Blunt Instrumentalism: On Tools and Methods

Dennis Tenen

I am on the side of the makers. I believe that the humanities can be a place not just to think about things, but to do things. Doing, when done right, can expand the scope of our critical activity, prepare our students for work in the world, and finally--and this despite the protestations of some--enact meaningful change in our communities.[[1]](#endnote-2) I write, then, being inspired by research at institutions such as the Critical Making Lab at University of Toronto, Concept Lab at UC Irvine, and metaLab at Harvard, along with many similar research centers that routinely engage with material culture as a matter of scholarly practice. In my courses as well, students create models, curate exhibitions, file patents, convene conferences, write grant applications, send letters to the Senate, draw, build, and code. However, the academy also presents some unique challenges to critical making of that sort, particularly when it comes to sustainable tool development. As tool makers, we should heed the lessons of the numerous forgotten projects that did not find an audience or that failed to make an impact. For every line of code actively running Pandoc, NLTK, or Zotero, there are hundreds that lie fallow in disuse. Yet even in failure, this codebase can teach us something about the relationship between tools and methods.[[2]](#endnote-3) In reflecting on my own failed projects, I have come to believe that with some notable exceptions, the university is an unfit place to develop “big” software. We are much better poised to remain agile, to tinker, and to experiment.

The digital humanities (DH) can be understood as a part of a wider “computational turn” affecting all major disciplines: see computational biology, computational linguistics, computational social science, computational chemistry, and so on. Computation in the humanities supplements the traditional research toolkit of a historian, a literary scholar, and a philosopher.[[3]](#endnote-4) In this essay however, I would like to bring into question a specific mode of tool making, practiced within the digital humanities and without, of the kind that confuses tools with methods. The tools I have in mind prevent, or--more perniciously--tacitly discourage critical engagement with methodology.

To see the problem with tools more clearly, imagine a group of astronomers using a telescope that reveals to them wondrous star constellations. Yet, our hypothetical scientists cannot tell if these stars actually exist, or whether they are merely an artifact of a faulty telescope. This has always been the tool-wielder’s dilemma. Contemporary research instrumentation in our field, from natural language processing to network analysis, involves complex mechanisms. Their inner workings often lie beyond the full comprehension of the casual user. To use such tools well, we must, in some real sense, understand them better than the tool-makers. At the very least, we should know them well enough to comprehend their biases and limitations.

The best kind of tools are, therefore, the ones that we make ourselves. After spending days wrangling a particularly messy corpus, I might write a script that automates data cleanup. My code may strip out extraneous HTML markup, for example. I could then release the script as a software library to help others who face the same task. With time, I might add a graphical user interface (GUI) or even build a web site that makes using my scripts that much easier. Such small acts accelerate the research capabilities of the field as a whole. I would do nothing to discourage analogously altruistic sharing. But let us be sure that in using tools we also do not forget to master them from the inside out. What if my code implicitly mangles important metadata; or worse, what if it alters primary sources in an unexpected and tendentious ways? Let the tool makers make such biases explicit to the public.

**Methods within**

Some tools encourage intellectual laziness by obscuring methodology. More often, it is not the tool but rather a mode of lazy thinking that is at fault. For example: the *nltk.cluster* module bundled in Python’s Natural Language Toolkit (NLTK) framework[[4]](#endnote-5) contains an implementation of something called “k-means clustering,” an unsupervised method of finding groups of similar documents within a large collection.[[5]](#endnote-6) The "unsupervised" part means that we are looking for hidden structure without making any assumptions about the documents at the outset.[[6]](#endnote-7) The documents may be grouped by the preponderance of personal pronouns or perhaps by sentence length. We do not know what elements the algorithm will identify, only that it will make piles "typical" of our corpus. The tricky part comes in estimating the number of expected document clusters (that is the *k* variable). In a corpus of nineteenth century novels, for example, one may expect a dozen or so clusters, which could perhaps correspond to novelistic genres. When clustering a large database of diplomatic communiques, one would reasonably expect more fine-grained "piles" of documents, which could have something to do with regional differences or with major political events. In either case, the algorithm will blindly return some groupings of distinctly-related documents.

