How to write a great research paper

Roy Campbell
I read your draft of the paper for the conference. Great work!

... I just made a few minor alterations...

Uh... Um... You... You rewrote the whole paper, sir...

Ah, yes, well... I didn't want you to embarrass yourself on your first paper.

Thanks... I think...
Four take aways

1. Write a paper about any idea no matter how weedy and insignificant it may seem to you. Repeat often.
2. Your paper should have just one clear, sharp idea
3. Make your paper look professional and interesting
4. Give credit when credit is due
Writing papers is a skill that can be learned

- Your ideas will have more impact
- You will have better ideas
Writing papers – The process

Idea->(Draft Paper-> Research)*->Submit

- Improves focus
- Clarifies goals
- Identifies what you don’t understand
- Allows dialogue with others
- Reality check
- Critique
- Collaboration

Idea-> Research -> Write Paper
The purpose of your paper
Why bother?

Your goal: to infect the mind of your reader with your idea, like a virus.
The idea

- Figure out what your idea is
- Be 100% explicit:
  - “The main idea of this paper is....”
  - “In this section we present the main contributions of the paper.”
- Many papers contain good ideas, but do not distil what they are.

Idea
A re-usable insight, useful to the reader
One ping

- Your paper should have just one “ping”: one clear, sharp idea
- Read your paper again: can you hear the “ping”?
- You may not know exactly what the ping is when you start writing; but you must know when you finish
- If you have lots of ideas, write lots of papers

Thanks to Joe Touch for “one ping”
Your narrative flow

- Here is a problem
- It’s an interesting problem
- It’s an unsolved problem
- Here is my idea
- My idea works (details, data) ✓
- Here’s how my idea compares to other people’s approaches

I wish I knew how to solve that!
I see how that works. Ingenious!
The purpose of your paper is not...

To describe the WizWoz system

- Your reader does not have a WizWoz
- She is primarily interested in re-usable brain-stuff, not executable artefacts
Structure (conference paper)

- Title (1000 readers)
- Abstract (4 sentences, 100 readers)
- Introduction (1 page, 100 readers)
- The problem (1 page, 10 readers)
- My idea (2 pages, 10 readers)
- The details (5 pages, 3 readers)
- Related work (1-2 pages, 10 readers)
- Conclusions and further work (0.5 pages)
The abstract

- Write the abstract often (first and last).
- Used by program committee members to decide which papers to read
- Four sentences [Kent Beck]
  1. State the problem
  2. Say why it’s an interesting problem
  3. Say what your solution achieves
  4. Say what follows from your solution
## Structure

- Abstract (4 sentences)
- **Introduction** (1 page)
- The problem (1 page)
- My idea (2 pages)
- The details (5 pages)
- Related work (1-2 pages)
- Conclusions and further work (0.5 pages)
The introduction (1 page)

1. Describe the problem
2. State your contributions

...and that is all

ONE PAGE!
Describe the problem

1 Introduction

There are two basic ways to implement function application in a higher-order language, when the function is unknown: the push/enter model or the eval/apply model [11]. To illustrate the difference, consider the higher-order function zipWith, which zips together two lists, using a function k to combine corresponding list elements:

\[
\text{zipWith} : (a \to b \to c) \to [a] \to [b] \to [c]
\]
\[
\text{zipWith } k \ [\ ] \ [\ ] = [\ ]
\]
\[
\text{zipWith } k \ (x:xs) \ (y:ys) = k \ x \ y : \text{zipWith } xs \ ys
\]

Here k is an unknown function, passed as an argument; global flow analysis aside, the compiler does not know what function k is bound to. How should the compiler deal with the call k x y in the body of zipWith? It can’t blithely apply k to two arguments, because k might in reality take just one argument and compute for a while before returning a function that consumes the next argument; or k might take three arguments, so that the result of the zipWith is a list of functions.
State your contributions

- Write the list of contributions first
- The list of contributions drives the entire paper: the paper substantiates the claims you have made
- Reader thinks “wow, if they can really deliver this, that is exciting; I’d better read on”
State your contributions

Which of the two is best in practice? The trouble is that the evaluation model has a pervasive effect on the implementation, so it is too much work to implement both and pick the best. Historically, compilers for strict languages (using call-by-value) have tended to use eval/apply, while those for lazy languages (using call-by-need) have often used push/enter, but this is 90% historical accident — either approach will work in both settings. In practice, implementors choose one of the two approaches based on a qualitative assessment of the trade-offs. In this paper we put the choice on a firmer basis:

- We explain precisely what the two models are, in a common notational framework (Section 4). Surprisingly, this has not been done before.

