How to write a great research paper: Simon's seven easy steps

Stephanie Weirich
University of Pennsylvania

based on a talk by

Simon Peyton Jones
Microsoft Research, Cambridge

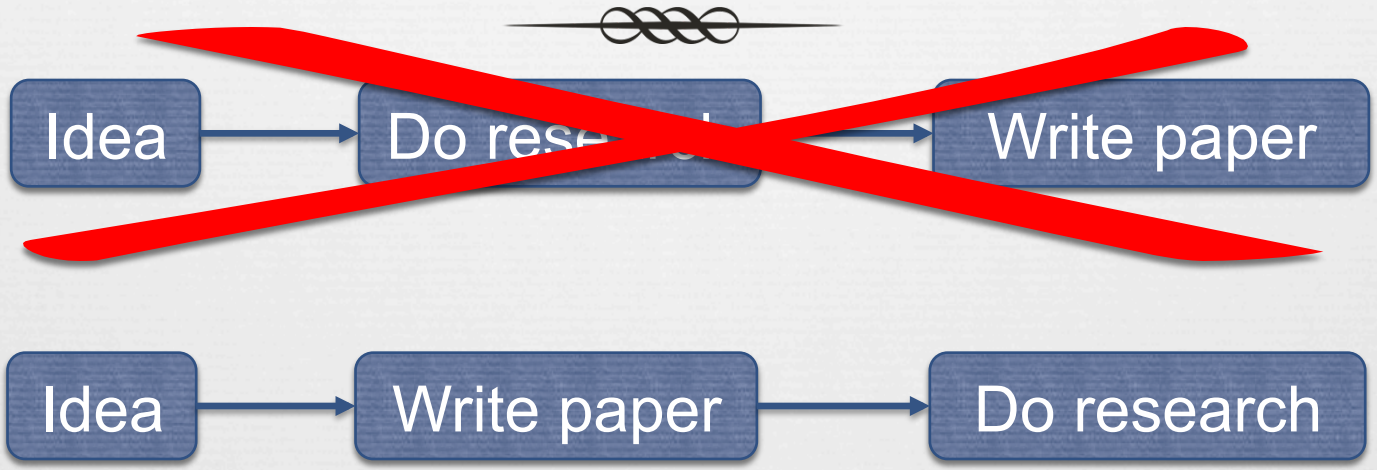
Why write a *great* paper?

- Maybe you don't need to learn how to write a great paper…
  - It takes a lot of time & effort, shouldn't you be doing research?
  - There are lots of bad papers out there, maybe it is not important
- THIS IS WRONG! Good writing is a fundamental part of research excellence.
  - You will get more papers accepted
  - Your ideas will have more impact
  - You will have better ideas

1. Don’t wait: write

Writing papers: model 1

Writing papers: model 2



- Forces us to be clear, focused
- Crystallizes what we don't understand
- Opens the way to dialogue with others: reality check, critique, and collaboration

How to get started

Fallacy You need to have a fantastic idea before you can write a paper. (Everyone else seems to.)

Write a paper about

any idea

no matter how insignificant it may seem to you

Do not be intimidated

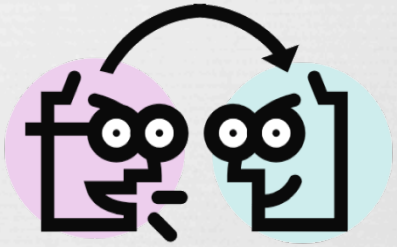
Write a paper about any idea, no matter how insignificant it may seem to you

- **Writing the paper is how you develop the idea in the first place**
- It usually turns out to be more interesting and challenging than it seems at first

2. Identify your key idea

Your paper's goal: convey your idea

- You want infect the mind of your reader with **your idea**, like a virus
- Your idea must provide some re-usable insight to the reader



