the language or even the initial objectives have been different if these uses had originally been anticipated?
2. **Contributions of Language**
   (a) What is the major contribution that was made by this language? Was this one of the objectives? Was this contribution a technical or a non-technical contribution, or both? What other important contributions are made by this language? Were these part of the defined objectives? Were these contributions technical or non-technical?

(b) What do you consider the best points of the language, even if they are not considered to be a contribution to the field (i.e., what are you proudest of, regardless of what anybody else thinks)?
(c) Are there now multiple independent implementations of the language?
(d) What other people or groups decided to implement this language because of its inherent value?
(e) Did this language have any effect (positive or negative) on the development of later hardware?
(f) Did this language have any effect (positive or negative) on the development of later languages? Have specific innovative features of this language appeared in later languages?
(g) Did this language spawn any "dialects"? If so, please identify them. Were they major or minor changes to the language definition? How significant did the dialects themselves become? Describe the resulting "family tree" of both languages and development communities.
(h) In what way do you feel the computer field is better off (or worse) for having this language?
(i) What fundamental effects on the future of language design resulted from this language development (e.g., theoretical discoveries, new data types, new control structures)?

3. **Mistakes or Desired Changes**
   (a) What mistakes do you think were made in the design of the language? Why do you consider them to be mistakes? Were any of these able to be corrected in a later version of the language? If you feel several mistakes were made, list as many as possible with some indication of the severity of each.
   (b) Even if not considered mistakes, what changes would you make if you could do it all over again?
   (c) What have been the biggest changes made to the language (possibly by people other than the original development team) since its early development? Were these changes or new capabilities considered originally and omitted in the initial development, or were they truly later thoughts? If they were originally considered and omitted, why were they omitted (e.g., desire for simplicity, consistency with overarching design principles, time or other resource constraints)? Were they omitted with the intent to reconsider them later?
   (d) Have changes been suggested but not adopted? If so, be as explicit as possible about changes suggested, and why they were not adopted.
4. **Problems**
   (a) What were the biggest problems you had during the language design process? Did these affect the end result significantly?
   (b) What are the biggest problems the users have had?
   (c) What are the biggest problems the implementors have had? Were these deliberate, in the sense that a conscious decision was made to do something in the language design, even if it made the implementation more difficult?
   (d) What are the biggest problems the documenters of the language have had? Would the language have been easier to document or to explain to users if the language had been designed differently?

(e) What trade-offs did you consciously make during the language design process? What trade-offs did you unconsciously make?
(f) What compromises did you have to make to meet other constraints such as time, budget, user demands, political, or other factors?

IV. IMPLICATIONS FOR CURRENT AND FUTURE LANGUAGES

These questions are directed to the ways in which your language or language project might affect its successors.

1. **Direct Influence**
   (a) Have other, more recent language designs been directly influenced by this one? Have other languages borrowed specific features, implementation techniques, specification techniques, documentation techniques, debugging techniques, or proof techniques from this language effort? Has any other language had interoperability with this language as one of its design goals?
   (b) What language developments of today and the foreseeable future are being directly influenced by your language? Regardless of whether your answer is "none" or "just these few…" or "many, such as…," please indicate the reasons.
   (c) Is there anything in the experience of your language development which should influence current and future languages? If so, what is it? Put another way, in light of your experience, do you have advice for current and future language designers? What are the important lessons learned from the development of this language? What are the important lessons from the user experience?
   (d) Does your language have a long-range future? Regardless of whether your answer is "yes" or "no," please indicate the reasons.
2. **Indirect Influence**
   (a) Are there indirect influences which your language is having now?
   (b) Are there any indirect influences that it can be expected to have in the near future? What are these, and why do you think they will be influential?

QUESTIONS FOR AUTHORS: LANGUAGE EVOLUTION

The principal objective of a contribution in this category is to treat the history of a major language subsequent to its original development. This may entail extending the history of some language whose origins were treated in an earlier HOPL conference.

When a programming language is first developed, it is typically the work of an individual or a small, concentrated group. Later development of a language is often the result of an expanded, re-staffed group, and perhaps additional individuals or groups outside the original organization. Similarly, while original work is often focused on language design and implementation for a single environment, later developments may be undertaken in a broader arena.

When compared with the Questions for Authors: Early History, the following questions reflect this change in context. In particular, these questions address the set of diverse development activities that may surround the language, such as standardization, new implementations, significant publications, and language-oriented groups of persons (such as ACM SIGs, user groups, and open-source communities). However, many of the questions here are similar or identical to those in the "early history" questions, as noted below.

These questions are grouped into the same four broad categories that apply to papers on the early history of a language:

(1) Background
(2) Rationale of content
(3) A posteriori evaluation
(4) Implications for current and future languages

The very first question applies broadly and is intended to identify the particular development activities that are the focus of the history paper. All subsequent questions, in all sections, should be considered for *each* of the activities identified in response to that first question.

You may also have significant new information to contribute regarding early history of the language, especially if such information is relevant to subsequent developments. If so, be sure to place such material in appropriate context, and carefully distinguish new information about early history from the history of subsequent development.

Where appropriate, your contribution should make reference to related papers from earlier HOPL conferences.

I. BACKGROUND

0. **Categories of Development and Activities**
   What triggered the developments reported on, and what major development activities resulted? (Examples of development activities are standardization, forking of specifications or implementations, creation of new implementations from specifications, research activities, creation of products from research prototypes, release as open source, significant publications such as specifications or tutorials, creation of formal or informal user groups, borrowing or adaptation from other languages, and creation or further development of competing languages. Note that activities may serve as triggers for other activities; for example, user requirements or requests might trigger standardization activities, which in turn might trigger new implementation activities. Similarly, release of open source or an unencumbered specification might trigger new implementation activity.)

From this point on, each question is intended to apply independently to each development activity identified by the preceding question. However, in the actual paper, it may be appropriate to group or consolidate answers that apply to multiple such activities.

1. **Motivation**
   Questions I.1(a)–(d) correspond to those in the "Early History" questions, Section I.1.
   (a) What (or who) was the real origin of the idea to further develop this language? (This question may have more than one good answer; the choice may shape the perspective of your paper. Be sure to consider whether there are multiple perspectives.)
   (b) Who was the prime mover for the evolutionary development effort?
   (c) What was the primary motivation to further develop the language (e.g., research, task assigned by management, turn a research prototype into a product, attract new users)?
   (d) Did the motivation change over time before the revised or further developed language was released? After the language was released?