But whatever the results of clustering, they are difficult to interpret in terms of meaningful literary-historical categories like "genre" or "period." Some of our piles will correspond to genres and periods, while others will seem meaningless. The algorithm produces non-hierarchical results--that is, the output is not ordered according to value or significance. As the algorithm is also non-deterministic, meaning that it will perform differently each time it is run, the groupings will also vary with each iteration. To complicate matters, NLTK implements other clustering algorithms, like expectation–maximization (E-M) and group average agglomerative clustering (GAAC). These methods will likely chance upon yet other hidden relations between documents and other ways of organizing the material into piles. The algorithm will always return *a* result, according to *some* set of formal commonalities. But what these results mean and why is open to interpretation. To make the clusters meaningful requires a deep understanding of the underlying logic.

NLTK facilitates such discovery by distributing detailed documentation along with the code. The documentation does more than just describe the code: it reveals implicit assumptions, citing external sources throughout. In experimenting with NLTK, I was able to get some output from the clustering methods in a matter of days. It took me months to understand what they could mean and how they could be applicable to my research. Just applying the tool or even "learning to code" alone was therefore insufficient for making sense of the results. What could help me then and what is only now beginning to surface in DH literature is a critical conversation on the methodology.

Unlike some other tools of its kind, NLTK is particularly good at revealing its methods. Its codebase is open to inspection; it is easy to read and well-formatted; and it contains much commentary along with links to related research. The NLTK project began in 2001, at the University of Pennsylvania, in a collaboration between a linguist and his student.[[7]](#endnote-8) Research based on the module started appearing in print several years later, around 2004. NLTK reached version 1.0 eight years after its inception, in 2009. In the intervening time, immense care must have went into the critical apparatus that comes with the tool. And I suspect that at this late stage of the project, more hours have gone into the writing of its documentation than into the crafting of its code. As of 2015, the NLTK GitHub page lists no fewer than 130 contributors. Reflecting on the history of NLTK gives us a glimpse into the realities of responsible academic making. Not every project will need to go through such a long development cycle or include such detailed documentation. But even my own small collection of data cleaning scripts would need substantial work to reach the level of polish required for empowered use of the kind NLTK enables.

Note also that NLTK itself is only a "wrapper" around a set of statistical methods for the analysis of natural language. That layer of encapsulation already poses a number of problems for the researcher. Using NLTK responsibly demands a degree of statistical literacy along with programming experience. The cited methodology often contains a mixture of code and mathematical formula. Yet higher-level encapsulations of NLTK, like a web-based topic modeler, for example, would further remove the user from that implicit logic. Each level of abstraction in the movement from statistical methods, to Python code, to graphical user interfaces introduces its own set of assumptions, compromises, and complications. Any "ease of use" gained in simplifying the instrument, is gained at the expense of added and hidden complexity. Hidden complexity puts the wielder of the tool in danger of resembling a hapless astronomer. To avoid receiving wondrous pictures from broken telescopes, in the way of actual astronomers, we would have to learn to disassemble the device and to gain access to its innermost meaning-making apparatus. Any attempt to further repackage or to simplify the tool can only add another layer of obfuscation.

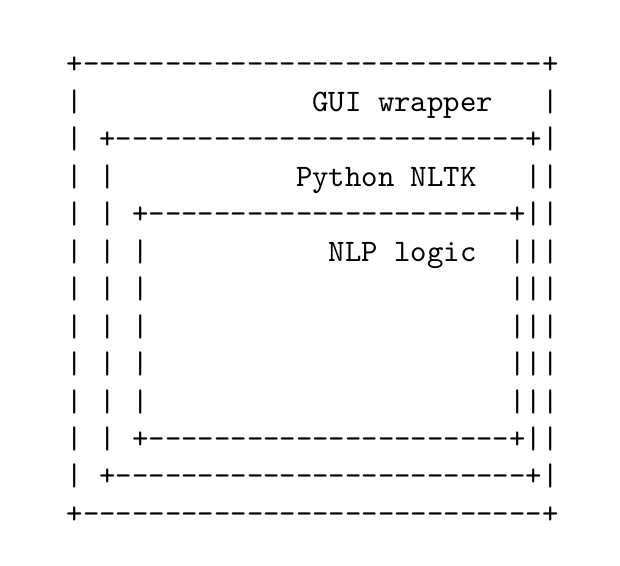


Figure 1: Layers of encapsulation.

