# A Letter to Research Students[1]

*Duane A. Bailey*
Department of Computer Science
Williams College
Williamstown, Massachusetts 01267
`bailey@cs.williams.edu`

Research can be made more enjoyable and productive through a little organization. For many computer scientists, research techniques are a product of experience rather than, unfortunately, formal training. Several organizational suggestions are outlined here, most of which are useful to the undergraduate researcher. Items in italics are aimed at graduate students, professional researchers, and faculty. Enjoy.

**Read My Tips.**
Successful computer science research involves the coordination of many efforts. To make your time as productive and interesting as possible I have collected a few "organizational hints" that you might find useful. Take time to read these tips. Some are painfully obvious (others are just painful!), but each has the potential to improve your life as a researcher.

**Reading is Fundamental.**
Finding and reading related work is the foundation of good research. If you are new to your field, you may only be familiar with material presented in textbooks. Two important initial resources for finding pertinent references are the *ACM Guide to Computing Literature*[3] and *Computing Reviews*[2]. When you initially investigate your topic you will start with a short list of foundation readings. You, as a researcher, are responsible for developing a bibliography of related works, and for finding future readings. You will find useful references others have missed.

Keep reading. Although background reading is the initial task in any research endeavor, important readings often surface later, especially if the focus of the research changes. Make sure you identify the sources that are important to your field and keep up with journals as they arrive. It only takes a couple of hours each month to peruse the latest papers. (Proceedings of conferences play an important role in quickly delivering research results, although they are not reviewed as rigorously as journal articles.) Since computer science research ages quickly, keeping abreast of recent technical readings is vital.

[1] The latest version of this letter is available through anonymous `ftp` from `cs.williams.edu` as `research.tar.Z`. The author welcomes comments.

Read articles with care. A cursory review of a work can identify material that is topical. When a paper makes a contribution, it is important to consider the paper in greater detail. As you read, consider the following:

> What is the contribution of this research?
> How does this contribution relate to work previously encountered?
> What are the important references cited in this paper by the author?

These may seem obvious and unsophisticated questions to be asking, but that is no reason to avoid asking them.

Write a summary of each paper you believe is pertinent to your research. Summaries are useful in quickly describing work to others, and they provide a "bootstrapping" mechanism should you have to reread the material later. In a research career, summarizing work (yours as well as that of others) is a common task; doing it concisely is an art.

After reading a well written paper, consider the presentation. It may be nicely organized, or it may underscore concepts with thoughtful examples.

> What makes this paper easy to read?
> What level of detail is provided?
> What examples are used to demonstrate important concepts?
> What questions are left unanswered?
> Can the results be generalized?

Since we learn by example, it is important to flag articles that are particularly well written.

Unfortunately, technical literature is often obscurely written. On occasion, it may be helpful to read an article with your advisor, or a interested colleague. The pace may be slow, but in the end the potential of a well written article will have been realized, and your comprehension of the article will be improved.

*In larger departments an informal seminar or "journal club" provides an excuse for researchers to meet weekly to discuss papers. Focus on a topic ("The Semester of Garbage Collection"), but allow for digression when a common interest develops. These informal weekly meetings reserve time in crowded calendars for research that might otherwise be pushed aside.*

**Writing is Fundamental.**
Good writing is the only lasting medium of the scientific process. Long after talks have

faded and programs have been purged, written work serves to carry important concepts forward. Thus, you should begin writing up your results as early as possible. While experimentation is important, the efforts are wasted unless they are carefully reported.

Write well. When in doubt, simple writing succeeds[11]. If you are a new or infrequent writer, you will be concerned with sharpening your writing skills. If you are lucky, your readings will include examples of successful writing in your area. Generally, effective articles convey the important points in a single reading. They discuss interesting examples. When technical details are presented directly, either as mathematics or code,[2] they are not a substitute for English. Writing in a style that conveys information correctly and concisely is difficult. When you are unsure of your stylistic success, reread a paper you enjoyed with an eye to integrating stylistic qualities into your writing.