   In addition:
   (e) In what ways did these motivations differ from the motivations for the original creation or development of the language?
2. **Languages or Systems Known at the Time**
   Questions I.2(a)–(i) correspond to those in the "Early History" questions, Section I.2.
   (a) What other specific languages were known to you and/or other members of the development group at the time the work started? Which other languages did any of you learn about as the evolutionary work progressed? Was such learning in intentional

pursuit of information to guide the evolutionary design or development of the new language? How much did you know about these languages and in what ways (e.g., as users, from having read unpublished and/or published papers, informal conversations, from documentation, from reading the code for the implementation, from having participated in the design or implementation)? Please try to distinguish between what you, the author, knew at the time and what the other members of the project knew at the time.

(b) Were these other languages regarded as formal inputs to consider in your own language development, or did they merely provide background? What was it about these languages that you wanted to emulate (e.g., syntax, capabilities, internal structure, application area, etc.)?

(c) How influenced were you by these languages? Put another way, how much did the prior language backgrounds of you and other members of the group influence the early language design? Whether the answer is "a lot" or "a little," why did these other languages have that level of influence?

(d) Was there a primary source of inspiration for further language development, and if so, what was it? Were specific evolutionary changes to the language modeled after this (or any other predecessors or prototypes)?

(e) How did information about each of these other languages or systems become available to the people it influenced?

(f) Were any specific new features borrowed from, inspired by, or created as an alternative to specific features in other languages or systems? Why?

(g) Were any proposed or plausible features omitted or removed in response to experience with other languages or systems? Why?

(h) Was the design, development, or deployment process modeled on that used for another language or system? Why?

(i) Was the design, development, or deployment process intentionally done differently from that used for another language or system? Why?

In addition:

(j) Was the design, development, or deployment process intentionally done differently from that used during the early history of this language because of experience with another language or system?

3. **Basic Facts About Activities, Organizations, and People**

Questions I.3(a)–(v) correspond to those in the "Early History" questions, Section I.3.

(a) Under what organizational auspices (if any) was the language further developed (e.g., name of company, department/division in the company, university, consortium or collaboration, open-source project, standardization effort)? Be as specific as possible about organizational units and subunits involved.

(b) What, if any, was the nature of any cooperation or competition among multiple organizations or organizational subunits?

(c) Were there problems or conflicts within the organization(s) in getting the project started? If so, please indicate what these were and how they were resolved.

(d) What was the source of funding (e.g., research grant, company R&D, company production units, government contract)?

(e) Who were the people on the project and what was their relationship to the author(s) (e.g., team members, subordinates, managers)? To the largest extent possible, name all the people involved, including part-timers, when each person joined the project,

and what each person worked on and when. Indicate the technical experience and background of each participant, including formal education. Please be as specific as possible regarding names, titles, and dates.

(f) How did the roles of various individuals change during the course of the activity?

(g) In what ways did geographic factors affect what was done, who participated, how they were organized, and how they communicated?

(h) In what ways did organizational or political factors affect what was done, who participated, how they were organized, and how they communicated?

(i) In what ways did computerized or networked facilities affect what was done, who participated, how they were organized, and how they communicated?

(j) In what ways did the availability (or lack) of other resources affect what was done, who participated, how they were organized, and how they communicated?

(k) Were the design and implementation done together, by a single person or team? Alternatively, were there separate design and implementation steps or phases, and if so, were the design and the implementation done by the same people or different (possibly overlapping) teams?

(l) Was the evolutionary development done originally as a research project, as a development project, as a committee effort, as an open-source effort, as a one-person effort with some minor assistance, or what? Was it a side effect of some other effort that happened to produce a revised language or implementation as a subgoal, but the language or implementation itself ended up being important in its own right?

(m) Was the implementation done originally as a research project, as a development project, as a committee effort, as an open-source effort, as a one-person effort with some minor assistance, or what?

(n) Was there a separate documentation team, or was documentation written by designers and/or implementors?

(o) Was testing ("quality assurance") done by a separate team? By the designers or implementors?

(p) Were there any persons whose roles were "marketing" or "user liaison" or "user training"? If so, who fulfilled such roles, and how?

(q) Was there a defined leader to the group? If so, what was his or her exact position (and title) and how did he or she get to be the leader (e.g., appointed "from above," self-starter, volunteer, elected)?

(r) Was there a *de facto* leader different from the defined leader? If so, who was this leader and what made this person the *de facto* leader (personality, background, experience, a "higher authority," something else)? Was one person the "manager" and another the "technical lead"? Or did more than one person fill either or these roles? Was technical leadership divided according to multiple design or implementation areas?

(s) Were there consultants from outside the project who had a formal connection to it? If so, who were they, how and why were they chosen, and how much help were they? Were there also informal consultants? If so, please answer the same questions.

(t) Did the participants in the project view themselves primarily as language designers, as implementors, or as eventual users? If there were some of each working on the project, indicate the split as much as possible. How did this internal view of the people involved affect the initial planning and organization of the project?

(u) Did the language designers know (or believe) that they would also have the responsibility for implementing the revised language? Whether the answer is "yes" or "no," was the technical language design affected by this?
(v) Did the language designers know (or believe) that they would also have the responsibility for maintaining subsequent versions? Whether the answer is "yes" or "no," was the technical language design affected by this?

In addition:

(w) How were the organizations involved in evolutionary developments related to the organizations that originally developed the language?
(x) How were the people involved in evolutionary developments related organizationally to the original developers for this language?

4. **Costs and Schedules**

Questions I.4(a)–(g) correspond to those in the "Early History" questions, Section I.4.