It follows, then, that without a critical discussion about implicit methods, out-of-the-box tool use is best treated with a measure of suspicion. The makers of out-of-the-box tools should similarly weigh the altruistic desire to make research easier against the potential side effects that come with increased complexity. The tool can only serve as a vehicle for methodology. The logic itself is the important part. Researchers can debate about methods and improve them with time. Tools proliferate and decline in quality relative to the researcher’s experience. A scholar will likely reach the instrumental limits of any NLTK derivative well before publication. And even NLTK itself acts as a shortcut to the conversation on the limits of natural language processing in the humanities. Where the implementation can be contained in a footnote, the method requires its own section. And if tomorrow’s researchers move from Python to Haskell, the effort of learning about k-means clustering will transfer with the language. The tool may become obsolete, where the method remains.

**Unplanned obsolescence**

In addition to such methodological concerns, tool making also involves pragmatic considerations about sustainability. Software is cheap and fun to build relative to the expense and drudgery of maintenance. “90% of coding is debugging. The other 10% is writing bugs.”[[8]](#endnote-9) This aphorism comes naturally to program managers and software engineers who have gone through the full software product development cycle. In the excitement of building new tools, it is however easy to underestimate the challenges of long-term application maintenance. Academic attention spans are naturally cyclical: articles are published, interest wanes, funding dries up, students graduate. Scholars start anew each year and each semester. By contrast, software is linear. It requires the continuity of care and much more of it as a codebase matures. Standards change, dependencies break, platforms decay, users have questions. The case for the humanities as a laboratory for innovation is strong, but I doubt that many are prepared to make “critical customer support” a part of their research agenda. A few select institutions have experience in dealing with the contingencies of long-term software maintenance successfully. Others should think twice before investing resources into tool development. Not every method needs to be packaged into a tool. Some projects would be better off contributing to existing efforts, or using their resources to encourage methodological literacy.

In fact, if you build it, they might not come at all. Startups know that beyond the initial excitement of a product launch, the challenge of any new application lies in the acquisition and the retention of users, no matter how “disruptive” or “innovative” the technology. A few years ago, I spent some time working with a talented French developer on the next generation of a crowd-sourced translation service. Despite his skills and dedication to the project, the tool did not gain significant traction among translators or language students. I learned then that no amount of innovative engineering or beautiful web design could guarantee participation. Neither of us had the time or the resources to *advocate* for the service. Advocacy would require arranging for training, outreach, and support: services we could not provide in addition to our professional obligations. It was however tempting to think that social or institutional change could ride on the coat tails of software alone. If we build it right, we thought, we could transform the practice of translation in the classroom. Yet we failed to consider the difficulty of implementing that vision into practice. We built the tool but not the community around it. The classroom environment resisted change, and for a good reason. Upon reflection, we saw that language teaching was grounded in proven, if sometimes imperfect, practices. Our platform development should have considered the strengths of that tradition and not just its weaknesses. Before rushing to innovate, we could have started with smaller classroom experiments to test our intuitions. We should have arranged for interviews, focus groups, and pilot studies.

Consider the following in the case of our hypothetical "wrapper" around NLTK--the one that would simplify the use of natural language processing. Every new Macintosh laptop comes prepackaged with powerful command-line tools for text manipulation, such as *wc*, *sort*, and *uniq*. Together they can already be used to count and sort words in a document or to generate a term-frequency distribution useful for formal text analysis. These small utilities are free, simple to learn, versatile, and require no additional installation. They come with their own textbook, accessible from the terminal.[[9]](#endnote-10) Yet most of my students, even at the intermediate level, remain unaware of them. Many were not exposed to the basics of file paths, networking, or operating systems. How can one better facilitate the practice of computational text analysis without closing the digital literacy gap that separates mere users from empowered tinkerers and tool makers?

