HOPL IV Reviewing Principles

Richard P. Gabriel, Mark Priestley, Guy L. Steele Jr

The *History of Programming Languages* conference (HOPL) is unlike many of the other SIGPLAN conferences. For one thing, it is held only every 12–15 years—there have been three so far: HOPL I in 1978, HOPL II in 1993, and HOPL III in 2007. For another, HOPL papers are not like ordinary research papers in the programming language area—they are oral histories, historical investigations, origin stories, and explorations. They really have more in common with essays than with research papers. Therefore, they are not reviewed by the same standards as research papers. What's even more confusing is that each submission should be reviewed according to its own quirks and characteristics. This guide should help.

Overview

A good HOPL paper is significant, relevant, accurate, and clear; it is neither novel nor original in the usual senses of those terms. In some cases, a good HOPL paper will be written from a first-person point of view; it can include opinions and unsubstantiated assumptions—for example, if those opinions and assumptions formed the basis for language design decisions. Conversations might be reported that have never been recorded.[1] The writing must be clear and generally scholarly, but it can be informal.

There are major complications: HOPL authors are not historians, they have been given few guidelines on how to write a HOPL paper, and there are no length constraints. Moreover, for the first round of reviews you will be looking at papers that range from "work in progress" to "done." Unlike many other conferences, HOPL is not a competition for scarce resources. There is no maximum number of accepted papers, nor is there an "acceptance rate" target. What we are looking for are thorough, valuable contributions to the historical record of programming language work.

Because we are looking for thorough papers on the history of programming languages, the lead time is long for authors. The initial call went out in Autumn 2017, the deadline for drafts was late Summer 2018, and camera ready is due in late Winter 2020. The first round of reviewing is likely to be of works in progress where the reviewer's job is to determine whether a good piece on history lurks in the manuscript and that the author is likely to be able to turn that into a good paper given another year to work on it.

Some reviewers might not be accustomed to reviewing papers like these. This document provides some principles for reviewing HOPL papers. HOPL papers will form a foundation of original documents for future historians, scholars, and researchers, and we want to get their reviews right.

[1] Things reported that were never recorded must be annotated as such.

Reviewing a HOPL paper is not like reviewing a computer science paper. Those papers have an established set of reviewing criteria having to do with novelty, originality, validation, and relevance. A good HOPL paper does not have to present novel technical results. Within reason, technical errors are not grounds for rejecting a paper.

Many HOPL authors—language designers, for example—are by definition the experts in their field. We want to hear their stories, and the primary job of reviewing is to enable those stories to be told in ways that will interest, inform, and inspire readers. A HOPL paper is not simply a gloss on a language definition. While (some) historians might be interested in the technical features of a language, historical accounts of language developments should be aiming to do something different.

A common view is that the job of history is primarily to establish "what happened," with the corollary that historical writing is a narrative of certain events, as comprehensive and as accurate as the writer can make it. The work of locating and interpreting sources, arranging the material in a comprehensible order, and dealing with gaps, ambiguities, and conflicts in the evidence is far from trivial, and the resulting narratives and the knowledge they encapsulate are an essential part of the historical literature.

However, practicing historians typically expect something more than this. As Martin Campbell-Kelly, one of the most prominent historians of computing, put it in his typically forthright fashion:

> *American history professors are noted for reducing their graduate students to nervous wrecks with the "So What?" question. Having read the student's paper or listened to a presentation, the professor wants to know what the purpose of it all was. The student's methodology may have been impeccable and the fact-grubbing diligent, but what did readers learn that will improve their minds and add to their stock of world knowledge? In short, So What? [...] History involves asking questions and providing context.* (Campbell-Kelly 2010)

An outstanding HOPL paper does not just describe something, but helps the reader come to an understanding of *why it matters*. This should be understood in historical, not technical, terms. In the context of computer science a language might matter because it solves problems in a new and elegant way, introduces innovative features, unifies a number of existing languages, and so on. These are perfectly reasonable things to draw attention to, but from a wider historical point of view languages can also been seen in a number of different contexts along with their significances.

Permit us to quote Herbert Butterfield:

> *We go to the past to discover not facts only but significances. It is necessary that we should go with instinct and sympathy alive and all our humanity awake. It is necessary that we should call up from the resources of our nature all the things which deflect the thought of the scientist but combine to enrich the poet's.* (Butterfield 1931)

Programming Languages in Context

Some computer scientists sometimes take a narrow view of programming languages, identifying them with their formal definitions:

> *A language is an abstraction, a formal notation; notions such as command line, tight control loop, and garbage collector do not and must not occur in a language definition because they concern the implementation only.* (Wirth 2007)

If programming languages are understood only in this way, their history will be similarly narrow: using this context, typical topics might be the identification of features occurring in languages, the classification of languages into paradigms, tracing patterns of evolution and influence between language definitions, or assigning priority for significant innovations. We aim for a broader view, a broader set of contexts.

First of all, some programming "languages" are considered by their designers and users as programming "systems," absorbing notions of command line, garbage collection, programming environment, and in some cases, specialized hardware. These "languages" include the Lisp family and Smalltalk. Further, these days a programming language is considered to include extensive libraries, frameworks, far-flung services & microservices, and integrated development environments (IDEs).

Second, although in the early days of the history of computing this kind of tight technical focus might have been appropriate, nowadays most historians of computing identify themselves as working in the overarching discipline of the *history of technology*. In this field technology is seen as developing in a symbiotic relationship with its social context: this is sometimes described as "social shaping" (Campbell-Kelly 2010), though this phrase makes the relationship seem one-sided. The emphasis on *social* shaping was intended to counter the views that technology evolves autonomously and that new technologies bring about societal change while not themselves being influenced by those changes. The more accurate and current view is that technology and its social context should be seen as two aspects of a coevolving whole.