- Papers are the best way to communicate and record research ideas

The greatest ideas are (literally) worthless if you keep them to yourself

One clear, sharp idea

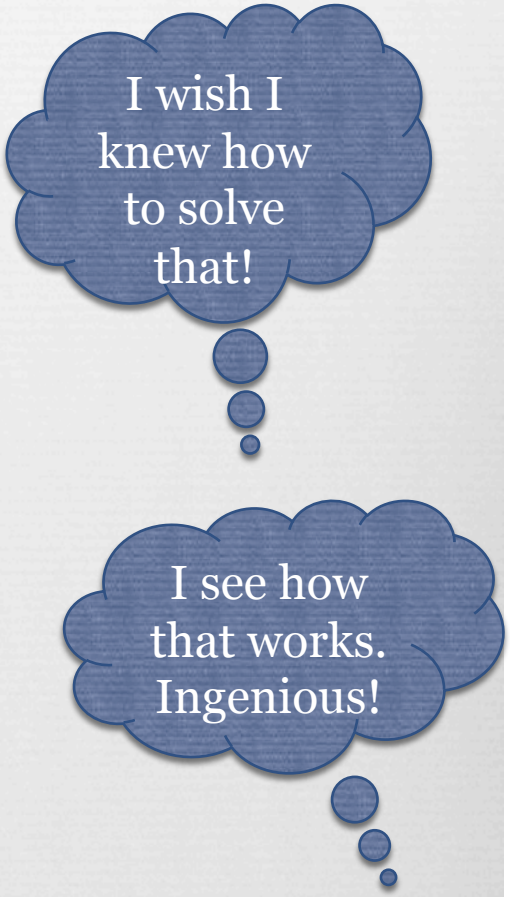
- Your paper should have just one “ping”
- You may not know exactly what the ping is when you start; **but you must know when you finish**
- If you have lots of ideas, write lots of papers

Be explicit about your idea

- Make certain that the reader is in no doubt what your idea is.
  - "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 distill what they are.

3. Tell a story

Your idea is the center of 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

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 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:

```
zipWith :: (a->b->c) -> [a] -> [b] -> [c]
zipWith k []     []     = []
zipWith k (x:xs) (y:ys) = k x y : 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.

Use an example to introduce the problem

Molehills not Mountains

- “Computer programs often have bugs. It is very important to eliminate these bugs [1,2]. Many researchers have tried [3,4,5]. It really is very important.”
- “Consider this program which has an interesting bug. <brief description>. We will show an automatic technique for removing such bugs.”

The introduction (1 page)

1. **Describe the problem**
2. **State your contributions**

…and that is all

ONE PAGE!

4. Nail your contributions

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 "gosh, if they can really deliver this, that'd be 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

Bulleted list of contributions

Do not leave the reader to guess what your contributions are!

Contributions should be refutable

NO!	YES!
We describe the WizWoz system. It is really cool.	We give the syntax and semantics of a language that supports concurrent processes (Section 3). Its innovative features are…
We study its properties	We prove that the type system is sound, and that type checking is decidable (Section 4)
We have used WizWoz in practice	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.

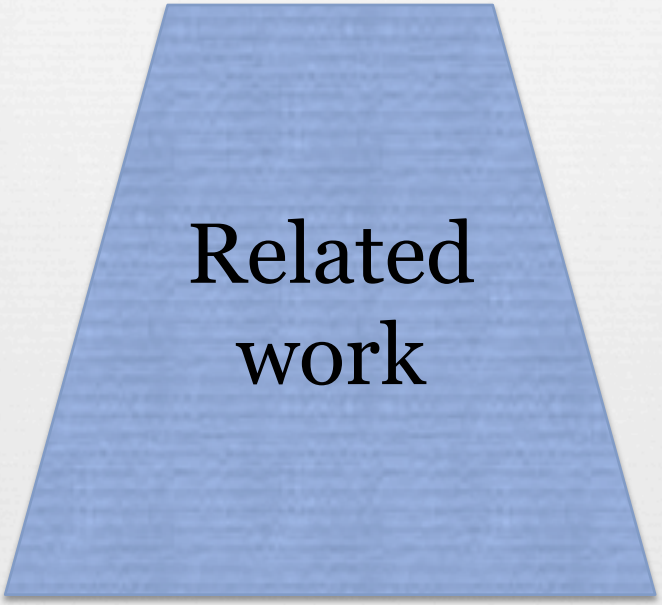
No "rest of this paper is structured as follows…"

5. Related work: later

Structure

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

No related work yet!



Your reader

Your idea

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].

Related work

Fallacy

To make my work look good, I have to make other people's work look bad

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

6. Put your readers first

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

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 $p_j < p_i$.

- Sounds impressive…but
- Sends readers to sleep
- In a paper you MUST provide the details, but FIRST convey the idea

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)
- Even if she skips the details, she still takes away something valuable

Conveying the intuition

Introduce the problem and your idea using

EXAMPLES

and only then present the general case

Using examples

2 Background

To set the scene for this paper, we begin with a brief overview of the *Scrap your boilerplate* approach to generic programming. Suppose that we want to write a function that computes the size of an arbitrary data structure. The basic algorithm is "for each node, add the sizes of the children, and add 1 for the node itself". Here is entire code for `gsize`:

