Why Andreas writes suboptimal code and why this hinders scientific research

Antonis Christofides[*]

National University of Athens, Greece

A presentation given at a meeting of Python programmers
Athens, Greece, 11 May 2014

I avoid talking about myself when making presentations, but this is a presentation about people. So we will talk about me and about Andreas, of course. You can see some things about me in Figure 1. I am primarily a computer guy, with significant education and some experience in hydrology, which is a branch of civil engineering. Sometimes, at the end of the day, when the work is done, when pythonmode indicates zero errors and zero warnings, when the documentation is written and good, when all unit tests run in Python 2 & 3, in Linux and Windows, and when everything is committed and pushed, I look at my code, which is beautifully coloured by the editor, and I feel I'm a great programmer. And then an email comes informing me that a Django core developer has written a patch for a Django bug that has hit me and which I'm monitoring; and I look at his patch and I think I suck. These are good feelings, because it is through this procedure that we learn, and after 5 or 10 years we can also be good core developers in something.

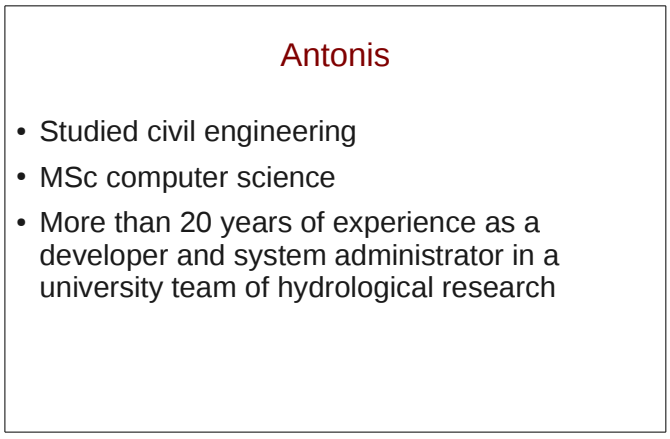

Figure 1

But our protagonist here is Andreas (Figure 2). He is a hydrologist who has dedicated his life to scientific research. He isn't among those who work for the prestige or for the money or for the beautiful female students; instead, he serves science as the search for truth. He is as smart as you and I. Needless to say, I am not concerned specifically with Andreas, but with university researchers, but I like to talk with concrete examples.

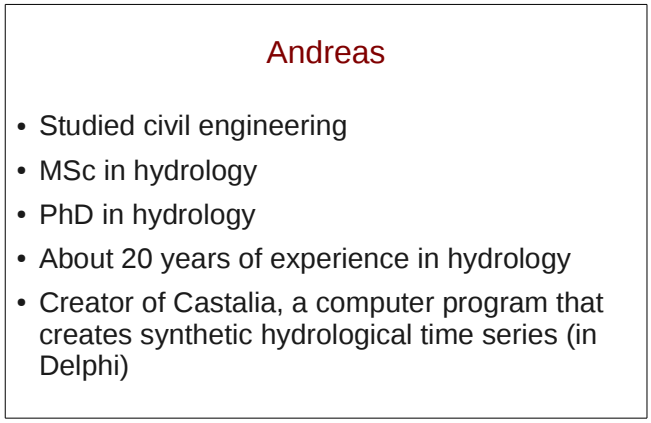

Figure 2

Like many researchers, Andreas also can program. He implements complicated algorithms that have to do with his research.

Andreas's narcissism applies to his theories, his equations, his scientific papers. He reads many scientific papers written by others, and I guess that whenever he comes across a particularly good one, he feels admiration for the author, and this is how he also improves himself. He doesn't give a damn about the code of the Django core developer.

I would show you some of his code here, but I prefer to sensor it (Figure 3) (I have the sad feeling, however, that Andreas' code is better than most of the code written by professional programmers, and that most of you have probably seen much worse code than his; but this doesn't change the fact that it's seriously suboptimal.)

There is also another reason Andreas's code is suboptimal. More than 10 years ago, I convinced that research team to start using CVS. A few years later I switched them to subversion. A few years later I switched them to Mercurial. On that occasion, the

[*] anthony@itia.ntua.gr

three members of the team who were programmers did learn Mercurial. Andreas did not seem to intend to learn it, and when I asked him why, he said, half-jokingly, "because you are going to change it again". Indeed, in 2013 I switched everything to git. But Andreas was not merely right; his statement goes right into the heart of the problem. Contemplate the things you have been learning the last three months: it's probably overwhelming. Andreas simply does not have the time to learn a fifth of that, because he has other work (Figure 4).

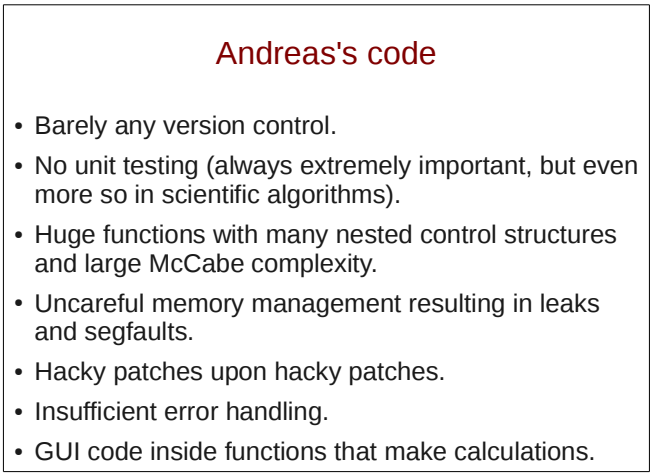

Figure 3

Andreas will thus never be a good programmer in this life, for pretty much the same reason you will never be a good hydrology researcher in this life.

