How to be an Efficient Researcher

Dylan S. Shah[1]

***Abstract*— This document gives my advice on how to efficiently conduct research involving already-purchased electronics hardware. The research approach advocated here can be applied in other areas of research ("systems"). Sometimes it makes sense to skip or return to steps, so it should be drawn as a highly-connected undirected graph. My philosophy shares many values and characteristics with Agile project management, especially Scrum, but I do not ascribe rigidly to any formal framework. Note that this is provided as-is, with no claims of optimality or assumption of liability. It's clearly not a professional business-school document, just one researcher's views.**

Main steps: 1) Know your goals. 2) Draw a visual representing your ideal system. 3) Refine your plan with a trusted colleague. 4) Build up your system from "easily"-understandable chunks. 5) Refine your final system with your colleague(s), until you met your goals. Effective communication will improve your enjoyment and efficiency of research.

CONTENTS

I. Efficient Programming (Read this First)

First, a procedure for writing code efficiently. orange text is an example 1-program project; more complicated programs might require multiple tiers of purposes, design requirements, psuedocodes, etc.

1) "Know thyself" - State your:
   - Project Purpose. Change outputs of 8-ch board between GND and +VDD upon command
   - Design Requirements. Be the size of a cellphone (or smaller), and be I2C-ready
   - Desired Features. Provide automatic shutoff when current exceeds 1A
2) Draw your program, including flow (flow of information, power, etc.) and/or interactions - between hardware components, as well as between significant software components/objects/functions/programs.
   Arduino $\xrightarrow{\text{States}}$ PIC μC $\xrightarrow{\text{States}}$ FET $\xrightarrow{\text{Current}}$ Device
3) Write Psuedocode
   PIC μC Firmware:
   1. Receive command
   2. Set States
4) Review your psuedocode (in M.E. this is a "design review") with a trusted coworker (labmate, boss, etc.).
5) **Repeat steps 1-4 until you and your coworker agree on the plan**.
6) Write skeleton script and run it. (The minimal program which can run in your PCB and/or IDE, such as the program shown when you open the Arduino IDE.)
7) Make minimal "testing driver" to test basic PCB functionality (examples: "Blink", "capSensorDriver"). Save this in a location where you won't lose it.
8) Add functionality in minimal increments, testing each step. (Also make a git commit after each significant functionality addition.)
9) Code review with coworker. This serves three purposes: 1) Organizing your work. 2) Catching bugs or inefficencies. 3) Improving collaboration (as well as code-sharing to future users of your project).
10) **Repeat useful steps until you met requirements**.
11) Optionally, optimize your code for computational efficiency
12) Show your code to a labmate, to ensure readability and verify that you have proper documentation.

II. Other Coding Tips

C.f. "Best Practices for Scientific Computing" by Wilson (2014) [1], "MATLAB Programming Style Guidelines" [2], and the Google Python Style Guide [3].

1) Write programs for people, not computers.
   - Use meaningful function/variable names
   - Directly in the document (if possible), concisely document: Why & how, not what; purpose & design, not mechanics. Every "major" function or action needs a comment & whitespace at a minimum.
2) Use version control (git).

*Contact Info: dylan.shah.50@gmail.com and dylanshah.com.

3) Use high-level languages (Ex: Python vs. C).
4) Use a single full-feature text editor for all editing, and learn keyboard shortcuts. My preference, first to last: Atom or Sublime; Notepad++; VIM in terminal.
5) Code in Linux when you're programming for more than $\tilde{3}0$ minutes. It has many built-in features that make coding efficient and enjoyable. (Exception: some programs, such as MATLAB, have better Windows implementations.)

III. Git

Git is an efficient way of tracking file versions (drawn in Fig. 1; see [4] for details). It is a command-line program (or you can use Git Desktop) that can turn almost any folder into a tracked folder. GitHub stores these folders online. Somebody pushing to GitHub CANNOT affect your local files unless you let it, typically with `git pull`. Many programs have Git integrations, including Sublime and "Sublime Merge" (has great graphical tools), Atom, and MPLAB. StackExchange and ChatGPT have answers to most situation-specific questions.