In this broader context programming languages can be viewed as *technological artifacts*. No particular priority is given to the formal definition, and implementation and other issues are no longer seen as being of secondary importance. Viewing languages this way opens up many new dimensions along which they can be studied, such as the following:

- they have a lifecycle that includes stages of planning, construction, use, maintenance, obsolescence,...
- they exist within a rich collection of other artifacts: manuals, specifications, compilers,...
- they exist within a rich social context: programmers, user groups, managers, development and maintenance committees,...

These perspectives open up a wider range of possible questions and topics for historical investigation:

- not just stories of innovation, but also stories of use

- not just success stories, but failures (the High Performance Fortran paper in HOPL III is an interesting example of a "failure" story)
- not just technical accounts, but histories of application
- reflection of wider historiographical trends: Socio-Technical Systems (STS) approaches, identity issues (gender, race...) in programming languages, importance of materiality

Within the field of the history of computing there is some work that considers individual programming languages in a wide perspective (on Algol, see (Nofre 2010); also Holmevik on Simula (Holmevik 1994); Allen on Cobol (Allen 1981)) but not much, and in particular not much from professional historians of computing. Neither is there a general "history of programming languages" in the sense of an accepted historical survey of developments, nor much reflection on what language history might look like after adopting wider historiographical trends. In part these absences reflect a drift in the history of computing away from overtly technical subjects toward topics of wider historiographical scope.

It might seem that these nuanced historiographical issues are not relevant to HOPL: HOPL authors are not professional historians, and the tradition of the HOPL conferences has not dealt with them. As Jean Sammet put it when describing the objectives of the 1978 HOPL conference:

> *The History of Programming Languages Conference was intended to consider the technical factors which influenced the development of certain selected programming languages. It should be emphasized that this was not a conference on the entire history of programming languages nor entirely a conference on history. It was not even a conference on the entire history of the selected languages, but covered primarily their early developments. The emphasis was on the technical aspects of the language design and creation.* (Wexelblat 1981, xvii)

Although the scope of subsequent HOPLs has expanded somewhat, the relationship between HOPL and the broader history of computing was described by Michael Mahoney as follows:

> *The business of HOPL-II itself is to enrich the sources for programming languages.* (Mahoney 1996b)

Contextual material is not peripheral, but likely to be central to stories that historians of computing would want to tell about programming languages. The historical papers on Algol cited above, for example, refer to the HOPL I proceedings not for the technical aspects of the design and creation of Algol 60, but precisely for this contextual material. And while the technical details of languages can often be retrieved from other sources—formal language descriptions, reference manuals, compiler descriptions, etc—questions of project organization, motivation, and context are much more ephemeral.

For HOPL IV we consider this contextual material essential.

HOPL authors, being participants in the history they write about, are in a unique position to preserve material of potentially greater value than the purely technical. Reviewers should encourage authors to widen their vision from the technical details and incorporate within their papers such contextual material as is available and relevant to the developments being discussed. Specific suggestions for this are given below.

Reviewing

There are two rounds of reviewing: an initial screening followed by an in-depth second round of reviewing.

Round 1

In the first round we want to find out which submissions appear to be headed toward a good final paper; we assume that each paper moving forward will have a strong *shepherd* willing to work with the author. A shepherd is like an editor who is also very familiar with the material—in this case, with the language, languages, or issues that make up the topic of the submission or at least with languages and issues like those in the submission.

Round 2

After a submission makes it into Round 2, the author and shepherd will have six months to finish the paper. The shepherd's job is to help the author with content, structure, organization, and writing. A shepherd interacts closely with the author in several rounds of review and commentary. The Program Chairs and original reviewers will be available to help with this round as well.

After the second round reviews are sent to authors, those authors will have two months to complete their final papers.

Round 1: Read first for sense

It's important to get a sense of what the paper is before you judge it. The paper might be the story of the start or evolution of a language, an exploration of programming language ideas and concepts in the past, the story of the standardization of a language or family of languages, an exploration of a family of languages through some number of generations, a piece of archeology, a philosophical framework, a framing of older concepts, a curious observation, an imaginative understanding, ideas for how things were done in the past, an uncovered way of programming, a pearl of understanding rooted in the past, a notable experience in the past, or perhaps other things.

With a standard technical paper, we understand—before we even start to read—what we are likely to see with only the details of the subject matter in question. A HOPL paper is typically not trying to advance scientific or engineering knowledge at the technical level, but—if it aims at something like that—at the historical / philosophical level or at some kind of meta level but with the twist that it is anchored in the past.

We've used the phrase "a HOPL...is trying," and it's not an accident. At the outset the author of a HOPL paper has some idea of what he or she is trying to say, but in a real sense the paper has the final word on what that is. This might seem odd, but it is a common observation in the writing community. Some examples:

I write entirely to find out what I'm thinking, what I'm looking at, what I see and what it means. What I want and what I fear.

–Joan Didion, Why I Write

You may wonder where plot is in all this. The answer...is nowhere.... I believe plotting and the spontaneity of real creation aren't compatible.... I want you to understand that my basic belief about the making of stories is that they pretty much make themselves. The job of the writer is to give them a place to grow.

–Stephen King, On Writing

There can be no discovery in a world where everything is known. A crucial part of the writing endeavor is to practice remaining in the dark.

