Embedded Systems & Programmer Culture

PhD course on Research Methodology, Ethics, and Innovation for Computing Disciplines
Patrik Persson, June 20, 2016

In this essay, I'll give some reflections on (what I perceive to be) research methods in my field, embedded systems design. I'll note some potential weaknesses in our ways of working and suggest how to address those weaknesses.

What we do now

Our group's web page (Embedded Systems Design) says:

> The big challenge of this research is to find efficient methods and tools which can be used in industry to enhance a design process, make time-to-market shorter and improve overall quality of the final products. [1]

Broadly speaking, we design and evaluate hardware and software constructions to make electronic gadgets cheaper and more efficient. Much work concerns hardware/software co-design, and results can include both software (algorithms) and hardware (e.g., accelerators). Like many computer scientists, we're interested in mathematically oriented methods for describing and analyzing problems.

Our work could perhaps be categorized as design science, and includes the following:

1. **Select** an architecture (typically an existing one). This could be, say, an operating system, or a programming language.
2. **Identify** an application for the given architecture, and identify cases where the application runs sub-optimally on this architecture.
3. **Propose** and construct a new or modified component, such as a scheduling algorithm, or a hardware accelerator.
4. **Evaluate** the new component with respect to the application in step (2), and possibly other benchmarks. We typically use quantitative measures of performance or power.

Steps 1–2 concern our choice of problem, and steps 3–4 concern our method to address that problem. In my work, step 1 could be an Android-based mobile phone, step 2 could be an imaging (photo) application overheating the phone in some cases, step 3 could be a revised, temperature- and power-aware task scheduler for imaging algorithms, and step 4 could involve power, heat, and performance measurements with/without the proposed scheduler.

A couple of points are worth noting here. First, I've presented the choice of problem and the research method together. We sometimes seem to prefer to think that we *first* start with a problem and *then* select our method; however, I'm convinced we frequently choose problems that allow us to apply our favorite methods. I think this phenomenon is not specific to this field, but rather to engineering research in general, and (to some degree) even all research. There's not necessarily anything wrong with this concept of ”method in search of a problem”; we select problems we think we can do something good with. But I think it's important to remember that we often know which method, and perhaps which architecture, we prefer *before* we know which problem to solve.

Second, we are the ones to both (3) propose a solution and (4) evaluate it. In contrast to a natural scientist, whose empirical work concerns a reality outside of its observer, our empirical work concerns our own constructions. There is an obvious risk for bias, since we usually *wish* our solutions to be good. Our preconceptions, methodological preferences, and wish for success may—perhaps unconsciously—nudge our conclusions in the right direction.

What we could do

So, I see two potential problems: (1) we may select architectures depending (partly) on what kind of research we like to do, and (2) our subjective views risk influencing our evaluation of our own work.

Problem (1) typically concerns programming models—the ways we like to express our ideas. A programmer would choose a particular operating system, programming language, or framework that fits his or her needs. Those needs often involve a fair dose of subjective views. Is object-oriented or functional programming better? Android or iOS? Threads or dataflow? OpenCL, Cuda or RenderScript? Universal, objective criteria can certainly be found in many cases, but far from all. It often boils down to individual programmers' preferences.

I think it could be useful to apply case study methods here. Can we learn how programmers like to express problems? Can we learn something, for example, about which problems are preferably expressed in an object-oriented form, and which in a functional one? We usually say that some problems lend themselves to one expressions, and some the other; however, Tedre and Sutinen [2] convince me that different computer science traditions view these things differently. Preferences depend not only on the problem, but also on the programmer and the programmer's teachers; programmer preferences need to be weighed into our selection of architecture. To understand those preferences, we would need to move into case study research.

Problem (2) above, of our wishes influencing our results, is trickier. To some extent it is addressed by choosing universally accepted benchmarks; still, those benchmarks reflect the choice of architecture, and, by extension, the programmer preferences discussed above. One metric never says it all—it's always based on an assumption of what to care about.

I argue that the research cannot be entirely separated from the researcher. We certainly strive towards the objective, and this is precisely why we should be good at understanding the subjective. Embracing the subjective does not mean that we give up on the objective: we can both agree that the grass is green, even if I'm not sure your green is the same as mine.

References

[1] Embedded Systems Design, Dept. of Computer Science, LTH: http://esd.cs.lth.se. Retrieved on June 6, 2016.

[2] Tedre, Matti, and Erkki Sutinen: ”Three traditions of computing: What educators should know.” *Computer Science Education* 18.3 (2008): pp. 153–170.