- The choice of evaluation model affects many other design choices in subtle but pervasive ways. We identify and discuss these effects in Sections 5 and 6, and contrast them in Section 7. There are lots of nitty-gritty details here, for which we make no apology — they were far from obvious to us, and articulating these details is one of our main contributions.

In terms of its impact on compiler and run-time system complexity, eval/apply seems decisively superior, principally because push/enter requires a stack like no other: stack-walking
Contributions should be refutable

<table>
<thead>
<tr>
<th>NO!</th>
<th>YES!</th>
</tr>
</thead>
<tbody>
<tr>
<td>We describe the WizWoz system. It is really cool.</td>
<td>We give the syntax and semantics of a language that supports concurrent processes (Section 3). Its innovative features are...</td>
</tr>
<tr>
<td>We study its properties</td>
<td>We prove that the type system is sound, and that type checking is decidable (Section 4)</td>
</tr>
<tr>
<td>We have used WizWoz in practice</td>
<td>We have built a GUI toolkit in WizWoz, and used it to implement a text editor (Section 5). The result is half the length of the Java version.</td>
</tr>
</tbody>
</table>
No “rest of this paper is...”

- **Use forward references from the narrative in the introduction.**
  
  The introduction (including the contributions) should survey the whole paper, and therefore forward reference every important part.

- **Not:**
  
  “The rest of this paper is structured as follows. Section 2 introduces the problem. Section 3 ... Finally, Section 8 concludes”.

---
We adopt the notion of transaction from Brown [1], as modified for distributed systems by White [2], using the four-phase interpolation algorithm of Green [3]. Our work differs from White in our advanced revocation protocol, which deals with the case of priority inversion as described by Yellow [4].
Structure

- Abstract (4 sentences)
- Introduction (1 page)
- The problem (1 page)
- My idea (2 pages)
- The details (5 pages)
- Related work (1-2 pages)
- Conclusions and further work (0.5 pages)
Presenting the idea

- Conveying the intuition is primary, not secondary
- Once your reader has the intuition, she can follow the details (but not vice versa)
- Explain it as if you were speaking to someone using a whiteboard
- Even if she skips the details, she still takes away something valuable
3. The idea

Consider a bifircuated semi-lattice $D$, over a hyper-modulated signature $S$. Suppose $p_i$ is an element of $D$. Then we know for every such $p_i$ there is an epi-modulus $j$, such that $p_j < p_i$. 
Putting the reader first

- Choose the most direct route to the idea.
- **Do not** recapitulate your personal journey of discovery. This route may be soaked with your blood, but that is not interesting to the reader.
  (Shades of Bob Dylan’s song: Blood on the tracks...
The payload of your paper

Introduce the problem, and your idea, using **EXAMPLES** and only then present the general case
The details: evidence

- Your introduction makes claims
- The body of the paper provides evidence to support each claim
- Check each claim in the introduction, identify the evidence, and forward-reference it from the claim
- Evidence can be: analysis and comparison, theorems, measurements, case studies
Structure

- Abstract (4 sentences)
- Introduction (1 page)
- The problem (1 page)
- My idea (2 pages)
- The details (5 pages)
- Related work (1-2 pages)
- Conclusions and further work (0.5 pages)
The truth: credit is not like money

Giving credit to others does not diminish the credit you get from your paper

- Warmly acknowledge people who have helped you
- Be generous to the competition. “In his inspiring paper [Foo98] Foogle shows.... We develop his foundation in the following ways...”
- Acknowledge weaknesses in your approach
Credit is not like money

Failing to give credit to others can kill your paper

If you imply that an idea is yours, and the referee knows it is not, then either

☐ You don’t know that it is an old idea
☐ You do know, but are pretending that it is yours (very bad)
Structure