Basic workflow:

- Optionally pull changes from remote. `git pull`
- Edit files to complete a significant feature or bugfix
- Add files to index. `git add .`
- "Commit" your index to commit history. `git commit -m ''Added error checking''`
- (Push changes to remote. `git push`)

IV. Coding Languages

Here, level means "level of abstraction". High-level code requires fewer lines of code to achieve a given function and is typically easier to read and write. **Bold** languages are essential for modern engineers.

Language	Purpose	"Level"
Python	General	High
C++ (Arduino)	Fast, general	Medium
MATLAB	Plotting	High
Assembly	Fastest	Low
R	Statistics, Plotting	High
.md, LaTeX, HTML	Documentation	High

V. Debugging

Debugging is an art. Over time, you will be able to skip and re-order these steps. Retest at various stages. Important: unplug all energy supplies (electricity, pressurized air, etc.) prior to changing or inspecting wiring, or reorienting parts. Follow lab safety guidelines. Add fuses or breakers.

1) Test your system with the **simplest known software that previously worked**. (I.e., your testing driver from earlier.) If it runs properly, you have a software issue: use a code compare tool ("Code Compare" in Windows; `git diff` in any OS) to see changes made between your simple code and the problematic code, and skip to the final step below.
2) Draw a **high-level schematic** of your *ideal* circuit (Arduino, PC, Pressure Regulators, etc.).
3) Make sure all your *real* system's **grounds** (GND) are properly connected.
4) Make sure all **power supplies** are: connected where they're supposed to be; set to the proper voltage (and max. amps. allowed).
5) Make sure all **communication wires** (I2C, USB, etc.) are connected properly and securely.
6) **Remove components** that were added since the most recent "previously working" configuration.
7) **Remove components** until you get to a configuration that can run simple code (your testing driver). Add components (and code) incrementally so you can effectively verify the operation of each component individually.
8) **Verify potentially problematic components with a simple circuit which is known to work**. Ex: test a pressure regulator (PR) by using the pressureRegulatorDriver on an Arduino, connected to one working PR. You should hear the working PR tick periodically. Add the questionable PR and observe.

VI. Reading Papers

For most papers, everything you need to learn can be presented in a single paper (front and back) of handwritten notes on line-free paper. Take notes for complicated papers; otherwise, highlighting the PDF is typically easier to manage. Use a reference manager such as Zotero. I'll abbreviate a "well-written, significant paper" (see [5], [6]) as a "WWSP".

1) Skim the paper: abstract, intro, figures, tables, key theorems, and conclusion. 15-30 minutes
2) Decide which sections are most relevant to you.
3) Read the full paper, taking notes on: assumptions, experimental flaws (and their consequences), key formulas (quasi-independently derive all formulas found in the sections identified previously), and results/conclusions. Don't spend more than $\tilde{2}0$ minutes on a single topic or formula: add it to a "TODO" sticky note or paper. Varies. A WWSP might take a full day.
4) **Learn:** read the SI & related work, Wiki pages, discuss with labmates, etc., addressing your TODO list. Varies. A WWSP might deserve a few days.
5) Organize your notes so you can quickly remember/locate key details in the future.

VII. Writing Papers

See [7], [8]. Basically, draw your paper by hand: write bullet points for your main claims and differentiation from prior art, sketch figures, make blank data tables, write experimental procedures, draw the data-flow and/or circuit. If significant, list necessary resources, and estimate the project timeline. For a conference article, aim for 1 paper front-and-back; for a journal paper maybe 3 papers front-and-back. While this section was written with academic papers in mind, the basic process generalizes well to most types of reports.