```
gsize :: Data a => a -> Int
gsize t = 1 + sum (gmapQ gsize t)
```

The type for `gsize` says that it works over any type `a`, provided `a` is a *data* type — that is, that it is an instance of the class `Data`[1] The definition of `gsize` refers to the operation `gmapQ`, which is a method of the `Data` class:

```
class Typeable a => Data a where
  ...other methods of class Data...
  gmapQ :: (forall b. Data b => b -> r) -> a ->
```

Example right away

The Simon PJ question: is there any typewriter font?

Put the reader first

- **Do not** recapitulate your personal journey of discovery. This route may be soaked with your blood, but that is not interesting to the reader.
- **Do not** craft a mystery novel, leaving the biggest surprise for the end.
- Instead, choose the most direct route to the idea.

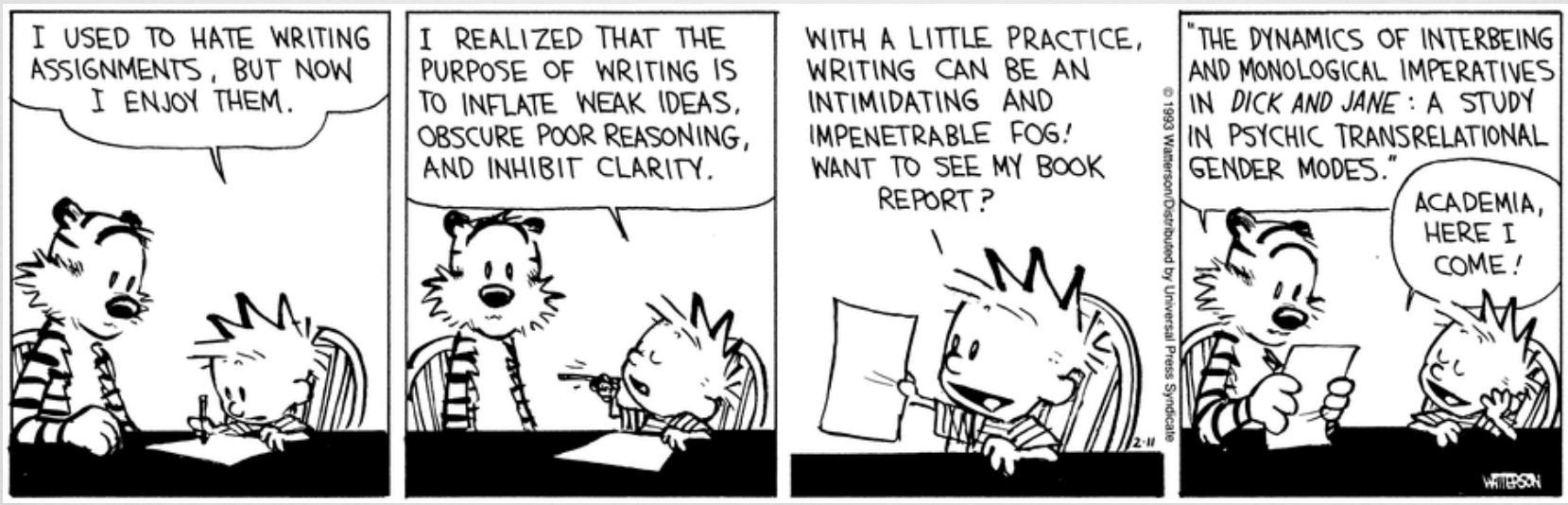
It's not about you

Fallacy

We write papers and give talks mainly to impress others, gain recognition, and get promoted

Fact

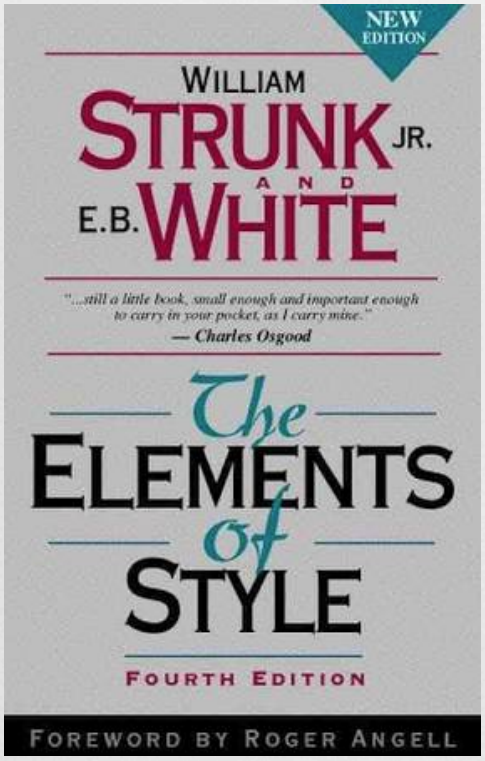
Great papers are influential because they communicate ideas to readers