–Robert Boswell, The Half-Known World...

I am writing this essay because I am puzzled.

–Richard P. Gabriel, "in the control room of the banquet"

We believe it's the phenomenon of a piece trying to be something that makes writing non-research papers hard for scientists. Scientists are used to a genre that is totally known, and they know how to fill in the blank parts of the template. This is not to say that writing technical papers is simple or mindless—only that in such writing the nature of the piece is not struggling to run away.

In a real sense most difficult writing teaches us how to read it, so you might have to read the first part of a paper to "learn" how to read it, and then go back and re-read based on what you learned. For example, the first paragraph of Cormac McCarthy's "All The Pretty Horses" can be difficult for some readers, but it also teaches readers how to read the rest of the novel:

The candleflame and the image of the candleflame caught in the pierglass twisted and righted when he entered the hall and again when he shut the door. He took off his hat and came slowly forward. The floorboards creaked under his boots. In his black suit he stood in the dark glass where the lilies leaned so palely from their waisted cutglass vase. Along the cold hallway behind him hung the portraits of forebears only dimly known to him all framed in glass and dimly lit above the narrow wainscotting. He looked down at the guttered candlestub. He pressed his thumbprint in the warm wax pooled on the oak veneer. Lastly he looked at the face so caved and drawn among the folds of funeral cloth, the yellowed moustache, the eyelids paper thin. That was not sleeping. That was not sleeping.

–Cormac McCarthy, All The Pretty Horses

This paragraph teaches us about how McCarthy uses punctuation—when he leaves it out, when he includes it—by showing examples, and it also shows how McCarthy uses multiple nuances of words to convey both facts and mood: glass as reflector versus glass as transparency, dim as darkness versus dim as unknown.

Rounds 1&2: Read for value

Once you know what the paper is trying to be you are in a position to start making judgments. Sometimes you need to let the paper sit in your head for a while before you judge. Try to judge the paper against what it is trying to be and not what you want it to be. After that, judge whether the best version of the paper would be valuable as part of the body of the historical record of the programming-centric community. Finally judge whether this paper can get close enough to that ideal to deserve publication.

Here are the questions we tend to address:

- **Does the paper present a serious historical perspective?**
- **Was it clear why the topic is important to our community?**
- **Did I come away knowing something new about the past?**
- **Is it likely that others in the community would also come away with similar thoughts?**
- **Is this a piece that the community would be better off having than not?**
- **Were all the relevant sources uncovered?**
- **Were they interpreted well?**
- **Was I convinced by the evidence?**
- **Were appropriate comparisons made?**

In close proximity to these questions are questions of voice, confidence, trust, freshness, and quality:

- **Did I feel in the presence of a trustworthy guide?**
- **Was the voice appropriate to the material; did it help get the points across?**
- **Did the guide make me feel safe and confident in the journey?**
- **Did reading the paper feel like a taking a journey?**
- **Was I bored?**
- **Was I ever confused?**
- **Were different parts of the story well-connected?**
- **Were the parts of the story told in the best order (or, a good order)?**
- **Was the paper polished? How much work will it take to finish it?**
- **Did the paper surprise or delight me?**
- **Was the paper too complicated?**
- **Was the writing beautiful?**
- **Did I need too much specialized knowledge?**
- **Was the paper the right length?**

Part of trustworthiness is technical historical “correctness.” By technical correctness we mean that the available evidence must be treated well and properly—when it isn’t, not only is the historic record tainted, but the reader can lose trust in the teller.

Evidence

History is not science, but nevertheless has a strong empirical basis. Historical writing is based on *sources,* and an important part of evaluating a history paper is judging the adequacy of the author's range of sources and the use made of them.

Classically, sources have been taken to be written documents, usually preserved in archives. In recent decades, historians have widened the range of acceptable source materials, and in the context of writing the history of programming languages, this can be widened again. Possible sources include the following:

- technical reports
- minutes of meetings
- memos
- presentations and lectures: copies of slides, recordings, videos, demos
- electronic communications: emails or social media
- mailing list or usergroup archives
- personal manuscripts, notes, diagrams
- source code of compilers, interpreters
- source code of libraries, application programs
- artifacts
- oral histories or interviews
- personal reminiscences

Reviewers should bear in mind the following questions about sources:

- **Does the author describe the source material that the paper is based on?**
- **Are the sources clearly identified?** For public sources, enough information should be given to enable interested readers to locate the sources themselves. In cases of doubt, more bibliographic information should be provided rather than less—HOPL has no page limit. Private sources should be clearly identified as such, and as much identifying information as possible given (dates of emails,...)
- **Is the range of sources appropriate for the story being told?** This will depend on context: it would be odd to tell the history of COBOL, for example, without appealing to the records of the CODASYL committee, but for a smaller or more personal project, the available sources might be more private and ephemeral.
- **Are the sources clearly cited in the text where they are drawn upon?** Historians tend to cite sources by giving a name and a date, rather than a bare number indexing a list of references. Sometimes called the "Harvard style," this helps the reader to keep the chronology in mind without cluttering the text with boilerplate phrases likes "At a meeting on June 24,...."