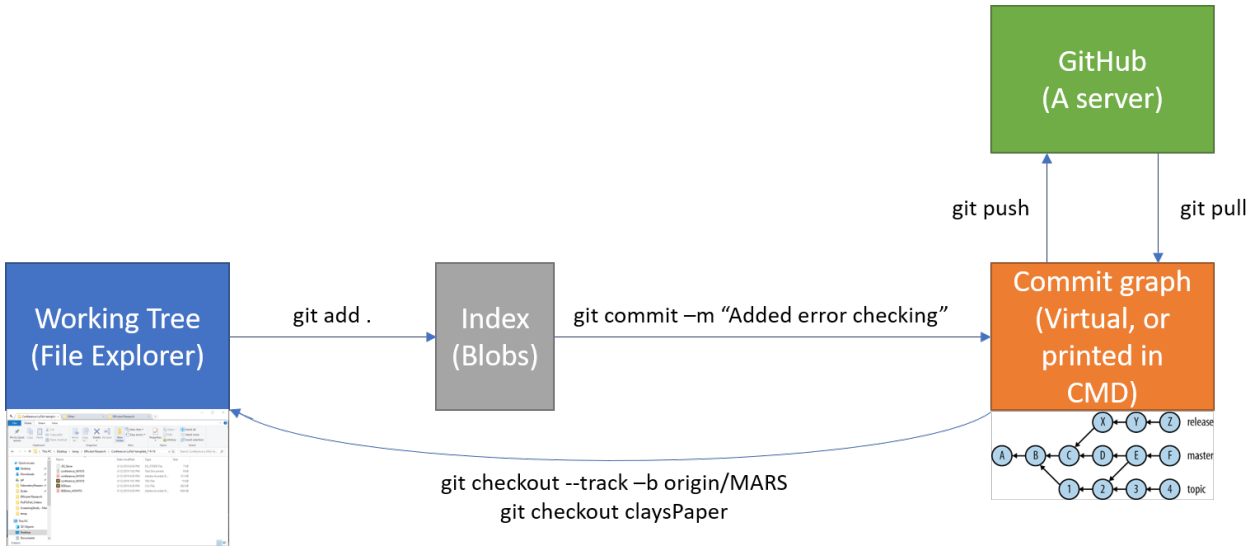

Fig. 1. A graphical explanation of git and GitHub. Black text are example commands which pull information from their source to the arrow-tip.

Then, complete the proposed research, continuously updating your outline (switching to Overleaf at some point). Finally, process all data, collect images, draw schematics (in Inkscape or Adobe Illustrator... NOT PowerPoint), and write the full paper in rough-draft form. Properly frame your work and explain the full story. Coherency and context go a long way to increasing the readability and impact of the work [8].

Show the first draft to your project PI and co-authors in person, until you agree on the overall structure. Once the structure is set, refine the paper to be presentable. Send a presentable draft to co-authors and revise. Once you agree on the draft, send to at least one “senior” writer in the group (a prolific 2nd-year, a senior grad student, etc.). Once your co-authors agree on the revision, send to the project PI. As you get closer to submission, begin drafting a cover letter (using project PI’s template .tex file).

Notes for revisions: minimize the amount of quotes reproduced in the Response to Reviewers. Write a cover letter for the revision, if you’re submitting to a “high-impact” journal. Show PI the revision in person for a minute to make sure you’re agreeing on the overall structure of the changes. Then treat the revision and cover letter like a new submission (see previous paragraph).

A checklist to look through during the submission process is added as an appendix.

