The Pragmatic Scientist

Arjun Krishnan, Associate Professor of Biomedical Informatics, CU Anschutz
Email | Website | Social

The Pragmatic Programmer [Copyright 2000, Addison Wesley] is a great book and a reliable reference on software engineering that I find myself going back to regularly. The authors, Hunt & Thomas, conclude the book with a List of Tips that summarizes its key ideas.

During a recent trip to the book, I realized that many of the tips in this list have general counterparts that go beyond software development in being valuable lessons for doing science, in general, and computational science, in particular.

So, I have picked, adapted, and generalized those practical tips here (below). I hope for this to be useful to graduate students, postdocs, & other scientists in training.

You and your science

1. **Care About Your Craft**: Why spend your life doing your project(s) unless you care about doing it well?
2. **Think! About Your Work**: Turn off the autopilot and take control. Constantly critique and appraise your work.
3. **Remember the Big Picture**: Don't get so engrossed in the details that you forget to check what's the goals and impact of your whole project/endeavor.

You and your learning

1. **Invest Regularly in Your Knowledge Portfolio**: Make learning a habit.
2. **Critically Analyze What You Read and Hear**: Don't be swayed by high-profile papers, media hype, or current dogma. Analyze information in terms of you and your project.
3. **Don't Repeat Yourself**: Every piece of knowledge must have a single, unambiguous, authoritative representation within a system.
4. **Create and Use a Project Glossary**: Create and maintain a single source of all the specific terms and vocabulary for a project.

You and your (computational) scientific practice

1. **Explore & Prototype to Fail Fast and Learn**: Exploration-and-prototyping is a learning experience. Its value lies not in the code/plots you produce, but in the lessons you learn.

2. **Don't Just Think It – Prove It**: Don't wring your hand and speculate. Work a small example that reveals, confirms, or eliminates something.
3. **Estimate to Avoid Surprises**: Plan and estimate before you start. You'll spot potential problems up front. Update your plan frequently as you go.
4. **Provide Options, Don't Make Excuses**: Instead of excuses, provide options. Don't say it can't be done; explain what can be done.
5. **Fix the Problem, Not the Blame**: It doesn't really matter whether the mistake is your fault or someone else's. It's still your problem, and it still needs fixing.
6. **Design to be Continuously Functional and Testable**: Write code that is tractable and testable every step along the way instead of once at the end.
7. **Eliminate Effects Between Unrelated Things**: Design code and analysis that are self-contained, independent, and have a single, well-defined purpose.
8. **Don't Live with Broken Windows**: Fix bad designs, wrong decisions, and poor code when you see them.
9. **Don't Use Wizard Code You Don't Understand**: Wizards can generate reams of code. Make sure you understand all of it before you incorporate it into your project.
10. **Don't Think Outside the Box – Find the Box**: When faced with an impossible problem, identify the real constraints. Ask yourself: "Does it have to be done this way? Does it have to be done at all?"
11. **Don't Be Bound to Formal Methods**: Don't uncritically adopt any technique without putting it into the context of your project and your capabilities.

You and your community

1. **Be a Catalyst for Change**: Share new ideas, concepts, and software with your peers & colleagues, and help in lifting-up the folks around you and create a vibrant, supportive community.
2. **It's Both What You Say and the Way You Say It**: There's no point in having great ideas if you don't communicate them effectively. What you say also needs to invite folks in, excite & educate them, and help build trust.
3. **Make It Easy to Reuse**: If your approaches, methods, and code are easy to reuse, people will, both within and beyond the lab.