Some Patterns of Convincing Software Engineering Research, or: How to Win the Ernst Denert Software Engineering Award 2020

Lutz Prechelt

Abstract We explain what properties the jury looked for to identify strong contributions and why they are important. They are formulated as seven pieces of advice: (1) Be in scope, (2) Enumerate your assumptions, (3) Delineate your contribution, (4) Honestly discuss limitations, (5) Show usefulness and practical applicability, (6) Have a well-prepared nutshell, and (7) Be timeless.

1 Introduction

This chapter is going to give some insight into the jury's decision process: What we found convincing or not so convincing, and why.

We write this up as much for the benefit of those who want to win this award in the future ("you") as for our own. It helps us to reflect:

- How do our award criteria work out?
- What is the nature of the limitations of works that do not come out victoriously?

It helps you and ourselves getting an idea of some of the relevant psychological dynamics of the jury work.

We write it as a list of separate pieces of advice, formulated as properties that good work should have. The first three we found by counter-examples, the other two by positive examples. We do not name the respective works, but you may be able to identify them by looking at the rest of the book.

We may or may not refine the list in the coming years, e.g. by sharpening the terminology used herein.

Each of the remaining sections covers one piece of advice. Please consider them as a proposer or when you prepare your presentation.

Lutz Prechelt
Freie Universität Berlin, e-mail: prechelt@inf.fu-berlin.de

2 Be in scope

Obviously, to appeal to the jury of a software engineering award, your work must look and feel like software engineering.

In this year's list of eleven works, there was one that we liked *a lot* in many respects, but we felt was too far removed from the technical work in software development. Another, also with some definite strengths, we classified as work in an application domain, not in software engineering as such. Both of these were removed from our consideration quickly.

3 Enumerate your assumptions

Make sure you explain carefully where the ideas and techniques you describe will apply, will likely apply, may apply, and will likely or surely not apply.

The jury is allergic to overselling, to claiming something that is not true. So if we spot a case or area where your techniques do not apply or where we expect they will likely not apply, yet perceive you as claiming they do, we will view your work much more critically.

Therefore, make sure you spell out your assumptions regarding aspects such as

- the target application domain(s) of the software your techniques help building (e.g. hard real-time systems), or the nature of those domains (e.g. regarding the possible precision of specifications);
- the specific technologies that software must be using (e.g. Java), or the nature of those technologies (e.g. statically typed languages);
- the software development process (team sizes, qualification of the people involved, roles, specific practices used or not used, etc.)

The above list is horribly poor and the software engineering community has not yet developed taxonomies that could be used as checklists, so you will have to think yourself what relevant assumptions are underlying your work and shortly explain them to the jury. The characterization can be made positively ("applies if") or negatively ("applies unless").

For several of this year's candidate works, the jury was unsure in its discussion where the work might or might not apply. Such uncertainty will stifle the enthusiasm your work must spark in the jury in order to win.

Works with broad applicability are of course great. Works with narrower applicability can often create much larger improvements – also great. Works with unsure applicability, however, are less likely to convince the jury and several of this year's contributions had a lot of room for improvement in this regard.

4 Delineate your contribution

You will, of course, point out what your research contribution is; how your work advances the state-of-the-art. But the flip side of this is also helpful: By pointing out what is *not* part of your contribution (because it was already there before your work), you will often make it a lot easier for the jury to understand what the contribution really is, and appreciate it.

We had at least two submissions where different members of the jury viewed the amount of originality in the work *very* differently. I suspect the ones with the more negative view recognized some aspects they believed were previously known, perceived them as being claimed to be part of the contribution, and then overlooked or under-appreciated what was really new. Such situations damage your chances.

5 Honestly discuss limitations

We value intellectual honesty. The more you bring of that, the better. Even within the realm of applicability outlined by your assumptions, there will be areas where your techniques work well and others where they work not so well, including cases where they do not work at all. Where the assumptions describe the macro level of applicability, these limitations describe the micro level.

The same advice applies as for assumptions: If the jury spots a limitation you did not mention, our opinion of your work will be damaged a lot more than the limitation itself ever could. Make sure you state all relevant limitations of your work.

6 Show usefulness and practical applicability

One reason why the Ernst Denert award even exists is that Ernst Denert feels academic software engineering work is often overly academic and neglects usefulness, practical applicability, or both. Therefore, a good candidate work will cater for these aspects and argue why the contribution is strong in this respect.

Ideally, it will report in detail on extensive field use and calculate return-on-invest. However, this is almost always unrealistic.

The next best thing would be widely spread pick-up of a tool: If dozens of teams decide the tool is worth using and keep up use over some time, this is also strong evidence that the tool is useful and applicable. This variant is likely applicable only to tools, not to other types of contribution.

The third-best possibility would be a limited field trial describing what about the contribution worked easily or well and what made problems or showed limitations. Such an approach should present initial evidence that the technique in question is indeed useful and the benefits exceed the costs and downsides. Subjective statements

of various participants may be the best you can get, but if done right, they can be convincing.

For contributions that apply to artifacts, applying them in the laboratory to a broad corpus of such artifacts can sometimes be an alternative.

But even that may be impossible: Your work may be groundwork that is not directly applicable. Or it may be insufficiently far developed to overcome many of the practical hurdles against using it in real software development settings. In such cases you need to resort to argumentation: Explain a scenario of how your contribution can be developed into something that is useful and applicable in practice. It will be a lot harder to convince the jury in this way, but if you have done a good job with respect to explaining delineation, assumptions, and limitations, it should be possible.

7 Have a well-prepared nutshell

The importance of the presentation you give to the jury can hardly be overestimated.

Two thousand pages of dissertation text is not something the average (or indeed any) jury member is going to read, let alone understand and remember. One hundred pages of expertises and submitters' rationales are more manageable for a jury member in principle, but are difficult to digest and keep in memory for so many candidates when one is not familiar with the individual works and many of their topic areas. Do not assume any jury member has read any of this material. They may have. But more likely than not, have not.

Therefore, the short presentation you are going to give essentially has to do it all: Motivate your work, explain its techniques, explain its results and their contribution, and address the concerns discussed above to overcome the jury members' diverse biases, preferences, and pet peeves.

It is a super-difficult task. While trying to find a solution, keep in mind the following:

- We are a more generalist audience than the collection of experts and semi-experts you encounter in a typical conference session. Therefore:
- Avoid losing us by explaining technical detail we cannot understand because we lack some of the foundations.
- Avoid losing us by going too fast wherever *any* concepts are involved that some of us do not encounter often.
- Under these circumstances, "telling a story" is the approach most likely to work. If you have never tried this, learn it now; it is highly effective in many situations, in research and beyond.
- On the other hand, you do not need to convince us that you "know your stuff". We know all of the works are very good and their authors very capable.

8 Be timeless

Imagine a time twenty years into the future: What will the role be of your contribution of today in the software engineering knowledge of tommorow?

- Will it be something that had its 15 minutes of fame twenty years ago, but has long become completely irrelevant?
- Will it have been surpassed by improved solutions that address the same problem, but build on your contribution?
- Will it have become a classical tool, hardly changed, but still somewhat useful (like *make*)?
- Will it be a well-known piece of method knowledge, long considered obvious and self-understood (like *refactoring*)?

If your contribution has aspects that you expect to last long, make sure you explain to the jury what those are and why you think so. We appreciate it.