(a) Was there a budget? Was it adequate? Did the budget provide a fixed upper limit on the costs? If so, how much money was to be allocated and in what ways? What external or internal factors led to the budget constraints? Was the money formally divided between language design and actual implementation? If so, in what way?
(b) Was there a fixed deadline for completion of the project? Was the project divided into phases and did these have deadlines? How well were the deadlines met?
(c) What is the best estimate for the amount of human resources involved (i.e., in person-years)? How much was for language design, for documentation, for implementation, and for other functions?
(d) What is the best estimate of the costs prior to putting the first system in the hands of the first users? If possible, show as much breakdown on this as possible.
(e) If there were cost and/or schedule constraints, how did that affect the language design and in what ways?
(f) What other resource constraints affected the schedule?
(g) Did any external constraints (such as customer requirements or competition with another organization) influence the schedule?

5. **Basic Facts about Documentation**

Questions I.5(a)–(j) correspond to those in the "Early History" questions, Section I.5.

(a) What are the significant publications that arose from design, development, testing, and deployment? For each provide:
  (a1) Full bibliographic citation
  (a2) Names of authors (if not part of the reference)
  (a3) Intended audience (e.g., user, implementer, language researcher)
  (a4) Format (e.g., manual, general trade book, standards document, conference paper, web page, blog entry)
  (a5) Availability
(b) In the planning stage, was there consideration of the need for documentation of the work as it progressed? If so, was it for internal communication among project members or external monitoring of the project by others, or both?
(c) What types of documentation were decided upon?
(d) Did any sort of intentional specification precede implementation, or was an initial implementation followed by documentation?

(e) If documentation and implementation were found to be in conflict, was one generally considered more definitive than the other? If so, which one, and why? Did this change over time?

(f) Was a test suite developed to verify the implementation? If so, was it used to perform regression tests on the compiler or other parts of the implementation? Was the test suite made public? To what extent was the test suite considered part of the (internal or external) specification or documentation?

(g) To the largest extent possible, cite both dates and documents for the following (including internal papers and web sites which may not have been released outside of the project) by title, date, and author.
  (g1) Initial idea
  (g2) First documentation of initial idea
  (g3) Preliminary specifications
  (g4) "Final" specifications (i.e., those which were intended to be implemented)
  (g5) "Prototype" running (i.e., as thoroughly debugged as the state of the art permitted, but perhaps not all of the features included)
  (g6) "Full" language compiler (or interpreter) was running
  (g7) Usage on real problems done by the developers
  (g8) Usage on real problems done by people other than the developers
  (g9) Documentation by formal methods
  (g10) Paper(s) in professional journals or conference proceedings
  (g11) Please identify extensions, modifications and new versions
  (g12) Independent implementations by other groups or organizations

  For each of these stages, where relevant, indicate the level of formality of the specifications. Were specifications entirely in prose (in English or some other "natural language"), or were any formalisms used (examples include BNF, regular expressions or other ways of marking optional or repeated elements in the syntax, syntax diagrams (aka "railway charts"), attribute grammars, mathematical equations, algebraic specifications, denotational semantics, the Vienna definition language, compiler-construction tools such as parser generators, and proof assistants)? Were the formalisms directed primarily to the explanation of syntax, or was formalism also used in the explanation of semantics (that is, program behavior)?

(h) If there were independent implementations, did they rely entirely on specifications and documentation, or did they read or borrow from an existing code base and/or communicate directly with the original development team?

(i) Please reflect on the project documents as historical sources. Throughout your paper, identify when you are relying on these documents for information, and when you are relying on memory or other sources. Are there places where memory and documentation conflict? Do you now regard the available documentation as a complete and faithful record of the history, or can you identify gaps or incorrect information in the documents?

(j) It may be appropriate to reproduce a limited number of original documents or photographs in your paper. Which ones are the most relevant for telling the story?

In addition:

(k) Which documents were revised or superseded as a result of the evolutionary development, and which documents were unchanged?

(l) Was it a stated principle of (or constraint on) the evolutionary development that certain documents remain unchanged?

6. **Intended Purposes and Users**

Questions I.6(a)–(e) correspond to those in the "Early History" questions, Section I.6.

(a) For what application area were evolutionary changes to the language or its implementation intended? What different or additional types of problems was it intended to be used for? Traditional labels such as "business data processing" or "scientific applications" may be too vague; please be as specific as possible and appropriate in describing the application area.
  (a1) Was a specific statement of application area or purpose drafted when the project started? If so, did it change over the course of the project?
  (a2) Was the apparent application area of some other language used as a model?

(b) For what types of users was the revised language intended (e.g., experienced programmers, mathematicians, business people, novice programmers, non-programmers)? Was there any conflict within the group on this? Were compromises made, and if so, were they made for technical or non-technical reasons?

(c) Why revise the language? Why was the existing language, or other languages deemed sufficiently inadequate for the purpose that a new effort was justified?
  (c1) Was the revisions developed specifically in response to a stated new application area or purpose?
  (c2) Was the intent to create a language "just like the old one, except for thus-and-so specific incremental improvements"?
  (c3) Was the intent to create a synthesis by incorporating ideas from other languages?
  (c4) Was the language revised in order to exemplify or promulgate a specific theoretical idea?
  (c5) Was the revision initially the result of "just hacking around," perhaps by grafting a new idea or two onto the existing language?

(d) How much of the language revision effort went into application-specific features, and how much of it went into "structural issues" or "software engineering issues" such as modularity, namespace management, debugging support, or features coders of libraries might need as opposed to application programmers?

(e) Were revisions driven by the requirements of porting the language to new target architectures? Wherever possible, cite specific machine(s) by manufacturer(s) and model numbers, or alternatively, give broad descriptions relevant to the time period with specific examples.

In addition:

(f) How did the intended set of applications differ from the originally intended set of applications for this language?

(g) How did the intended set of users differ from the originally intended set of users for this language?

(h) Was increased machine independence a significant design goal of the language revision? Alternatively, was better access to or exploitation of machine-dependent facilities a significant design goal of the language revision?

7. **Distribution**
   Questions I.7(a)–(d) correspond to those in the "Early History" questions, Section I.7.
   (a) Was the revised language specification (information needed to create a complete implementation) made available to users? Was it considered proprietary? Was it provided for free or for sale? What distribution media were used?
   (b) How was the revised language implementation made available to users? Was it considered proprietary? Was it provided for free, for lease or subscription, or for sale? What distribution media were used? Were users able to run it on their own hardware, or only as part of a service on hardware they did not own?
   (c) How was the revised language documentation (information needed to use the language, which may or may not include the specification and may or may not include tutorial material) made available to users? Was it considered proprietary? Was it provided for free or for sale? What distribution media were used?
   (d) Did the answers to any of the preceding questions change over time?