VIII. Reviewing Papers

This is my process of reviewing a paper, after I already accepted the invitation. See [9] for opinions on how to decide whether to accept an invitation. I read the paper’s PDF in the following order, taking notes on a Markdown file and highlighting the PDF:

1) **Abstract**: look for coherency, relevance to the field, and a concise summary of their claims, methods, evidence, and conclusions.
2) **Figures & Tables**: verify that the presentation is clear and well-captioned. Figure quality (both technical content and organization) highly correlated with the rest of the paper. Good papers (some theory papers are exceptions) can be 80% understood just through the figures.
3) **Formulas**: skim the formulas to get a overview of the theory they are operating under. Typically the theory will be used to justify their broader claims.
4) **Complete read-through**: read the entire paper, word for word.
5) **Math Check-up**: verify all derivations and equations by hand, with minimal hints from the paper.
6) **Convert all comments to text**. I use a “.md” file, because it can be color-coded in Sublime. The publisher’s website will present the authors with the plain text. My sections are: “To the Editors” (Confidential comments, as a super-condensed gut feel. Optional.), “General Comments”, “Detailed Comments”, “Figures”, and “Supplemental”.

References

[1] G. Wilson, D. A. Aruliah, C. T. Brown, N. P. C. Hong, M. Davis, R. T. Guy, S. H. D. Haddock, K. D. Huff, I. M. Mitchell, M. D. Plumbley, B. Waugh, E. P. White, and P. Wilson, “Best Practices for Scientific Computing,” *PLOS Biology*, vol. 12, p. e1001745, Jan. 2014.
[2] R. Johnson, “MATLAB Style Guidelines 2.0,” 2014.
[3] Google, “Google Python Style Guide,” 2019.
[4] R. E. Silverman, *Git Pocket Guide: A Working Introduction*. O’Reilly Media, Inc., 2013.
[5] J. Hiller and H. Lipson, “Dynamic Simulation of Soft Multimaterial 3d-Printed Objects,” *Soft Robotics*, vol. 1, pp. 88–101, Feb. 2014.
[6] W. M. v. Rees, E. A.Matsumoto, A. SydneyGladman, J. A.Lewis, and L. Mahadevan, “Mechanics of biomimetic 4d printed structures,” *Soft Matter*, vol. 14, no. 43, pp. 8771–8779, 2018.
[7] G. M. Whitesides, “Whitesides’ Group: Writing a Paper,” *Advanced Materials*, vol. 16, pp. 1375–1377, Aug. 2004.
[8] J. Strassmann, “Why this editor wont be sending your paper out for further review at PNAS,” Apr. 2015.
[9] E. Pain, 2016, and E. Pain, “How to review a paper,” Sept. 2016.
[10] “Science Robotics, Information for Authors-Research Articles,” Feb. 2018.
[11] “Effective Writing | Learn Science at Scitable.”

S1. Quotes / Maxims

These quotes are concise expressions of what I feel are generally-applicable truths. The explanations show how I interpret them in the context of engineering research.

1) **"Solvitur ambulando"** — Latin. Solve by walking: if you are stuck on a difficult problem, take a walk outside and return to the problem.
2) **"Do not solve equations, analyze them"** — Yale professor Madhusudhan Venkadesan. Presumably referencing L. Mahadevan. You can typically learn more by looking for the relation and effects of key variables than by treating equations as math that you need to "solve".
3) **"You will not get what you deserve. You will get what you negotiate."** — author Chester Karrass. Remember this when applying for jobs, grants, fellowships.
4) **"Never eat alone"** — some guy on a cruise. Most networking and great ideas start over meals and breaks. (Follow-up calls, brainstorming, etc. are second steps.)
5) **"Everything should be made as simple as possible, but not simpler."** — Albert Einstein. A related idea is that you only truly understand something when you can show somebody else that it is simple (a flowchart, equation, or diagram is typically necessary).
6) **"If it feels difficult, you're probably doing it wrong."** — Dylan. IK, it's weird to quote yourself. However, I can't find an equivalent quote online, but live by this. Once you know how to do something, even if it's complicated and difficult, you should be able to find a way to make it not feel difficult. Otherwise, you're probably in the wrong line of work.