A proposal to implement yet another tool duplicating the functionality of ubiquitous native utilities gives me pause. That is not to say that existing word frequency tools cannot be refined in some way. But, any new project that hopes to do that would have to at least match the power of the existing array of tools*,* and then improve upon them in some capacity. And even then, our hypothetical project would face the same barriers to literacy and adoption as the original toolkit. These would have to be addressed before writing a single line of new code.

Furthermore, whatever adoption the new alternative might achieve risks fracturing the existing user base, already limited to a small number of practitioners. By analogy, a new publishing platform that hopes to uniformly “disrupt” academic publishing is far more likely to enter an already fragmented market, rife with alternatives. The fragmentation prevents any one them from gaining critical mass. Instrumental efficacy alone therefore cannot address the lack of adoption. Legacy tools like Microsoft Word or clunky journal management systems (used behind the scenes for peer review), for example, do not account for the range of "planned obsolescence" problems in academic publishing.[[10]](#endnote-11) The tool comprises but a small part of a much larger publishing ecosystem. It can act as a wedge that initiates change, but not without a larger communal effort to address the way we read, write, and *do* research. No matter how promising a tool's potential, its adoption will be stymied by insufficient training and lack of support. Rather than fracturing the community, we would often do better to join forces: to congeal our efforts around common standards and best practices. Funding however favors statements of bold innovation, where it would be prudent to invest into organic growth.

The effort to shift the habitus of a community, as Pierre Bourdieu would describe it, involves a delicate balance between disruption and continuance. Much can be learned from the success of the open source and free culture movements in this regard.[[11]](#endnote-12) Take, for example, the story of *Wikipedia* and *MediaWiki.* *MediaWiki,* the software platform powering *Wikipedia*, was neither the first nor the most technically sophisticated wiki software package. But in the hands of Wikipedians, as that community is known, *MediaWiki* became a tool capable of transforming the contemporary information landscape. Despite some of its problems, *Wikipedia* struck the right balance between traditional forms of knowledge-making (i.e. the encyclopedia) and innovative editorial structures (e.g. commons-based peer production).[[12]](#endnote-13),[[13]](#endnote-14) *Wikipedia* the community inspires me more than *MediaWiki* the tool. In the *Wikipedia* world, the platform is secondary to community development.

The care of academic research communities, of the kind that encourages empowered tool use, happens in departments and through professional organizations. Programs like the Digital Humanities Summer Institute answer the need for training necessary to do research in our rapidly-developing field. However, more resources are needed to initiate methodological and not just instrumental innovation. Few humanities-based alternatives exist to parallel associations like the *Society for Political Methodology and the International Association of Legal Methodology;* journals like *Sociological Methods & Research,* *Journal of Mixed Methods Research*, *International Journal of Qualitative Methods;* prizes and funding opportunities like the *Political Methodology Career Achievement and Emerging Scholars Awards,* or the *Program for Promoting Methodological Innovation in Humanities and Social Sciences* administered by the Japan Society for the Promotion of Science. To sharpen our tools we must similarly prioritize methodological development. Only then can we build platforms that answer to the values of humanistic critical inquiry.

A shared concern with data and computation has brought a number of disciplines closer together. Biologists, linguists, economists, and sociologists increasingly integrate their methodologies, as evidenced by a vigorous cross-disciplinary publishing record. DH is primed to join that conversation, but only if its methods develop without abridgment. Tools are great when they save time, but not when they shield us from the complexity of thought. Working as a digital humanist or a new media scholar means taking on extra responsibilities: to do well by history when writing history, to make things that last when making things, and to do good science when doing science.