II. RATIONALE OF THE CONTENT OF THE DEVELOPMENT

1. **Environmental Factors**
   Questions II.1(a)–(p) correspond to those in the "Early History" questions, Section II.1.
   To what extent was the revision of the language influenced by:
   (a) Program Size: Was it explicitly thought that programs written in the language would be large and/or written by more than one programmer? Or, conversely, was it explicitly thought that typical programs would be small and/or written primarily by one programmer? What features were explicitly included (or excluded) for this reason? If this factor wasn't considered, did it turn out to be a mistake? Were specific tools or development environments designed at the same time to support these choices?
   (b) Program Libraries and/or Frameworks: Were "program libraries" (or "standard frameworks") envisioned as necessary or desirable, and if so, how much provision was made for them? Were "standard libraries" a major portion of the language design effort? If so, what libraries were especially important? Which libraries were novel, unusual, or essential to the "flavor" of the language? Alternatively, was it an important design goal of the language to be able to interoperate with existing libraries already coded in another language?
   (c) Portability: How important was the goal of machine independence? What revisions reflect concern for portability? How well was this goal attained?
   (d) User Background and Training: What revisions catered to the expected background of intended users?
      (d1) Were any notational or semantic features of the language intentionally modeled on notations or semantics not widely used in computer science but customary in an intended domain of application?
      (d2) How difficult did it prove to train users in the correct and effective use of the language, and was the difficulty a surprise? What changes in the language would have alleviated training problems? Were any proposed features rejected because it was felt users would not be able to use them correctly or appropriately?
      (d3) In retrospect, what features of the language proved to be difficult for programmers to use correctly? Did some features fall into disuse? Please identify such features and explain why they fell into disuse.

(e) Execution Efficiency: How did requirements for executable code size and speed affect the language revision? Were programs in the language expected to execute on large or small computers (i.e., was the size of object programs and/or execution speed expected to pose a problem)? What design decisions were explicitly motivated by the concern (or lack of concern) for execution efficiency? Did these concerns turn out to be accurate? How was the design of specific features changed to make it easier to optimize executable code?
(f) Target Computer Architecture: To what extent were revisions in the language dictated by the characteristics of the anticipated target computer (e.g., word size, floating-point hardware, instruction set peculiarities, application-specific instructions, register or cache structure, special-purpose co-processors and accelerators, parallelism, synchronization features, transactional memory)? Were there multiple targets, and if so, in what way did this affect the language design or implementation?
(g) Compilation Environment: To what extent, if any, did concerns about compilation efficiency affect the revision? Were features rejected or included primarily to make it easier to implement compilers for the language or to ensure that the compiler(s) would execute quickly? In retrospect, how correct or incorrect do you feel these decisions were? What decisions did you make regarding use of the compiler run-time system?
(h) Programming Ease: To what extent was the ease of coding an important consideration and what revisions to the language reflect the relative importance of this goal? Did maintainability considerations affect any design decisions? If so, which ones?
(i) Execution Environment: To what extent did the language design reflect its anticipated use in a batch, timeshared, embedded, portable, mobile, wearable, kiosk, office, or networked environment? What revisions reflect these concerns?
(j) Multiple Implementations: Were there multiple implementations being developed at the same time as the later part of the language development? If so, was the language design hampered, improved, or influenced by this in any way?
(k) Standardization: In addition to (or possibly separate from) the issue of portability, what considerations were given to possible standardization? What types of standardization were considered, and what groups were involved and when?
(l) Networking/Parallel Environment: To what extent did the language revision reflect its anticipated use in a networked- or parallel-execution environment? What features reflect these concerns? Was execution efficiency in a networked or parallel environment (as opposed to single-CPU efficiency) a concern?
(m) Interoperability: Was it important for programs written in this language to interoperate with code written in other languages, operating systems, or other tools? If so, which ones, and why? Did this require specific technical changes to the language?
(n) Documentation: Were features included to specifically to support documentation of programs coded in the language? If so, did they support documentation tied to the program representation, generation of separate documents (paper documents, files, web pages), or both? What tools and conventions were used or supported?
(o) Reliability: Were features included specifically to support detection or prevention of programming errors and/or proofs of program correctness? Were otherwise common or familiar features omitted in order to help prevent programming errors?
(p) Debugging: Were features included specifically to support the debugging process?

In addition:

(q) In what ways had the environment changed since the original development of the language? In what ways did such changes motivate or influence new developments?

2. **Functions to be Programmed**

Questions II.2(a)–(e) correspond to those in the "Early History" questions, Section II.2.

(a) How did the operations and data types in the revised language support the writing of particular kinds of algorithms or applications?

(b) What revisions might have been left out, if a slightly different application area had been in mind?

(c) What revisions were considered essential to properly express the kinds of programs to be written?

(d) What misconceptions about application requirements turned up that necessitated redesign of application-specific features before the revised language was actually released?

(e) To what extent were application-specific features incorporated directly into the language design, and to what extent was the expectation that application-specific features would be provided (perhaps as libraries) by coding in the language itself? Were any features of the language specifically intended to support or enable the latter?

In addition:

(f) Was the language changed or extended in order to better support existing classes of applications or users?

(g) Was the language changed or extended in order to support new classes of applications or users?

(h) What features were considered essential to meet intended applications?

(i) What features were considered "nice" or "convenient" but not essential?

(j) Which changes were "upward compatible" and which were "incompatible"?

(k) How were tradeoffs evaluated? Who evaluated the tradeoffs? Who made the decisions?

3. **Language Design Principles**

Questions II.3(a)–(j) correspond to those in the "Early History" questions, Section II.3.

(a) What consideration, if any, was given to revising the language so that programming errors could be detected earlier and more easily? Were the problems of debugging and testing considered? Were debugging and testing facilities deliberately included in the language?

(b) To what extent was the goal of keeping the language simple considered important? What kind of simplicity was considered most important? What did your group mean by "simplicity"? Was there an explicit statement of simplicity that guided the revision effort?