Writing a HOPL paper may lead an author to collect a set of documentary material that forms a significant archive in its own right. Valuable as such archives are, Michael Mahoney (Mahoney 1996b) pointed to a likely

gap: sources will not reflect the things that are completely taken for granted because nobody bothers to record the things that "everybody knows." As well as explicit sources, then, authors can be asked to throw light on the tacit knowledge within which a particular development took place (that is, the authors should take into account familiar practices of work as well as explicit textual material):

- **Is all the relevant tacit knowledge at the time adequately described?**

Interpreting sources

Historians use sources as the raw material from which to build accounts of what went on in the past. This is sometimes described as a form of re-creation. For example, Dick Hamming described the goal of historical research as "knowing what they thought at the time," a formulation that was later picked up by Michael Mahoney in HOPL II. But in most if not all cases, the past cannot simply be "read off" the sources. Interpretation is required, in at least two distinct forms: external and internal.

External interpretation—sources as evidence

A historian cannot assume that the available sources are complete, consistent, or reliable. In much the same way as evidence in a legal trial, all the available source material needs to be compared, interrogated, and tested for consistency. Some of the problems are as follows:

- some sources may simply be missing
- sources may conflict: for example, different people's accounts of a meeting
- many sources will have been written with a view to bringing about a particular outcome; for example, writing a proposal or describing progress in such a way as to impress a potential or actual project funder
- some sources are intrinsically unreliable: reminiscences and oral histories recorded in some cases many years after the events concerned are classic examples

Historians must triangulate between the available sources: claims made in oral histories can be cross-checked with contemporary documentary evidence; different accounts of the same event can be compared with each other. In a slightly larger context, historians need to make judgments about actors' motivations that might depend on a much wider range of source material.

A wide range of sources is important, not simply because more sources provide more information, but to provide a wide context for the historian to interpret the material.

Internal interpretation—sources come from a different culture

The Past is a Foreign Country: *They do things differently there.* The statement in bold is the title of a book by David Lowenthal (1985) on the past as viewed from the present. The statements in bold and italic taken together are the first line of a book by L. P. Hartley called *The Go-Between,* published in England in 1953. One danger in writing about the past is to write about it as if everything we know today was known then; another

danger is to write about the past as if the present were the inevitable result of that past. This is a cardinal historical sin—historians label these two dangers "presentism" or "Whiggism" (Butterfield 1931).

What we hope to see in HOPL papers is a description of the origin of programming languages and programming language ideas written with explicit descriptions of the context and the knowledge available at that time. Moreover, we do not necessarily want to read about how ideas from the past formed the basis of ideas important in the present. Such an analysis is ultimately important, but it is not at the center of the intention behind HOPL; such an analysis might form the basis of a paper in a future HOPL.

For example, in "The Evolution of Lisp," published in HOPL II in 1993, Steele and Gabriel describe how the Lisp 1.5 (circa 1960) era functions ERRSET and ERR were used by programmers to cobble together a form of what we now call a dynamic non-local exit, which was codified in MacLisp's (circa 1972) CATCH and THROW functions—these were adopted into Common Lisp (circa 1984) with some modifications; but Steele and Gabriel did not describe how these ideas flowed into other programming languages closer in vintage to 1993.

Imagine historical actors choosing between two alternatives. As historians, we know what the outcome was, and probably have strong views on whether the actors made the right or wrong choice. This information was not available to the actors, and interpreting their actions in the light of it biases our account of the past.

The issue is that "evolution" and "progress" are simplifications of a very complex technical, cultural, economic, and political set of interacting processes, and a simple cause & effect line of explanation from the past to the present is likely born of wishful thinking.

It is very tempting for scientists looking at the past to interpret what they see in terms of the present. The literature contains assertions such as "the Babylonians were really writing algorithms," or that Zuse's Plankalkul was the first non-von Neumann programming language, etc.

Imagine that Steele and Gabriel had written that the functions ERRSET and ERR were used by programmers "to simulate the monad of exceptions." Such a statement would be undesirable. First, it appears to imply an anachronism: that Lisp programmers in the 1960s had the idea of monads explicitly in mind. But monads did not become a topic of programming language design until much later (in fact, just before Steele and Gabriel wrote that 1993 paper). Second, such a statement would be guilty of explaining the thoughts of those earlier programmers in modern terms, rather than explaining how they themselves viewed the problem. Worse, the wording of the statement does not clarify which of these two interpretations might have been intended.

It requires considerable mental discipline to attempt to read a source as it would have been read at the time it was written, putting aside our knowledge of subsequent developments.

> *What makes history? It's a matter of going back to the sources, of reading them in their own language, and of thinking one's way back into the problems and solutions as they looked then. It involves the imaginative exercise of suspending one's knowledge of how things turned out so as to recreate the possibilities still open at the time.* (Mahoney 1996)

Fallacy of the Lone Genius: We hope to avoid histories that explain the origins of ideas and mechanisms by reference to the sole efforts of a single individual or a small group. When ideas, inventions, and innovations are explored, we almost always find similarities to other things and a host of precursors. Taking credit is one thing; perhaps we can accept taking blame for mistakes.

Incommensurability: By this we mean the idea that the meanings of language and words are different in different time periods and in different paradigms (as noted by Thomas Kuhn (Kuhn 1996)). Someone in one time frame might not be able to understand what someone in a different time frame means by words, phrases, and sentences even when the same words and language are common to both frames. Sometimes this is true incommensurability, sometimes simple misunderstandings or ambiguity of terms.

Even in computing history, where the past is not all that long ago, technical vocabulary changes its meaning quite quickly. A word as simple as "program" did not have the same meaning when used by John von Neumann in 1945 as it does now: a crucial part of the historian's job is to be sensitive to the possibility of this kind of change.