1. See Stanley Fish, *Save the World on Your Own Time*, Second Edition (Oxford University Press, USA, 2008) [↑](#endnote-ref-2)
2. William Pannapacker has written eloquently on the topic in the Chronicle of Higher Education. See William Pannapacker, “[Pannapacker From MLA: The Success of 'Failure’](http://chronicle.com/blogs/brainstorm/pannapacker-from-mla-failure-is-the-new-normal/30864),” The Chronicle of Higher Education, *From the Archives: Brainstorm*, (January 2011). [↑](#endnote-ref-3)
3. I do not mean to imply that DH can be *reduced* to computation. See Stephen Ramsay and Geoffrey Rockwell, “Developing Things: Notes Toward an Epistemology of Building in the Digital Humanities,” in *Debates in the Digital Humanities* (Minneapolis: Univ Of Minnesota Press, 2012). Also: D Elliott, R MacDougall, and W.J Turkel, “New Old Things: Fabrication, Physical Computing, and Experiment in Historical Practice,” *Canadian Journal of Communication* 37, no. 1 (2012): 121–28. [↑](#endnote-ref-4)
4. Steven Bird, Ewan Klein, and Edward Loper, *Natural Language Processing with Python* (Cambridge [Mass.]: O’Reilly, 2009) [↑](#endnote-ref-5)
5. Astronomers also use k-means clustering to identify star constellations. See also J.MacQueen, “Some Methods for Classification and Analysis of Multivariate Observations,” in *Proc. Fifth Berkeley Sympos. Math. Statist. and Probability (Berkeley, Calif., 1965/66)* (Berkeley, Calif.: Univ. California Press, 1967), Vol. I: Statistics, pp.281–97. [↑](#endnote-ref-6)
6. Shi Na, Liu Xumin, and Guan Yohng, “Research on K-Means Clustering Algorithm: An Improved K-Means Clustering Algorithm,” in *2010 Third International Symposium on Intelligent Information Technology and Security Informatics* (IITSI), 2010, 63–67. [↑](#endnote-ref-7)
7. Loper, Edward, and Steven Bird. 2002. “NLTK: The Natural Language Toolkit.” In *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics* - Volume 1, 63–70. ETMTNLP ’02. Stroudsburg, PA, USA: Association for Computational Linguistics. [↑](#endnote-ref-8)
8. The quote is commonly attributed to Bram Cohen, the creator of BitTorrent, posted on tweeter.com in 2011. There are however numerous earlier instances of the exact quote, itself a variation of Sturgeon’s Law coined by Theodore Sturgeon (the American science fiction writer) in a 1957 article for *Venture* magazine and cited as such in the Oxford English Dictionary. [↑](#endnote-ref-9)
9. If you are behind one of these machines now, search for your terminal application using Spotlight and type man wc in the prompt (q to exit). For mere examples see: <https://github.com/xpmethod/dhnotes/blob/master/command-line/109-text.md> [↑](#endnote-ref-10)
10. I have in mind the sort of problems Kathleen Fitzpatrick outlines in Kathleen Fitzpatrick, *Planned Obsolescence Publishing, Technology, and the Future of the Academy* (New York: New York University Press, 2011), <http://public.eblib.com/choice/publicfullrecord.aspx?p=865470>. [↑](#endnote-ref-11)
11. Steve Weber, *The Success of Open Source* (Cambridge, MA: Harvard University Press, 2004). [↑](#endnote-ref-12)
12. See for example Collier, Benjamin, and Julia Bear. 2012. “Conflict, Criticism, or Confidence: An Empirical Examination of the Gender Gap in Wikipedia Contributions.” In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*, 383–92. CSCW ’12. New York, NY, USA: ACM and Callahan, Ewa S., and Susan C. Herring. 2011. “Cultural Bias in Wikipedia Content on Famous Persons.” *Journal of the American Society for Information Science and Technology* 62 (10): 1899–1915. [↑](#endnote-ref-13)
13. A point made by Benjamin Mako Hill in his *Almost Wikipedia: What eight early online collaborative encyclopedia projects reveal about the mechanisms of collective action*, summarized in a recent talk at the Berkman Center for Internet and Society, abstract and transcripts available at <http://cyber.law.harvard.edu/events/luncheon/2011/10/makohill>. Another good summary by Garber, Megan. “[The Contribution Conundrum: Why Did Wikipedia Succeed While Other Encyclopedias Failed?](http://www.niemanlab.org/2011/10/the-contribution-conundrum-why-did-wikipedia-succeed-while-other-encyclopedias-failed/)” Nieman Journalism Lab. Accessed December 22, 2012. [↑](#endnote-ref-14)