(c) To what extent was the goal of keeping the language consistent considered important? What kind of consistency was considered most important? What did your group mean by "consistency"? Was there an explicit statement of consistency that guided the revision effort?

(d) Were there any other explicitly stated principles that guided the language revision? If so, were any specific proposed features modified or omitted after coming into conflict with a stated principle?

(e) What thought was given to make programs more understandable and how did these considerations influence the design? Was there conscious consideration of making programs "easy to read" versus "easy to write"? If so, which were chosen and why?

(f) Did you consciously develop the data types first and then the operations, or did you use the opposite order, or did you try to develop both in parallel with appropriate iteration? Were data and operations combined into objects?
(g) To what extent did the revision reflect a conscious philosophy of how languages should be designed or revised (or how programs should be developed)? What was this philosophy?
(h) Did you intentionally model the revised language according to an existing style or school of thought (e.g., Algol-like, Lisp-like, macro-like, imperative, declarative, block-structured, object-oriented, functional, pure, or stack-based)? If so, why? In what ways does your resulting design reflect this style or school of thought, and in what ways does it differ, and why?
(i) Were any slogans or catchphrases used as summaries or reminders of language design principles? Were they used internally or also externally? Were they used seriously or jocularly?
(j) Did the developers of the revised language make any attempt to evaluate specific features, or the entire language design, objectively/empirically? Did the results lead to changes in the language design? Did such changes result in programs that were somehow better (e.g., shorter, more reliable, more efficient, easier to read, easier to maintain)?

In addition:

(k) What language design principles from the original development were preserved? Of these, which were considered important?
(l) What language design principles from the original development were intentionally modified, discarded, or replaced?
(m) What language design principles from the original development were unintentionally violated, disrupted, negated, or abandoned?
(n) Was it a stated principle of (or constraint on) the evolutionary design effort that certain code bases remain unchanged?

4. **Language Specification**

Questions II.4(a)–(c) correspond to those in the "Early History" questions, Section II.4.

(a) What techniques for specifying languages were known to you? Did you use these or modify them, or did you develop new ones?
(b) To what extent (if any) was the revised language itself influenced by the technique used for the specification?
(c) What formalisms and/or tools were used in the specification process?

In addition:

(d) What language specification principles, tools, and techniques were used in this new development? To what extent was the result of the development influenced by these choices?
(e) Was the language specification revised to produce a single new comprehensive specification? Alternatively, was an incremental addendum produced?
(f) What language specification principles, tools, or techniques from the original development were preserved? Of these, which were considered important?
(g) What language specification principles, tools, or techniques from the original development were intentionally modified, discarded, or replaced?
(h) What language specification principles, tools, or techniques from the original development were unintentionally violated, disrupted, negated, or abandoned?

5. **Concepts about Other Languages**
   Questions II.5(a)–(c) correspond to those in the "Early History" questions, Section II.5.
   (a) Were you consciously trying to introduce new concepts? If so, what were they? Do you feel that you succeeded? (Conversely, was it a design goal to *avoid* introducing new concepts? If so, why?)
   (b) If you were not trying to introduce new concepts, what was the justification for revising the language? (Such justification might involve technical, personal, political, or economic factors.)
   (c) To what extent did the design consciously borrow from previous language designs or attempt to correct perceived mistakes in other languages?

   In addition:
   (d) To what extent was the introduction of new language concepts or features a part (or an explicit goal) of this development?
   (e) To what extent was the development influenced by a perceived need to compete with other languages or other development efforts?
6. **Influence of Non-technical Factors**
   Questions II.6(a)–(f) correspond to those in the "Early History" questions, Section II.6.
   (a) How did time and cost constraints (as described in the Background section) influence the technical design?
   (b) How did the size and structure of the design group affect the technical design?
   (c) How did the size and structure of the implementation group affect the implementation?
   (d) How did the size and structure of the documentation group affect the documentation?
   (e) Did the membership of any of these groups change over time, and if so, how did such changes affect the design, implementation, or documentation efforts?
   (f) Provide any other information you have pertaining to ways in which the technical language design was influenced or affected by non-technical factors.

   In addition:
   (g) What non-technical factors were different for the new development as compared to the original language development?
   (h) How did differences or changes in non-technical factors affect the new development?

III. A POSTERIORI EVALUATION

Sections III.1–4 correspond to those in the "Early History" questions, Sections III.1–4.

These questions are directed to assessing the language project from today's standpoint.

1. **Meeting of Objectives**
   (a) How well do you think the revised language met the original objectives for the revision?
   (b) What is the size and nature of the current user community? Do the users think the language has met its objectives? Do the users think the revised language has met their needs? If not, what do users think could be done, or should have been done, to further improve the language?
   (c) How well do you think the computing community (as a whole) thinks the objectives were met?
   (d) How much impact did portability (i.e., machine independence) have on acceptance by users?
   (e) How much impact did other specific aspects of the language (such as execution efficiency, compilation efficiency, ease of programming, or specific libraries) have on acceptance by users?

(f) Did the objectives change over time? If so, how, when, why, and in what ways did they change?
(g) Was the revised language eventually put to unexpected uses, or adopted by an unexpected user community? If so, what were the results? Did the language change further in response to these new uses? How might the language or even the initial objectives have been different if these uses had originally been anticipated?