In "The Virtual Memory in the STRETCH Computer" (1959, http://dx.doi.org/10.1145/1460299.1460308), Cocke and Kolsky describe a mechanism that supports prefetching of instruction operands and delayed stores of instruction results. It is certainly not what we mean today by the phrase "virtual memory"; they describe it in terms that make it sound like what we would now call a cache, but it isn't that either—it's more like what we would now call a circular buffer with multiple pointers into it. They also speak of the "levels" of the virtual memory, but this term does not refer to what we might now call different levels of backing store, nor does it refer to what we might now call multiple cache levels (such as L1 and L2 cache), but simply to different positions in the circular buffer. It's not enough to read the words of an old paper; you have to know what they were intended and understood to mean in that time and place.

In their seminal paper, "Mixin-based Inheritance," Gilad Bracha and William Cook appear to confuse the notions of *system* and *language* as well as not grasp totally the idea of *mixin* as described by the creators of earlier programming *systems* (Bracha & Cook 1990). Note: This instance of incommensurability gave rise to an example of the "The Past is a Foreign Country" problem. In writing about mixins as defined in those earlier programming systems (Flavors and CLOS), those earlier authors (Cannon, Moon, and Bobrow) were careful to explain that modular design was essential to good practice, but that it was the software designer who must make those clean designs—programming systems provided mechanisms to accomplish the implementation of modular designs, but it was not the responsibility of programming languages to enforce them. Bracha and Cook ignored the mechanisms provided by Flavors and CLOS to avoid the problems they wrote about in their paper. (Gabriel 2012)

In our own lifetimes we have seen communities differ in their uses of the phrases "strong typing," "static typing," "dynamic typing," "weak typing," and "untyped." In one community, "strong typing" means that a programming system would never permit an operation to be invoked on arguments of improper types, and this could be accomplished with either "static typing" or "dynamic typing." In another, "strong typing" is

synonymous with “static typing,” and languages that didn’t have “static typing” either had “weak typing” or were “untyped.”

Adopting a suitably distanced “historical stance” is of course even harder to achieve when actor and author are the same person.

Descriptive history

Once the available sources have been tracked down and interpreted, a natural approach is to write a straightforward chronological description.

A temptation is to try to put everything in: after all that work collecting and reading the sources, authors are often reluctant to leave any of them out. This temptation is particularly strong when the author was a participant and is emotionally engaged with the events being described. HOPL papers are traditionally long and comprehensive, aiming at completeness. Sometimes the detail can make it hard for the reader to see the forest for the trees. G. R. Elton identified some common pitfalls as follows:

- a devotion to detail for its own sake
- treating all facts as equally significant
- a lack of selection, judgment, and evaluation in the treatment of sources

For example, if a language development involved a long series of committee meetings over a number of years, it would be possible to simply *chronicle* these meetings, giving each its own subsection and describing who attended, what decisions were taken, what features were introduced, removed, or modified, and so on.

> *Chronicle is [...] a setting down of events one after the other, without considered discrimination or any discernible purpose except merely to record.* (Elton 1967)

Campbell-Kelly comments on similar tendencies in texts about historic computers:

> *A typical history would consist of a detailed description of how the machine was physically constructed, how it was programmed, and there would be supporting appendices of instruction code and a specimen program. [...] Papers of this kind are useful as summaries of particular machines but they take us only a little further than the original programming manual. [...] It lacks a narrative balance.* (Campbell-Kelly 2010).

Not everything is equally interesting. The reviewer’s job is to act as a reflective reader, answering questions such as the following:

- **Was the narrative engaging, or were you tempted to skip?**
- **Was it easy to get a grip on the main outline of the story, or were you lost in minutiae?**

- **Did you end up with a sense of what made this language development effort different from others?**

A common technique that historians use to introduce structure is *periodization*. The history of the previous two millennia is made more approachable and memorable when understood as a sequence of extended temporal periods: dark ages, medieval, renaissance, early modern, and so on. The best periodization for making sense of a language development depends on the internal details of the narrative. In some cases, different language versions or releases might define useful periods, while in others the development might have been more significantly shaped by changes in administrative structure, or even significant technical decisions about the scope, content, or form of the language.

Encouraging an author to reflect on an appropriate periodization for their story can foster reflection on priorities: what are the really important parts of the story?

Historical narratives do not need to be all-inclusive. The historian's task is to present an interesting account of the significant developments by means of a judicious selection from the available material. If necessary, tables, lists, and summaries can be added as appendices to a paper rather than cluttering up the narrative with too much detail.

Framing the story

While admitting the value and legitimacy of purely descriptive accounts, practicing historians emphasize the value of writing that aims to *explain why* as well as *describe how*, a difference often characterized by presenting history as written to answer a specific question about the past.

> *His work has value of a kind because the facts he collects can in the proper light contribute to the answering of real questions, that is intellectually challenging questions.* (Elton 1967)

In a similar vein, Campbell-Kelly (an admirer of Elton's book) wrote:

> *History involves asking questions and providing context.* (Campbell-Kelly 2010)

The HOPL website provides detailed lists of questions that authors were invited to consider when collecting material and drafting their papers. Some of these deal with internal and rather technical points, but others point toward more general aspects of the context within which the technical developments take place. The guidelines for authors emphasize that a paper should not simply consist of a list of answers to these questions, but don't really distinguish different categories of question.