- Abstract (4 sentences)
- Introduction (1 page)
- The problem (1 page)
- My idea (2 pages)
- The details (5 pages)
- Related work (1-2 pages)
- Conclusions and further work (0.5 pages)
Conclusions and further work

- Be brief.
IEEE 5 Levels of Plagiarism

1. Uncredited verbatim copying of a full paper.
2. Uncredited verbatim copying of a large portion (up to half) of a paper.
   • Violation notice in article’s bib. record, offender’s IEEE publication privileges suspended <= five years.
3. Uncredited verbatim copying of individual elements such as sentences, paragraphs, or illustrations.
   • Possible violation notice in article’s bib. record, written apology to original creator or offender’s IEEE pub. privileges suspended <= three years.
4. Uncredited improper paraphrasing of pages or paragraphs (by changing a few words or phrases or rearranging the original sentence order).
   • Possible violation notice in article’s bib. record, calls for written apology to avoid IEEE pub. privileges suspension.
5. Credited verbatim copying of a major portion of a paper without clear delineation of who did or wrote what.
   • Requires a written apology, and to avoid suspension, the document must be corrected.
Repeated offenses (and papers)

1. Papers are accepted for review with the understanding that the same work has been neither submitted to, nor published in, another journal or conference.
   - If it is determined that a paper has already appeared in anything more than a conference proceeding, or appears in or will appear in any other publication before the editorial process at IJCCC is completed, the paper will be automatically rejected.

2. Papers previously published in conference proceedings, digests, preprints, or records are eligible for consideration provided that the papers have undergone substantial revision, and that the author informs the IJCCC editor at the time of submission.

3. Concurrent submission to IJCCC and other publications is viewed as a serious breach of ethics and, if detected, will result in immediate rejection of the submission.
The process of writing
The process

- Start early. Very early.
  - Hastily-written papers get rejected.
  - Papers are like wine: they need time to mature
- Collaborate
- Use CVS to support collaboration
Getting help

Get your paper read by as many friendly guinea pigs as possible

- Experts are good
- Non-experts are also very good
- Each reader can only read your paper for the first time once! So use them carefully
- Explain carefully what you want (“I got lost here” is much more important than “Jarva is mis-spelt”.)
Getting expert help

- A good plan: when you think you are done, send the draft to the competition saying “could you help me ensure that I describe your work fairly?”.

- Often they will respond with helpful critique (they are interested in the area)

- They are likely to be your referees anyway, so getting their comments or criticism up front is Jolly Good.
Listening to your reviewers

Treat every review like gold dust
Be (truly) grateful for criticism as well as praise

This is really, really, really hard

But it’s really, really, really, really, really, really, really, really important
Listening to your reviewers

- Read every criticism as a positive suggestion for something you could explain more clearly.
- DO NOT respond “you stupid person, I meant X”. Fix the paper so that X is apparent even to the stupidest reader.
- Thank them warmly. They have given up their time for you.
Remember what you are trying to do...

- Was this the right conference/journal
  - The first Prolog paper was turned down by POPL
- If at first you don’t succeed, try another conference....
- If it really is a bad idea, be glad you were told not to publish it
- Sometimes you need to establish a reputation in an area before publishing.... Write papers for a workshop.
- YOU ARE TRYING TO GET YOUR IDEAS OUT THERE.
Language and style
Basic stuff

- Submit by the deadline
- Keep to the length restrictions
  - Do not narrow the margins
  - Do not use 6pt font
  - On occasion, supply supporting evidence (e.g. experimental data, or a written-out proof) in an appendix
- Always use a spell checker and grammar checker
Visual structure

- Give strong visual structure to your paper using
  - sections and sub-sections
  - bullets
  - italics
  - laid-out code
- Find out how to draw pictures, and use them
Visual structure

The three cases above do not exhaust the possible forms of \( f \). It might also be a `THUNK`, but we have already dealt with that case (rule `THUNK`). It might be a `CON`, in which case there cannot be any pending arguments on the stack, and rules `UPDATE` or `RET` apply.