2. **Contributions of Language**
   (a) What is the major contribution that was made by this language revision? Was this one of the objectives? Was this contribution a technical or a non-technical contribution, or both? What other important contributions are made by this language? Were these part of the defined objectives? Were these contributions technical or non-technical?
   (b) What do you consider the best points of the language, even if they are not considered to be a contribution to the field (i.e., what are you proudest of, regardless of what anybody else thinks)?
   (c) Are there now multiple independent implementations of the language?
   (d) What other people or groups decided to implement this revised language because of its inherent value?
   (e) Did this revised language have any effect (positive or negative) on the development of later hardware?
   (f) Did this revised language have any effect (positive or negative) on the development of later languages? Have specific innovative features of this revision appeared in later languages?
   (g) Did this language spawn any "dialects"? If so, please identify them. Were they major or minor changes to the language definition? How significant did the dialects themselves become? Describe the resulting "family tree" of both languages and development communities.
   (h) In what way do you feel the computer field is better off (or worse) for having this revised language?
   (i) What fundamental effects on the future of language design resulted from this language development (e.g., theoretical discoveries, new data types, new control structures)?
3. **Mistakes or Desired Changes**
   (a) What mistakes do you think were made in the revision of the language? Why do you consider them to be mistakes? Were any of these able to be corrected in a yet later version of the language? If you feel several mistakes were made, list as many as possible with some indication of the severity of each.
   (b) Even if not considered mistakes, what changes would you make if you could do it all over again?
   (c) What have been the biggest changes made to the language (possibly by people other than the original development team) since its early development? Were these changes or new capabilities considered originally and omitted in the initial development, or were they truly later thoughts? If they were originally considered and omitted, why were they omitted (e.g., desire for simplicity, consistency with overarching design principles, time or other resource constraints)? Were they omitted with the intent to reconsider them later?
   (d) Have changes been suggested but not adopted? If so, be as explicit as possible about changes suggested, and why they were not adopted.

4. **Problems**
   (a) What were the biggest problems you had during the language revision process? Did these affect the end result significantly?
   (b) What are the biggest problems the users have had?
   (c) What are the biggest problems the implementors have had? Were these deliberate, in the sense that a conscious decision was made to do something in the language design, even if it made the implementation more difficult?
   (d) What are the biggest problems the documenters of the language have had? Would the language have been easier to document or to explain to users if the language had been designed differently?
   (e) What trade-offs did you consciously make during the language design process? What trade-offs did you unconsciously make?
   (f) What compromises did you have to make to meet other constraints such as time, budget, user demands, political, or other factors?

In addition:

5. **Supersession and Cohesion**
   (a) Has the original version of the language been superseded, or do the original and revised versions coexist with a substantial set of users for each? Why?
   (b) Is there now a single revised version of the language, or has language evolution resulted in a diversity of dialects or incompatible implementations? Why?

IV. IMPLICATIONS FOR CURRENT AND FUTURE LANGUAGES

Sections IV.1–2 correspond to those in the "Early History" questions, Sections IV.1–2.

These questions are directed to the ways in which your language or language project might affect its successors.

1. **Direct Influence**
   (a) Have other, more recent language designs been directly influenced by this language revision? Have other languages borrowed specific features, implementation techniques, specification techniques, documentation techniques, debugging techniques, or proof techniques from this language effort? Has any other language had interoperability with this language as one of its design goals?
   (b) What language developments of today and the foreseeable future are being directly influenced by your language? Regardless of whether your answer is "none" or "just these few…" or "many, such as…," please indicate the reasons.
   (c) Is there anything in the experience of your evolutionary language development which should influence current and future languages? If so, what is it? Put another way, in light of your experience, do you have advice for current and future language designers? What are the important lessons learned from the revision of this language? What are the important lessons from the user experience?
   (d) Does your language have a long-range future? Regardless of whether your answer is "yes" or "no," please indicate the reasons.
2. **Indirect Influence**
   (a) Are there indirect influences which your language is having now?
   (b) Are there any indirect influences that it can be expected to have in the near future? What are these, and why do you think they will be influential?

QUESTIONS FOR AUTHORS: FEATURE OR CONCEPT

The principal objective of a contribution in this category is to record the history of a programming language feature or concept.

This question set has a quite different focus from the set dealing with the development of some particular language. A particular language feature or concept is incorporated in multiple particular languages; therefore much of the information solicited here is comparative. However, to the extent that you were party to the development of particular language(s) embodying this feature, it may be appropriate also to be guided by the Questions for Authors: Early History (of a single language).

For simplicity, these questions usually refer to a "feature" rather than a "feature or concept."

1 **Origins**
   (a) Who invented or introduced this feature? Was it a single person, a team, or a committee? What was their background? How did they come to use this background in their development?
   (b) What are the important publications (e.g., papers, language manuals) about the feature? Please make a special point of identifying as many early publications as possible, including informal and internal documents.
   (c) If the feature first appeared in a simpler or even "half-baked" form before the widespread form emerged, be sure to describe the early forms, with citations, and how and why they evolved. What factors caused the feature to reach a "definitive" form (e.g., perceived technical merit, feedback from early or prospective users, institutional pressures)?
   (d) Was the creation or implementation of the feature motivated more by theory or by practice?
   (e) Did the feature emerge as the result of a research project? A corporate or industrial project? A standardization process? "Just hacking around"?
   (f) Was there a stated rationale or set of goals for the feature?
   (g) Was there a stated set of design principles for the feature?
   (h) Was the feature first created especially for a single language (from which it then presumably spread to other languages), or did its creator(s) originally have in mind a class of languages (e.g., Algol-like, functional, object-oriented, parallel)?
   (i) In which language(s) was this feature first introduced? Who introduced it? How did the people involved with it interact in the process, if more than one person was involved?
   (k) Was a new language created specifically to embody or exemplify this new feature, or was its first implementation or embodiment grafted into an existing language, or was it one of a number of innovations in a new language?
   (l) Please consult the Background section of the Questions for Authors: Early History for questions about project organization, project funding, documentation, and distribution.
2. **The Nature of This Feature or Concept**
   (a) What are the distinctive characteristics of this feature? Be sure to discuss both syntax and semantics, and what aspects are essential and which are nonessential.
   (b) Was the feature intended to express an idea previously very difficult or impossible to express in existing languages? Alternatively, was it intended to make an already expressible idea more convenient, more concise, more abstract, more concrete, more specific, more reliable, or more maintainable?
   (c) What shortcomings of existing languages prompted the discovery, invention, and/or introduction of this feature?