As a general rule, papers will be improved—made more interesting and relevant to a wider range of readers—if the core descriptive narrative is framed by a consideration of relevant or important aspects of the context within which the technical developments took place. The number of historians of computing interested in the details of a specific language is probably rather small while the number interested in, say, the differences between

languages developed in industry and academia is potentially larger. The more the context can be made relevant to the wider concerns of the historical community, the more a paper is likely to be read. The following is a list of contextual questions reviewers should ask:

- **Organizational context**
  - **Where was the language developed? In industry, academia, or elsewhere.**
  - **Was it a large and bureaucratically managed project, or did it start as an individual's private undertaking?**
  - **Who sponsored the development?**
  - **What were the initial aims of the project?**
  - **Why was a new language development begun?**
- **Technical context**
  - **Was the language developed for a particular type of machine?**
  - **What was its relationship to existing languages?**
- **Biographical context**
  - **Who were the principal developers?**
  - **What were their backgrounds?**
  - **What brought them to this project, and what did they bring to it?**
  - **What about the rest of the team?**
- **Context of use**
  - **Was the language intended as a research project in language design?**
  - **Was it intended for application in particular areas? What areas?**
  - **What were the perceived shortcomings of existing languages?**
  - **How are the characteristics of the application area reflected in the language?**
- **User community**
  - **What was the interplay between use and development?**

Specific issues

This section describes a mixed bag of questions that reviewers should be aware of, ranging from issues that are specific to HOPL papers to more general questions in historiography.

Technical descriptions and evaluations of languages: A retrospective description of a language accompanied with reflections on what was done right and what wrong is not really a history paper. If the primary purpose of the paper is evaluative, it is hard not to fall into the trap of interpreting the past in terms of the present. Rather than observing that, say, a decision made was wrong, it is more useful to analyze why, in the past, that decision was made.

This is not to say that technical details have no place in history. Many good HOPL papers provide a lot of detail about the syntax and semantics of language features. This can be completely appropriate and necessary in many cases, but it is important to resist the temptation to reproduce this material for its own sake.

HOPL papers are not language definition documents nor commentaries thereon, and technical material should be included not for its own sake, but because it serves some higher purpose in the overall conception of the paper.

Purely personal reminiscences: These can be fascinating and provide an invaluable and irreplaceable sense of "what it was like." But as sources, they are famously problematic and require careful handling. Memory is fallible, and recollections should be tested against contemporary documentary sources whenever possible.

Asymmetrical explanations: At HOPL I, Alan Perlis wrote the following:

> *The acceptance of Fortran within SHARE and the accompanying incredible growth in its use, in competition with a better linguistic vehicle [Algol], illustrated how factors other than language determine the choice of programming language of users.* (Perlis 1981)

If Algol had triumphed over Fortran, it might be tempting to attribute that simply to it being a "better linguistic vehicle," whereas explaining the success of the (supposedly) technically inferior Fortran requires extra-linguistic factors to be taken into account. (Note that Perlis does not commit this error: he is careful to say that "factors other than language determine the choice of programming language," not "...the success of Fortran"—he explains success and failure in the same terms.)

Following a highly influential book by David Bloor (1976), this kind of explanation is widely seen by historians as a significant methodological error. Bloor argued for an *explanatory symmetry*, where all phenomena are explained—and their history written—by appealing to the same types of causes. Any hypothetical triumph of Algol 60 over Fortran should also be explained as the consequence of "external" factors such as institutional support, a prompt supply of efficient compilers, etc.

This does not mean that technical factors cannot enter into such explanations, but that if they do, they should be seen not as absolutes; but in some context: for example, the *belief* that Algol was a better language might have been a factor in its gaining the support of key institutions, or its provision of recursion, say, might have been key to making the programming of certain salient problems easier.

Teleological explanations: Unlike historical actors, we know how things turned out—and it is sometimes tempting to see history as a flow of events leading up to the present state of affairs. Events, decisions, and choices that conform to our current understanding can seem natural, and require no specific explanation beyond the observation that they were, as things have turned out, the obvious or best approach. Errors or decisions that were subsequently overturned, on the other hand, seem to require explanations that appeal to a different set of factors (an asymmetrical explanation). Social, financial, or organizational factors might be invoked to explain why actors "went wrong." This kind of methodological error amounts to using the present state of affairs to explain the past, as if the present state of affairs had been the original *goal* of the past, hence "teleological."

Creative Nonfiction techniques

It's possible a HOPL paper will use techniques from creative nonfiction. Creative nonfiction is a genre that uses techniques from fiction—from stories, from novels, from poetry—to further the goals of the nonfiction. The first well-known instance of this was the nonfiction novel "In Cold Blood," by Truman Capote, which traced the murders of the Clutter family in Holcomb, Kansas, in 1959, along with the subsequent investigation, trial, and executions of the killers.

Because we expect to get submissions that tell the story of people designing, implementing, using, and otherwise working on programming languages (and systems) from a variety of viewpoints and from authors from a variety of cultures, we may very well see some creative nonfiction techniques. We should keep in mind Butterfield's advice (mentioned earlier):

> *It is necessary that we should call up from the resources of our nature all the things which deflect the thought of the scientist but combine to enrich the poet's.* (Butterfield 1931)

Two papers published in scientific venues that use creative nonfiction techniques are Tomas Petricek's "Miscomputation in software: Learning to live with errors" and David West's "The Cuban Software Revolution: 2016–2025." Petricek's essay appeared in the first Art, Science, and Engineering of Programming Conference in 2017, and its topic was the treatment of software errors. David West's Onward! essay—which appeared in Onward! Essays 2015—uses a fictional setting and the format of a dialog to frame his argument, which is that there is an approach to writing software that is better than what is used in commercial-centric development methodologies.

Writing that brings tears to your eyes

One of us—Gabriel—happens to believe in beautiful writing, and for him, HOPL papers must exhibit beautiful writing. This is his bias. Watch out for it.