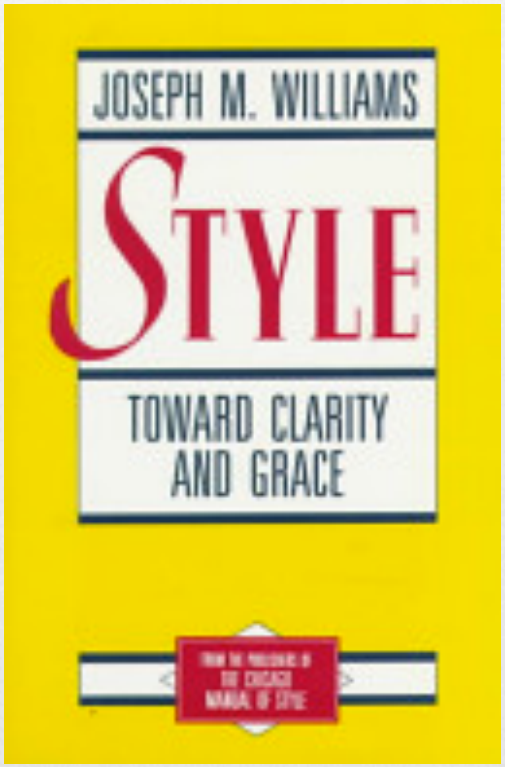


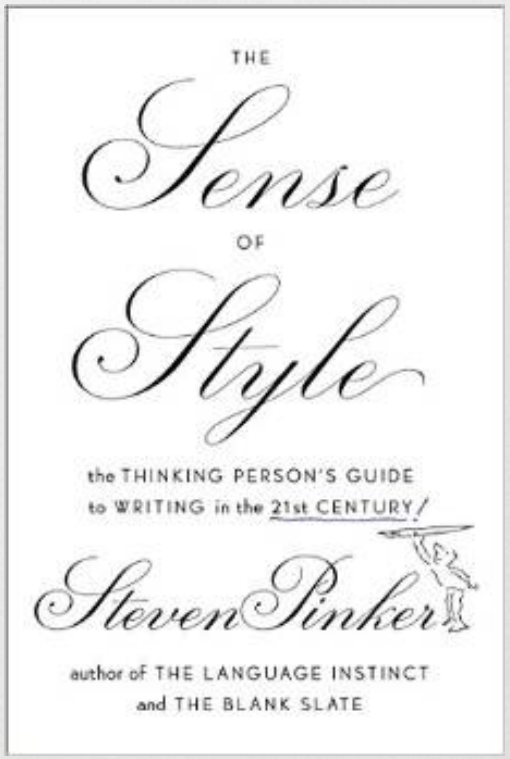
Use simple, direct language

NO	YES
The object under study was displaced horizontally	The ball moved sideways
On an annual basis	Yearly
Endeavour to ascertain	Find out
It could be considered that the speed of storage reclamation left something to be desired	The garbage collector was really slow

Many good resources







7. Listen to your readers

Get help from readers

- Collaborate (via version control software)
- Get feedback from your friends
  - Each reader can only read your paper for the first time once! So use them carefully
- Get feedback from your competitors
  - “Could you help me ensure that I describe your work fairly?”
- Listen to your reviewers
  - Read every criticism as a positive suggestion for something you could explain more clearly

Simon's Steps to Great Writing

1. **Don't wait: write**
2. **Identify your key idea**
3. **Tell a story**
4. **Nail your contributions**
5. **Related work: later**
6. **Put your readers first**
7. **Listen to your readers**