(d) What set of expressive or methodological problems did this feature address at the time of its introduction? How was the absence of this feature handled prior to its introduction?
  (d1) Does the feature address some specific application area?
  (d2) Does the feature address a problem related to program organization?
  (d3) Does the feature address a problem related to compile-time processing?
  (d4) Does the feature address a problem related to run-time processing?
  (d5) Does the feature address a problem related to a specific hardware architecture or class of architectures?
  (d6) Does the feature address a problem related to operating systems, input/output, user interfaces, or networking?
  (d7) Does the feature address a problem related to documentation?
  (d8) Does the feature address a problem related to debugging?
  (d9) Does the feature address a problem related to ensuring (or increasing the likelihood of) program correctness?
  (d10) Does the feature make programs easier to write? To read? To compile? To execute? To verify?
  (d11) To what extent did the feature support the coding of libraries (as opposed to end applications)?
(e) How did this feature enhance a programmer's ability to express the solution of a problem?
(f) What forces (e.g., theoretical concerns, market demand, error-proneness of alternative forms) supported the introduction and evolution of the feature?
(g) What external influences (e.g., changes in technology, new classes of problems, new classes of users) contributed to the need for this feature, or to its introduction and evolution?
(h) What other features or concepts were precursors of this one? In which language(s) did they exist?
(i) Was the feature intended to extend, supersede, provide an alternative to, or complement an already existing feature or concept?
(j) Is the concept of a "negative" form that perhaps manifests itself as a *lack* of language features (e.g., there are no 'goto' statements, there are no global variables, there are no side effects, multiple processes or threads cannot share data structures or pointers, the programmer cannot explicitly deallocate data structures)? What are the positive and negative consequences of such restrictions?
  (j1) Did this concept require the creation of new complementary features, or the adoption of already existing features from other languages, to compensate for this lack?
  (j2) Are such compensating features essentially the same in all languages that embody the concept, or did different languages end up with different sets of compensating feature? Why?
(k) To what extent did implementation of the feature require changes to a compiler (e.g., intermediate code representations, code analyses, phase structure)? To the structure of compiler output (such "binary files")? To linkers, loaders, debuggers, or other tools? To run-time organization (e.g., code, data, register conventions, stack organization, heap organization)? To libraries?

(l) To what extent does the presence of the feature make interoperability another language easier or more difficult? Does the answer to this question depend on whether that other language does or does not also support or provide the feature?

3. **Use and Subsequent Evolution**
   (a) To what extent was the feature used in practice? Did its actual usage correspond to the expectations of its designers (in terms of both amount of use and nature of use)? Did experience from use lead to changes in the nature of the feature in the same language, or in subsequent languages?
   (b) How did this feature evolve over time? Into what other forms has the feature evolved in languages currently in use?
   (c) What language(s) have embodied or now embody this feature? (Please be as inclusive as you can. However, it may be appropriate to concentrate on a few exemplars when making comparisons.)
      (c1) How does the feature manifest itself in each language? Give examples.
      (c2) In what ways are these languages similar or different in their incorporation of the feature?
      (c3) Why do languages differ in the ways they incorporate the feature? Do such differences cause any problems for users?
   (d) How did this feature interact with other features in the languages which embody it? How has this interaction influenced the evolution of this or other features? With what other language features is it most or least compatible? Are conflicts primarily syntactic or semantic in nature?
   (e) What obstacles (institutional, financial, psychological, political, temporal, technological, etc.) were encountered during the introduction and evolution of this feature? How did they affect it? How were they overcome?
   (f) What is the present incarnation of the feature, if any? In which languages can it be found today?
   (g) Were there competing or complementary features or concepts that conflicted with this one? If so, what was the nature of this conflict, and how was it resolved?
   (h) Has the feature itself evolved into distinct (and possibly competing) forms? Which languages exemplify these distinct forms? Why did differing forms arise?
   (i) Did the developers or users of the feature make any attempt to evaluate the feature objectively/empirically? Did the results lead to changes in the feature? Did such changes result in programs that were somehow better (e.g., shorter, more reliable, more efficient, easier to read, easier to maintain)?

4. **A Posteriori Evaluation**

   These questions are directed to assessing the feature or concept from today's standpoint.
   (a) How well did the feature meet (or fail to meet) its expectations and goals?
      (a1) Is the feature still available today in one or more widely used langages?
      (a2) Is the feature still in widespread use today by programmers? If not, was the feature eventually superseded by yet another idea, or was it simply abandoned?
      (a3) If the feature failed, why did it fail?
   (b) How did specific implementations affect its acceptance?
   (c) How was the feature received by those affected by it (e.g., programmers, problem solvers, documenters, managers)?

(d) What were the biggest problems that arose during the design or creation of the feature? How were they addressed? Did these problems affect the end result significantly?
(e) What were the biggest problems that arose during further evolution of the feature? How were they addressed?
(f) If languages have differed in how they incorporate the feature, have such differences caused any problems for users? If so, have these problems been addressed? In what manner? Has addressing these problems resulted in further changes to the feature?
(g) What are the biggest problems the users have had in understanding or using the feature? Does using the feature well require a significant amount of education, training, or experience?
(h) What are the biggest problems the implementors have had? Were these deliberate, in the sense that a conscious decision was made to implement the feature "elegantly" or "completely," even if it made the implementation more difficult?
(i) Conversely, was a conscious decision made to implement the feature in "quick and dirty" manner or "incompletely," in order to avoid certain implementation difficulties? Has this decision been made the same way in all languages that embody the feature, or have languages differed in how completely they implement the feature? Has the answer to this question changed over time?
(j) What are the biggest problems the documenters of the feature have had? Would the language have been easier to document or to explain to users if the feature had been designed differently?
(k) What other trade-offs were made during the feature design and implementation process? Were they conscious or unconscious?
(l) What compromises were made to meet other constraints such as time, budget, user demands, political, or other factors?
(m) If there are competing or complementary features or concepts that conflicted with this one:
    (m1) Has the feature effectively displaced or even rendered obsolete some other feature?
    (m2) Does the feature now coexist in the same language(s) with some competing or alternative feature? If so, in which situations may its use be considered preferable, and in which situations might an alternative be preferred?
    (m3) Does the feature now exist in a language that competes with other languages that do not have the feature? If so, in which situations may the availability of the feature make use of its language preferable, and in which situations might an alternative language be preferred?
(n) What other features or concepts were influenced by this one? Were other features or concepts later created specifically to complement, build on, or supersede this one?
(o) How did this feature contribute to software methodology?

QUESTIONS FOR AUTHORS: CLASS OF LANGUAGES

The principal objective of a contribution in this category is to record the history of a class of languages. A class of languages is defined either by an application area for which they were intended (e.g., simulation, graphics, machine-tool control, scripting, interactive games) or by a language paradigm (e.g., functional, data-flow, object-oriented, constraint satisfaction, higher-order array operators).