Here is what Ralph Waldo Emerson wrote about Michel Eyquem de Montaigne, arguably the first essayist:

> *I remember the delight and wonder in which I lived with it [Montaigne's Essays]. It seemed to me as if I had myself written the book, in some former life, so sincerely it spoke to my thought and experience. ...*
>
> *The sincerity and marrow of the man reaches to his sentences. I know not anywhere [in] the book that seems less written. It is the language of conversation transferred to a book. Cut these words, and they would bleed; they are vascular and alive.*
>
> –Ralph Waldo Emerson, *Montaigne; or, the Skeptic*

However, writing that is beautiful might not serve the paper well. In those cases, the writing should serve the essay and not drive it.

Trivialities

Although the practice has become more sophisticated or, shall we say, suave over the years, a common practice in computer science research papers is to put the punchline at the start of the paper and to describe early on all the results and where they will be presented in the paper—a sort of roadmap. Articles talking about how to write research papers emphasize this practice and some talk about why it makes sense. And it does (or at least it can) make sense for purely technical / scientific papers, but it doesn't necessarily make sense for HOPL papers and especially for ones that are like essays. The punchline comes where the punchline needs to be.

One common view is that a paper should move forward like a predator toward its conclusion. But some points need to sink in before they are deeply understood, and sometimes a number of supporting arguments or thoughts need to come together before a point can be properly made and understood. An *episodic* approach can help here.

Episodic writing proceeds by telling a little bit of the story, then switching for a while to another aspect of the overall narrative. Imagine a story in which two remarkable characters come together for an epic confrontation after each has had a series of important, relevant, and difficult challenges. One way to tell this story is to tell the entire story of one character from start to confrontation, then tell the entire story of the other character from start to confrontation, and finally to tell the story of the confrontation. If the two stories are long, the first one might be forgotten by the final confrontation; and sometimes an episode from one character's story is best told at the same time a (related) episode from the other character's story is told. (If you like science fiction, check out *Lucifer's Hammer* by Niven and Pournelle (1977). In the words of Judith Yamamoto's review for the *Library Journal*, it has "a gigantic but well developed and coordinated cast of characters"; the stories of these individual characters are necessarily developed in an episodic style.)

There are many such nonlinear structures in narrative: an envelope in which the start of the story is similar to the end of the story ("I met her on a beach on a cloudy day.... Later, years after she left and started her dental floss farm in Montana, I went to that beach—again, on a cloudy day"). Another structure is a flash forward where the final surprising scene is previewed at the start of the story. It wasn't written that way, but the story of Moby Dick could have started like this:

> *On the second day, a sail drew near, nearer, and picked me up at last. It was the devious-cruising Rachel, that in her retracing search after her missing children, only found another orphan.*
>
> —
>
> *Call me Ishmael. Some years ago—never mind how long precisely—having little or no money in my purse, and nothing particular to interest me on shore, I thought I would sail about a little and see the watery part of the world....*
>
> –Herman Melville & Richard P. Gabriel, *Moby-Dick; or, The Whale*—a mashup

When reviewing a HOPL paper, determine whether the structure supports the material.

- **Did the structure make things easy to grasp?**

- **Did it help propel you forward?**
- **Did you feel as though in the presence of a worthy guide?**

HOPL papers don't always need descriptive section headings or summaries of the material at the start of sections unless such things are important for understanding. This sounds like we are talking about not bothering to be helpful and supportive to the reader, but we are trying to suggest ways to achieve the vivid and continuous dream that good writing is, which we believe is as important as or more important than such trivial signposts.

Length: Two forces come into play here. The first is that writing is hard for authors, and sometimes authors will explain too little in order to be able to stop writing. They will become too generous in assumptions about readers' knowledge and the paper will be too short and hence a mystery. The writer gives up writing. The second is that authors have gathered a lot of material in preparation for writing the paper, and they don't want it to go to waste, so they write and write and write. Too many side paths are explored; the writing seems long-winded. Readers will give up reading.

You should read the whole paper, but please note where interest in the material ends—your review can talk about why that's where a reader would stop. If a reviewer couldn't finish the paper, there are likely lots of readers who won't be able to finish either. This is important information for the author.

So…

You were assigned to review a HOPL paper and it doesn't seem to be like anything you have reviewed before for a journal or technical conference. You don't know how to approach it, and are tempted to review it as "a scientific investigation…to be judged according to the standards in our field." Don't do that.

Instead, read the paper as if a colleague enthusiastically told you that you should have a look at it. Try to get its message and how the author is presenting that message. If you feel that it is a message worth spreading, think about whether this paper is a good starting point for that message, and then think how it could be made better.

That's how you start your review.

References

Allen, F. E. (1981). "The history of language processor technology at IBM," *IBM Journal of Research and Development*, Vol. 25, Num. 5 (September 1981), pp. 535-548.

Bergin, T. J., Gibson R. G. (eds.) (1996): History of Programming Languages II. ACM Press/Addison-Wesley.

Bloor, D. (1976). *Knowledge and Social Imagery*. University of Chicago. (Second edition, 1991.)

Boswell, R. (2008). *The Half-Known World: On Writing Fiction*.' Graywolf Press. St. Paul, MN, 2008.

Bracha, G. & Cook, W. (1990). *Mixin-based Inheritance.* Proceedings of the Fifth ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications. 1990.

Butterfield, H. (1965). *The Whig Interpretation of History*, W. W. Norton.