4.3 The eval/apply model

The last block of Figure 2 shows how the eval/apply model deals with function application. The first three rules all deal with the case of a `FUN` applied to some arguments:

- If there are exactly the right number of arguments, we behave exactly like rule `KNOWNCALL`, by tail-calling the function. Rule `EXACT` is still necessary — and indeed has a direct counterpart in the implementation — because the function might not be statically known.
- If there are too many arguments, rule `CALLK` pushes a `call`

remainder of the object is called the `payload`, and may consist of a mixture of pointers and non-pointers. For example, the object `CON(C \ a_1 \ldots \ a_n)` would be represented by an object whose info pointer represented the constructor `C` and whose payload is the arguments \( a_1 \ldots a_n \).

The info table contains:

- Executable code for the object. For example, a `FUN` object has code for the function body.
- An object-type field, which distinguishes the various kinds of objects (`FUN`, `PAP`, `CON` etc) from each other.
- Layout information for garbage collection purposes, which describes the size and layout of the payload. By “layout” we mean which fields contain pointers and which contain non-pointers, information that is essential for accurate garbage collection.
- Type-specific information, which varies depending on the object type. For example, a `FUN` object contains its arity; a `CON` object contains its constructor tag, a small integer that distinguishes the different constructors of a data type; and so on.

In the case of a `PAP`, the size of the object is not fixed by its info table; instead, its size is stored in the object itself. The layout of its fields (e.g. which are pointers) is described by the (initial segment of) an argument-descriptor field in the info table of the `FUN` object which is always the first field of a `PAP`. The other kinds of heap object all have a size that is statically fixed by their info table.

A very common operation is to jump to the entry code for the object, so GHC uses a slightly-optimised version of the representation in Figure 3. GHC places the info table at the addresses immediately
The passive voice is “respectable” but it DEADENS your paper. Avoid it at all costs.

<table>
<thead>
<tr>
<th><strong>NO</strong></th>
<th><strong>YES</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>It can be seen that...</td>
<td>We can see that...</td>
</tr>
<tr>
<td>34 tests were run</td>
<td>We ran 34 tests</td>
</tr>
<tr>
<td>These properties were thought desirable</td>
<td>We wanted to retain these properties</td>
</tr>
<tr>
<td>It might be thought that this would be a type error</td>
<td>You might think this would be a type error</td>
</tr>
</tbody>
</table>

“We” = you and the reader

“We” = the authors

“You” = the reader
Use simple, direct language

<table>
<thead>
<tr>
<th><strong>NO</strong></th>
<th><strong>YES</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>The object under study was displaced horizontally</td>
<td>The ball moved sideways</td>
</tr>
<tr>
<td>On an annual basis</td>
<td>Yearly</td>
</tr>
<tr>
<td>Endeavour to ascertain</td>
<td>Find out</td>
</tr>
<tr>
<td>It could be considered that the speed of storage reclamation left</td>
<td>The garbage collector was really slow</td>
</tr>
<tr>
<td>something to be desired</td>
<td></td>
</tr>
</tbody>
</table>
10 Principles for Writing Clearly

- Get to the main verb quickly
  - Avoid long introductory phrases and clauses
  - Avoid long abstract subjects
  - Avoid interrupting the subject-verb connection.
- Push new, complex units of information to the end of the sentence
- Be concise
  - Cut meaningless and repeated works and obvious implications
  - Push the meaning of phrases into one or two words
  - Prefer affirmative sentences to negative ones
- Control sprawl
- Above all, write to others as you would have others write to you.
Summary

If you remember nothing else:

- Identify your key idea
- Make your contributions explicit
- Use examples
Last joke

1. Actually, it hasn’t been that bad.
2. It’s slow going, but...
   ...that’s ok.
3. I mean, Rome wasn’t built in a day!
   Tolstoy didn’t write “War and Peace” in one sitting...
4. Actually, it took him six years.
   Six? I’m already behind!

Source: www.phdcomics.com
Thanks to

- Slides based of presentation by Simon Peyton Jones, Microsoft Research, Cambridge
- Klara Nahrstedt, University of Illinois
A Good Paper in Computer Science

When one discovers a fact about nature, it is a contribution per se, no matter how small. Since anyone can create something new (in a synthetic field), that alone does not establish a contribution. Rather, one must show that the creation is better. Accordingly, research in computer science and engineering is largely devoted to establishing the “better” property.