The principal objective of a contribution in this category is to record the history of a class of languages. A class of languages is defined by a coherent set of unifying aspects or design criteria, such as:

- an intended application area (e.g., simulation, graphics, machine-tool control, scripting, interactive games, finance, database access, machine learning)
- a language paradigm (e.g., imperative, functional, data-flow, object-oriented, data parallel, logic programming, constraint satisfaction, higher-order array operators)
- distinctive characteristics (e.g., reflection, dynamic types, ownership types, two-dimensional notation, non-textual presentation and/or gestures, elaborate macro mechanisms, pattern matching, a specific approach to storage management, features that support verification of program correctness)

This question set has a quite different focus from the set dealing with the development of some particular language. Here the focus is on the distinctive characteristics of a set of languages; therefore much of the information solicited here is comparative. However, to the extent that you were party to the development of particular language(s) embodying this feature, it may be appropriate also to be guided by the Questions for Authors: Early History.

The Questions for Authors: Language Evolution should be consulted in any case.

A particular language paradigm or application area often implies particular programming language features, or particular restrictions or extensions of language features. Please consult the Questions for Authors: Feature or Concept for additional guidance.

1. **Basic Information**
   (a) What characterizes this class of languages (e.g., language paradigm or application area)?
   (b) Which languages are in this class? (Please be as comprehensive as possible, including early experimental languages.)
   (c) Did the class begin with a single language that then inspired all the others, or was there some degree of independent initial invention?
   (d) Who were the initial creator(s), what was their background, and how did they come to be involved in this class of languages?
   (e) When and by whom was this set of languages (or some initial subset) perceived as being a distinct class?
   (f) How (and why) did this class of languages grow and/or evolve after initial recognition as a class?
   (g) What are the basic publications (e.g., conference papers, technical reports, language manuals, books, websites) about this language class or particular languages within this class? Please make a special point of identifying as many of the early publications as possible, including informal or internal documents.
2. **The Nature and Origin of This Class of Languages**
   (a) What are the distinctive attributes of this class of languages? How did these attributes affect the features that the designers of these languages included or excluded?

(b) What, if any, class of algorithms inspired the emergence of this language class? What application area(s) were these languages intended to address? How were these applications dealt with prior to the introduction of languages in this class?
(c) What expectations and goals did the community interested in this class of languages have in mind?
(d) What were the earliest languages in the class? How did they influence later languages in the class?
(e) Describe the relevant characteristics of languages the class comprises. (Please discuss as many languages as you can. However, it may be appropriate to concentrate on a few exemplars.) In what ways are languages in the class similar? How do they differ? In particular, compare them with respect to:
    (e1) declarative constructs (types, procedure and function headers)
    (e2) imperative constructs (statements, assignments)
    (e3) flow of control (conditionals, loops, procedure and function calls, nonlocal transfer of control)
    (e4) program modularization constructs
    (e5) data encapsulation constructs
    (e6) concurrency constructs
    (e7) communication constructs
    (e8) nesting of constructs, or prohibition of nesting
    (e9) scope and extent of names
    (e10) memory management (allocation, deallocation, reclamation)
    (e11) processor management (thread creation, task assignment)
    (e12) memory hierarchy and volatility
    (e13) reflective capabilities
    (e14) environmental interactions (time of day, inquiries about execution architecture)
    (e15) input-output
    (e16) interoperability with other languages in the class
    (e17) interoperability with other languages outside the class
    (e18) machine-dependent behaviors
    (e19) undefined behaviors
    (e20) other language-specific definitions, gestures, or behaviors?
(f) Describe the ways in which the languages in the class similar or different with respect to meeting the goals of:
    (f1) readability
    (f2) writability
    (f3) maintainability
    (f4) reliability
    (f5) portability
    (f6) compile-time efficiency
    (f7) run-time efficiency
    (f8) testability
    (f9) debuggability
    (f10) teachability and explainability
(g) Are there special editors or IDEs or other tools that accommodate many or all languages in the class? Alternatively, is the availability of a good editor or IDE or associated tool a feature that distinguishes certain members of the class from the others?

3. **Evolution**
   (a) How did this class of languages evolve over time? How did later languages in the class differ from earlier languages? What were the reasons for the differences? To what extent was upward compatibility a constraint? In what ways can the later languages be considered superior or inferior to the earlier ones? (A later language could, of course, be superior in some respects and inferior in others: it may be useful to treat the nature of these trade-offs. Consider Tony Hoare's remark about Algol 60 in "Hints on Programming Language Design" (1973): "The more I ponder the principles of language design, and the techniques which put them into practice, the more is my amazement and admiration of ALGOL 60. Here is a language so far ahead of its time, that it was not only an improvement on its predecessors, but also on nearly all its successors.")
   (b) To what extent did the languages in this class reflect external trends in programming language methodology (e.g., extension facilities, support for object-oriented programming, data abstraction, structured programming, pair programming, design patterns, agile programming)?
   (c) To what extent did the languages in this class reflect external trends or developments in computer hardware and operating systems (e.g., special-purpose hardware, availability of cheap processors and memory, multiprocessing, networking)?
   (d) Trace lines of influence among languages in the class, as well as comparative influence or lack of influence from languages outside the class. Describe the resulting "family tree" of both languages and development communities.
4. **Evaluation**
   (a) How well has this class of languages met (or failed to meet) its expectations and goals?
   (b) Has this class of languages proved to have applications beyond its original focus?
   (c) In the application area(s) relevant to the class, what are the competing languages (e.g., general purpose languages with special libraries) outside the class? How successful are the languages in the class versus these competing languages?
   (d) How did the languages in this class influence trends in programming language methodology?
   (e) How did this class of languages influence trends or developments in computer hardware and operating systems?
   (f) How has this class of languages influenced languages outside the class (e.g., by the latter incorporating characteristic features without adopting the underlying paradigm)?
   (g) Have ideas from this class of languages been adopted by more "general-purpose" languages?
   (h) Which early members of this language class have been superseded by later ones? Which continue to be used widely despite the emergence of later languages in the same class? Why?

Author Index