Keep a journal. In this journal write references, pose questions, describe problems and their solutions. Keep track of experiments and their results. As lab scientists know, the journal is a simple tool for organizing your research and it is a valuable record of progress you make.[3]

Keep a summary of results encountered in related readings. If you expect to present your research formally (always be optimistic!), positioning your work with respect to others is a service to your readers. Reconsidering related research also serves to remind you of important contributions, and to align your goals with those of others in your area. Since this document needs to be edited frequently, it should be stored electronically. (BibTeX[10] is a good system for organizing bibliographic information and it is a portable mechanism for exchanging references with others.)

*Keep a sheaf of small projects on paper. These are one or two pages of formally presented ideas on a single subject. When ideas are not allowed to get out and wander about they become malformed and misaligned. These writeups keep the pencil sharp and serve to organize spontaneous ideas. Having these available at a moment's notice pays off when students are looking for research projects.*

Reserve a good portion of time for writing. An hour spent writing is an hour spent *considering a problem* instead of, say, *grappling with a computer*. You *must* spend time away from other distractions to document work and focus your efforts. Like regular exercise, it needs to be done, and you'll come to enjoy it.

[2]The etymology of this word is telling.

[3]Knuth's log of errors in TeX gives insight to the record keeping of a productive computer scientist.[9]

**Working With Others.**
For many, success comes from work with others. It is important to share ideas and to let them develop in a group atmosphere. (In small departments this can be problematic — especially if one is in an isolated field — but cooperative efforts are still important.) Whenever possible, use an advisor or research peer as a sounding board for your ideas. You may think that your idea is not worthwhile, but give it a try anyway. Get the conversation going! Most researchers are happy to participate in productive discussions.

Keep to a regular schedule of meetings. If it appears that there is "nothing to discuss," *that* is an important topic. A skipped or cancelled meeting sets a precedent that is hard to reverse. If you have become dislodged from your group, *you* should re-establish contact. It is unlikely that they will seek you out, and common interest decays with time.

Meetings provide points of synchronization where concentration is refocused on the issue. Important points are made, so take the time to write these ideas down (bring your journal!) and distribute copies. These informal "minutes" help to maintain context between discussions and document decisions that have been made. Rehashing old material is usually a waste of time.

Carefully consider criticism. Since peer review is an integral component of the scientific process, there will be times when your research is critiqued. Do not take critical commentary personally, instead use it as a guideline to making your arguments more effective. Likewise, when criticizing the work of others, it is important to make your arguments *constructive*; the nonconstructive comment is rightfully ignored.

**Talk Isn't Cheap.**
In any serious research you will find yourself giving a talk. Good talks require considerable preparation, so start soon. Slides should be legible and void of inessentials, including "code." They should contain illustrative examples, but only at the level of detail that is appropriate for your audience. Slides should work together to document your contributions.

Rehearse and time your talks. Talks that are unprepared are obvious, and usually cause an otherwise open-minded audience to mount an attack. A talk that is well structured and rehearsed better prepares you for that Unexpected Question.

**The Project.**
Research in computer science often leads to a "project" involving programming. It is important to remember that *programming is not computer science research*. Instead, for most computer scientists, programming is merely a mechanism for performing an

experiment. As with any experiment, it should be carefully planned ahead of time:

⋆ Establish goals. Know where you are headed, and approach the solution without distraction. Develop a list of milestones which demonstrate progress, and strive to accomplish them. If you cannot formulate concise goals, you should stop and reconsider the motivations for the project.

⋆ Think simple. Design your projects so that they may be completed within a reasonable period of time. An experienced programmer generates little more than a hundred lines of reliable code per month. A project that demands thousands of lines of code, therefore, will take more than a couple of semesters to implement correctly. Time spent pruning the experiment to a manageable size is time well spent.[4] Large projects do not necessarily yield large results.

⋆ Build prototypes. Most projects benefit from the construction of a prototype. A well considered prototype validates assumptions, tests the value of abstractions, and motivates reconsideration of weak ideas. While there is little *research value* associated with polishing off a "product," many research questions can be answered satisfactorily through mock-ups or partial implementations.

⋆ Use tools. A programmer's performance is dramatically improved through the use of a few simple tools. Today's programmers should be aware of, for example, anim[6],[5] awk[1, 4, 5],[6] bison[7], HyperCard[8], and Mathematica[13]. In windowing environments, interface builders can eliminate many of the distractions of interface design. There are, of course, many other important and useful tools, but the main point is clear: the correct choice of tools can reduce the total work in a project. Find them, learn from them, and use them.