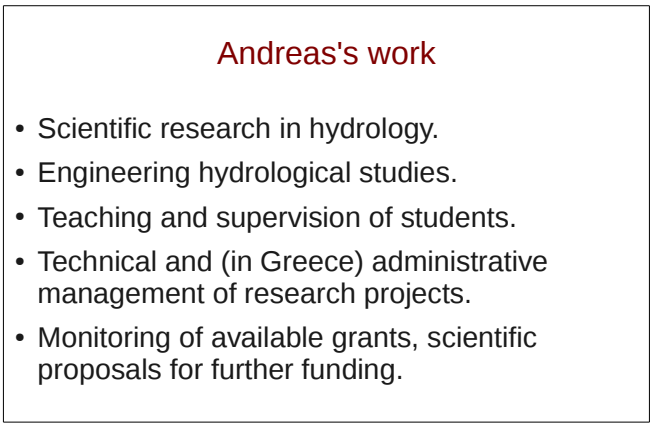

Figure 4

The most important consequence of suboptimal code, in this case, is that it is an obstacle to the dissemination of scientific progress (Figure 5).

What is the solution? The most obvious, you will tell me, is to let Andreas write prototypes, and leave the actual programming to programmers. Unfortunately, this doesn't work very well. Castalia is based on several scientific papers (Figure 6), which have fairly advanced math and concepts. Andreas and Demetris (Andreas's supervisor) told me they estimate that if I did heavy reading and worked closely with them so that they could help me, it would take me about one month to understand the whole thing. They were talking specifically about *me*, not you. I've been working with them for 20 years already, so I already have some understanding of the concepts. I believe it would take you much more than it would take me. I also have the feeling that their research is relatively easy to understand; research based on more advanced math could require years to learn.

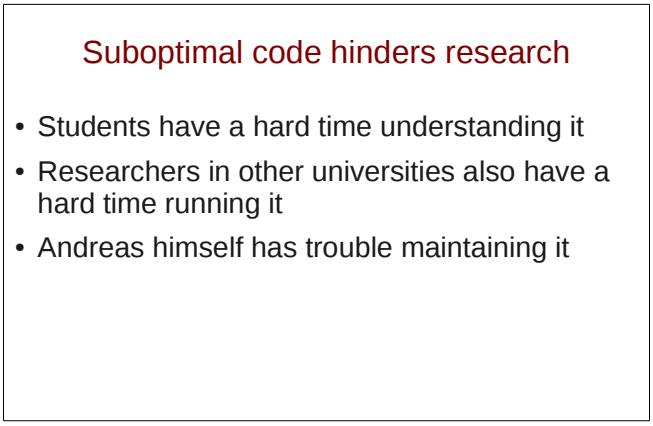

Figure 5

In any case, suppose we find the funding to do this: to have me or you study the issue alongside Demetris and Andreas, and rewrite the code in a clean Python + numpy + pandas (+ Cython) application. Finding such funding is already hard to do because of the way research funds are managed, but suppose we do it. How is it going to be maintained? How is Andreas going to continue his research? He will need to learn Python+numpy+pandas to develop new prototypes that build on the previous work. Assuming funding is secured so that the programmer continues to work alongside with them in order to maintain the program and to help Andreas learn Python 4, the new revision control system, nose, and all such small changes in the workflow that will be needed every now and then, it will work for some time.

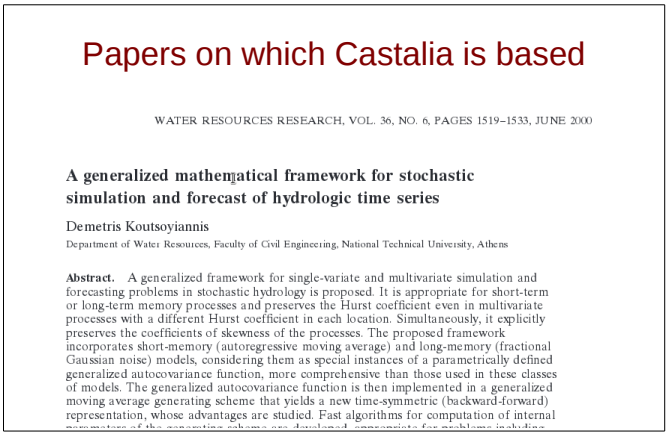

Figure 6

But, in five or ten years, it is likely there will be a new language, say Viper, that is particularly suited for such research. It will be compelling to rewrite the whole thing in that language. Python + numpy + pandas may be looking as obsolete as Delphi does now and as FORTRAN did in the 1990s. Andreas will already be having enough trouble to keep up with Python 4 and all the new modules I will be introduc-

ing all the time; if I tell him that I will rewrite the whole beast in Viper, he will kill me. He will also bring in his former students to help him. Some of these former students will now be professors in other institutions abroad, and they will have their own students. How will all these people learn Viper if they aren't working closely with me?

You see it's not a trivial problem. It is, I think, the main reason why FORTRAN is still being used today. It's just not that simple to change.

The main tendency today is to have private companies of programmers offer support to universities; but I think that this is too expensive and does not work well; a programmer that is part of the research team but has significant interaction with other programmers is way better. But I explained that this, also, does not work very well. I don't have an answer.