Campbell-Kelly, M. (2010). Writing computer history: Historians, pioneers, and practitioners.
Didion, J. (1976). *Why I Write*. New York Times. December. 5, 1976.
Elton, G. R. (1967). *The Practice of History*. (Crowell/Sydney University Press).
Emerson, R. W. (1876). *Montaigne; or, the Skeptic*.
Gabriel, R. (2012). *The Structure of a Programming Language Revolution*, Onward! 2012.
Gabriel, R. (2015). *"in the control room of the banquet".* Onward! 2015.
Hashagen, U., Keil-Slawik, R., Norberg, A. L. (eds.) (2002). *History of computing: Software issues*. Springer.
Holmevik, J. R. (1994). Compiling SIMULA: A Historical Study of Technological Genesis. *IEEE Annals of the History of Computing*, Vol 16, No. 4, 1994.
Jones, C. (2003). The early search for tractable ways of reasoning about programs. *IEEE Annals of the History of Computing* 12(2), 26-49.
King, S. (2001). *On Writing: A Memoir of the Craft.* Simon & Schuster.
Kuhn, T. (1996). *The Structure of Scientific Revolutions.* University Of Chicago Press. 1996.
Mahoney, M. (1988). The history of computing in the history of technology. *Annals of the History of Computing* 10(2), 113-125.
Mahoney, M. (1996a). Issues in the history of computing. In (Bergin and Gibson 1996, 772-791).
Mahoney, M. (1996b). What makes history? In (Bergin and Gibson 1996, 831-2).
McCarthy, C. (1992). *All the Pretty Horses*. Vintage.
Metropolis, N.. Howlett, J., Rota, G.-C. (eds.) (1980). *A history of computing in the twentieth century*. Academic Press.
Nofre, D. (2010). Unraveling Algol: US, Europe, and the creation of a programming language. *IEEE Annals of the History of Computing* 32(2), 58-68.
Ryder, B., Hailpern, B. (eds.) (2007). *Proceedings of the third ACM conference on History of Programming Languages*. (https://dl.acm.org/citation.cfm?id=1238844).
Sammet, J. (1969). *Programming languages: History and fundamentals*. Prentice-Hall, Inc.
Steele, G. & Gabriel, R. (1993). *The Evolution of Lisp.* HOPL II.
Wexelblat, R. L. (ed.) (1981). *History of programming languages*. Academic Press.
Wirth, N. (2007). Modula-2 and Oberon. In (Ryder and Hailpern 2007).

Reviewer Questions

This is a list of all the reviewer questions described herein:

- **Does the paper present a serious historical perspective?**
- **Was it clear why the topic is important to our community?**
- **Did I come away knowing something new about the past?**
- **Is it likely that others in the community would also come away with similar thoughts?**
- **Is this a piece that the community would be better off having than not?**
- **Were all the relevant sources uncovered?**
- **Were they interpreted well?**
- **Was I convinced by the evidence?**
- **Were appropriate comparisons made?**
- **Did I feel in the presence of a trustworthy guide?**
- **Was the voice appropriate to the material; did it help get the points across?**
- **Did the guide make me feel safe and confident in the journey?**
- **Did reading the paper feel like a taking a journey?**
- **Was I bored?**
- **Was I ever confused?**
- **Were different parts of the story well-connected?**
- **Were the parts of the story told in the best order (or, a good order)?**
- **Was the paper polished? How much work will it take to finish it?**
- **Did the paper surprise or delight me?**
- **Was the paper too complicated?**
- **Was the writing beautiful?**
- **Did I need too much specialized knowledge?**
- **Was the paper the right length?**
- **Does the author describe the source material that the paper is based on?**
- **Are the sources clearly identified?** For public sources, enough information should be given to enable interested readers to locate the sources themselves. In cases of doubt, more bibliographic information should be provided rather than less—HOPL has no page limit. Private sources should be clearly identified as such, and as much identifying information as possible given (dates of emails,...)
- **Is the range of sources appropriate for the story being told?** This will depend on context: it would be odd to tell the history of COBOL, for example, without appealing to the records of the CODASYL committee, but for a smaller or more personal project, the available sources might be more private and ephemeral.
- **Are the sources clearly cited in the text where they are drawn upon?** Historians tend to cite sources by giving a name and a date, rather than a bare number indexing a list of references.
- **Is all the relevant tacit knowledge at the time adequately described?**
- **Was the narrative engaging, or were you tempted to skip?**
- **Was it easy to get a grip on the main outline of the story, or were you lost in minutiae?**

- **Did you end up with a sense of what made this language development effort different from others?**
- **Organizational context**
    - **Where was the language developed? In industry, academia, or elsewhere.**
    - **Was it a large and bureaucratically managed project, or did it start as an individual's private undertaking?**
    - **Who sponsored the development?**
    - **What were the initial aims of the project?**
    - **Why was a new language development begun?**
- **Technical context**
    - **Was the language developed for a particular type of machine?**
    - **What was its relationship to existing languages?**
- **Biographical context**
    - **Who were the principal developers?**
    - **What were their backgrounds?**
    - **What brought them to this project, and what did they bring to it?**
    - **What about the rest of the team?**
- **Context of use**
    - **Was the language intended as a research project in language design?**
    - **Was it intended for application in particular areas? What areas?**
    - **What were the perceived shortcomings of existing languages?**
    - **How are the characteristics of the application area reflected in the language?**
- **User community**
    - **What was the interplay between use and development?**
- **Did the structure make things easy to grasp?**
- **Did it help propel you forward?**
- **Did you feel as though in the presence of a worthy guide?**