⋆ *Collaborate. When resources can be coordinated, groups are often more productive than individual efforts in isolation. If you share research interests with others, contact them and collaborate. Undoubtedly they will have solved problems you are currently considering, and their solutions will influence how you achieve common goals.*

*One side effect of collaboration is increased discipline, discipline that is necessary to reduce the amount of energy expended synchronizing efforts. For programmers, a variety of resources are*

[4] " 'Make it simple as possible, but not simpler.' — Einstein " — Wirth[12].

[5] Trademarks are properties of their respective holders.

[6] It is interesting to note that our department presents a copy of *Programming Pearls*[4] to a student that demonstrates excellence in presenting research.

*currently available for managing distributed projects. For example, a suitable set of coding standards have been developed by the GNU project,*[7] *and common code style can be suitably enforced by programs (e.g.* `indent`*). When access to the project is shared, a version control system (e.g. SCCS or RCS*[8]*) should be considered.*

⋆ Document results. Finished projects should be documented. At a minimum, a technical overview of the experiment allows others to see what motivates your research. The document should describe the problem, your assumptions, your approach, and an honest evaluation of your results. When documenting software, include illustrative examples, tutorials, and any experience gained from its use. Well written documentation greatly increases the impact of a project.

**Summary.**
Your participation in research, ideally, is part of a pipeline. Your work builds on that of others and produces something of utility. As a participant in a fast-moving field, you are *responsible* for keeping informed, developing and performing suitable experiments to test your hypotheses, and presenting your results in a manner that makes them accessible to the research community. The suggestions presented here can help you keep up with these responsibilities.

---

**References.**
Library of Congress call numbers follow each reference.


[1] Alfred V. Aho, Brian W. Kernighan, and Peter J. Weinberger. *The AWK Programming Language.* Addison-Wesley, Reading, Massachusetts, 1988.
`QA 76.73 A95 A35`.

[2] Association for Computing Machinery. *ACM Computing Reviews.* Association for Computing Machinery, New York, 1960–.
`QA 76 C5854`.

[3] Association for Computing Machinery. *ACM Guide to Computing Literature.* Association for Computing Literature, New York, 1977–.
`QA 76 B486`.


[7] See `prep.ai.mit.edu:pub/gnu/standards.info`, available through anonymous `ftp`.
[8] See files in `cs.purdue.edu:pub/RCS`, available through anonymous `ftp`.

[4] Jon L. Bentley. *Programming Pearls.* Addison-Wesley, Reading, Massachusetts, 1986.
`QA 76.6 B453 1986.`

[5] Jon L. Bentley. *More Programming Pearls: Confessions of a Coder.* Addison-Wesley, Reading, Massachusetts, 1988.
`QA 76.6 B452 1988.`

[6] Jon L. Bentley and Brian W. Kernighan. A system for algorithm animation tutorial and user's manual. Technical Report 132, AT&T Bell Laboratories, Murray Hill, New Jersey, 1987.

[7] Charles Donnelly and Richard M. Stallman. *Bison: The YACC-compatible Parser Generator.* The Free Software Foundation, Cambridge, Massachusetts, 1991.
Available via anonymous `ftp` from `prep.ai.mit.edu`.

[8] Danny Goodman. *The Complete HyperCard 2.0 Handbook.* Bantam Books, Toronto, third edition, 1990.
`QA 76.8 M3 G645 1988.`

[9] Donald E. Knuth. *Literate Programming.* Center for the Study of Language and Information, Stanford University, 1992.
`QA 76.6 K644 1991.`

[10] Leslie Lamport. LaTeX*: A Document Preparation System.* Addison-Wesley, Reading, Massachusetts, 1986.
`Z 253.4 L38 L35.`

[11] William Strunk and E. B. White. *Elements of Style.* Macmillan, New York, third edition, 1979.
`PE 1408 S772 1979.`

[12] N. Wirth. The programming language Oberon. Technical report, ETH, Zurich, January 1990.
Available via anonymous `ftp` from `neptune.ethz.ch:Oberon/Docu/OberonReport.ps`.

[13] Stephen Wolfram. *Mathematica: A System for Doing Mathematics by Computer.* Addison-Wesley, Redwood City, second edition, 1988.
`QA 76.95 W65 1991.`