S2. Presentation feedback

1) Start with a story, fact, or question (difficult in academic presentations).
2) Put a bar on the bottom darkly (black) showing which section you're in, and show other sections in a faded (gray) font.
3) Add slide numbers.
4) Make the photos higher-resolution. Also, for that snip of the equation from XYZ et al., zoom in more before taking the snip. Alternately, write the equation in PowerPoint's equation writer.
5) **Provide more context in the introduction. Why does anybody in the room care about the work?**
6) Have more concise text.
7) Sections should typically be roughly:
   - Motivation (including prior work as a subset)
   - Basic principles of your technology
   - Visual summary of your work
   - Important details (theory, experiments, PCB's)
   - Interesting results (successes AND useful or instructive failures)
   - Conclusion
   - Thank you and questions
8) Use a consistent slide format. I prefer a simple template, such as the ones used in the Presentations section on my website.
9) I was lost in the theory section. Analyze the importance of only the most significant equations.
10) Use language and a level of detail that is appropriate for your audience.

S3. Paper feedback

1) Abstract should be a single paragraph, approximately 200 words, written in plain language. Do not include citations or undefined abbreviations in the abstract. It should include [10]:
   - An opening sentence that states the question/problem addressed by the research
   - Enough background content to give context to the study
   - A brief statement of primary results
   - A short concluding sentence.
2) **Provide more context in the introduction. Why does anybody care about the work?** Intro should include:
   - An opening sentence that states the question/problem addressed by the research. Continue with related potential benefits of solving the problem, and/or how it is currently solved.
   - Enough background content to give context to the study, including weaknesses/gaps in prior research. Typically 1-2 paragraphs.
   - A brief statement of your primary results. Typically one paragraph.
   - A short concluding sentence that inspires readers.
3) First figure should be a summary figure that clearly shows the advance presented in the paper
4) For a high-impact paper (in a top-tier journal/conference), sections should typically be (theory may be interspersed, depending on its relative significance):
   - Introduction (including prior work as a subset)
   - Results
   - Discussion
   - Conclusion
   - Materials and Methods
   - Supplementary Information
5) Abstract, intro, and final section (discussion/conclusion) should follow a similar format and flow.
6) Have more concise text.
7) What do you mean when you say "this"?
8) Use terms consistently. Ex.: stiffness vs. rigidity; shape-changing vs. shape changing vs. shape change.
9) I was lost in the theory section. Analyze the importance of only the most significant equations, and remove other equations and discussions.
10) Only introduce abbreviations if that term is either used several times (> 5) or in several sections (> 3).

11) Use language and a level of detail that is appropriate for your audience.
12) Tenses [11]:
    - Past tense = what happened in the past: what you did, what someone reported, what happened in an experiment, and so on.
    - Present tense = general truths, such as conclusions (drawn by you or by others) and atemporal facts (including information about what the paper does or covers).
    - Future tense = perspectives: what you will do in the coming months or years.
    - Most of your sentences will be in the past tense, some will be in the present tense, and very few, if any, will be in the future tense.
13) Use consistent, SI-ish units. Spaces between numbers and units (exception: percent$\%$, and degree$^\circ$).

S4. Misc Notes

Compose vs. comprise link. Comprise: include or contain. Compose: to be or constitute part or element of. Use composed of. Not comprised of. Comprising == non-exhaustive list. Composed $\triangleq$ Consisting == exhaustive list. Examples: The whole comprises the elements or parts. The elements or parts compose the whole.

S5. Final Submission Checklist

Check your venue's submission guidelines. Some of the items below will not be necessary. Roughly in the order you should complete them.

1) Get the submission link (if it's a special issue)
2) **Write cover letter in .tex, give to your PI**
3) Polish the video. Add a title card @ beginning and end
4) Write text descriptions of the supplementary videos
5) Review verb tenses, units
6) **Print the document and give to your PI**
7) Split paper into main text and SI
8) Render videos to the appropriate filesize
9) Make sure you fit in the limits (page length, number of references, number of figures)
10) Review the editor's email. Sometimes they have other requirements, like linking the corresponding author's ORCID or adhering to specific journal formatting rules.
11) Submit
12) Move everything to our org's group drive

