# Trusting Others to 'Do the Math'

Rebecca Sutton Koeser

*Emory University Libraries & IT Services, Emory University, Atlanta, Georgia, USA*

Researchers effectively trust the work of others anytime they use software tools or custom software. In this article I explore this notion of trusting others, using Digital Humanities as a focus, and drawing on my own experience. Software is inherently flawed and limited, so when its use in scholarship demands better practices and terminology, to review research software and describe development processes. It is also important to make research software engineers and their work more visible, both for the purposes of review and credit.

KEYWORDS software development, research software engineers, digital publishing, digital humanities

In November of 2012, the Maryland Institute for Technology in the Humanities (MITH) hosted a one-day workshop 'Topic Modelling for Humanities Research' which brought together humanities scholars and natural language processing researchers to present on their work and discuss the various applications of topic modelling to humanities research. At some point, after explaining some of the specifics of the probabilistic approach used in the Latent Dirichlet Allocation (LDA) algorithm, David Mimno attempted to reassure those who were struggling with the details by saying 'trust me to do the math' (Mimno 2012).[1] Later on in the workshop, in a discussion of the some of the particulars of the implementation including Gibbs sampling and hidden variables, David Blei referenced this comment, opining that the humanists did not need to care how those particular values were arrived at, but just 'trust that it came from David Mimno'.[2]

When we use software, whether we recognize it or not, we are always trusting others to 'do the math'. This is just as true for the software engineer developing a brand new application as it is for the end-user checking email or surfing the web, and you can count nearly as many layers down as you want to — from the programming language and its libraries, to the operating system, to the physical circuits in the hardware, and all the various protocols used in between and across devices. We usually do not think about all the components we are trusting to work properly — especially when we are acting simply as a technology consumer — and these various pieces often go unnoticed and unremarked until something

goes wrong, whether a hardware failure, a bug in a dependency, or even an unexpected limitation of a particular piece of software. When scholars use software to complete or share their research at any point in the process, how reasonable is it for them to ignore the complexity and the various points at which they must trust others to 'do the math'? How much do scholars need to understand the technological tools they use, and what role do the creators of that software play in the resulting scholarship? According to a survey of UK researchers by the Software Sustainability Institute, 92% of academics use research software (Hettrick 2014). That means software is incredibly pervasive in scholarly work; but how thoughtful are we about it?

Software is pervasive in our lives and our work, but we often take it for granted. Consider a simple case of the MD5 message-digest algorithm (Rivest 1992). MD5 was originally used for cryptographic purposes, and is now still often used to generate checksums for verifying data integrity. We now know that the algorithm is flawed for security purposes and that it is possible to find or even maliciously engineer different files with the same checksum. Despite this, MD5 remains in use for verifying that a downloaded file has transferred correctly, or to check for 'bit rot' — small random changes to data — in preservation systems. Even the numeral 5 in the name MD5 is actually an indicator that this algorithm succeeded previous versions, just as it has now been superseded by others like the 'Secure Hash Algorithms' such as SHA1 or SHA256. If a scholar wishes to use MD5 for verifying download integrity, they do not necessarily have to know the exact algorithm. However, it would probably be a good idea to have some understanding of how it works, what it should be used for, and where it might fail. In the case of MD5, the algorithm is publicly available, and there are implementations of it in any number of programming languages, which make it easier to verify a result by comparing the output of two different systems, and if for some reason I needed to, I could actually implement the algorithm for myself based on the documentation. However, in this case, the algorithm has been broken for cryptographic purposes, and even if I implemented my own version I might still easily miss those flaws. I have to rely on others to do the math for me, whether it is identifying flaws or coming up with new, alternative hashing algorithms.

In technical discussions, people often use the idiom of 'looking under the hood' when they discuss the implementation details. So, for an alternate way of thinking about our knowledge and use of technical systems, consider the automobile. Most drivers and passengers probably do not have a very deep understanding of the mechanisms that propel a car forward, allow the steering to operate, or slow the car down when the brake is pressed — from the relatively simple mechanics of the piston engine, crankshafts, and distributors to the more complicated computer-assisted systems that monitor and adjust all the various components. Of course, this level of knowledge is not necessary to drive successfully — a person can operate a vehicle without understanding how it works exactly, as long as they are familiar with the 'user interface' of power, steering wheel, and gas and brake pedals. However, we might expect the person with a deeper understanding of how things are connected and what actually happens when the various pedals are

pressed to be a better driver, better able to react properly when an unusual situation comes up.

When scholarship and software become intertwined, as they are now in almost every field of research, it is important to make sure we have some understanding of the tools we are using, what their flaws may be, and where they are taking us. Are scholars who use software simply riding along in a vehicle to arrive at some destination — somewhere that perhaps they could have reached just as well without the vehicle, if maybe a bit slower or by a different route? Or is the software more integral to the process or the results than that? Researchers do not yet have consistent ways to articulate how the software is used, or how significant it is to the scholarship that results from it. But some scholarship *is* more dependent on software, and in those cases of deeper involvement, it is important to be careful and thoughtful. How should researchers working with software developers or tools establish the trust necessary to let them 'do the math'? Do they need to be more willing to try to 'look under the hood'? And what are the stakes for scholarship if there is something wrong in the software?

My focus here is on the humanities, and on the field of Digital Humanities in particular. This is in part due to my own expertise and familiarity, since I have worked for several years as a software engineer in an academic library where I have contributed to multiple, diverse digital library and digital humanities projects; but also because the treatment of software in humanities research seems to be lagging behind other fields. The place of software in research in the sciences, engineering, and medical research, and the like seems to be more entrenched, particularly with groups like the Software Sustainability Institute and Software Carpentry advocating and providing better training and better practices. But surely, we all could use a reminder of the flaws and assumptions that are inevitably present in the software we use as well as considering how we should document and account for the roles software and its creators play in scholarly work.

## Flaws in the software

We know that software is flawed; this is frustratingly evident when a tool crashes and loses unsaved data, or when we try to do something with a piece of software that it was not designed to support. There are limitations, bugs, and security holes. The recent high profile and widespread 'Heartbleed Bug' and 'goto fail'; security problems demonstrate that these flaws can be present for a long time and go unnoticed. The Heartbleed vulnerability, for example, was exposed for over two years (The Heartbleed Bug 2015). Individual software projects may go through multiple versions, but are rarely ever perfect or complete. One algorithm is replaced by another as people refine ideas and approaches, or as new hardware with more computing power becomes available. Unicode replaces ASCII. MD5 succeeds MD4, and succeeded by the Secure Hash Algorithms.

What are the stakes for scholarship, when it makes use of or is built on software which may be flawed or unfinished, or where one algorithm may soon be superseded by another? When Digital Humanists use digital tools for 'distant

reading' on large corpuses of text, does it matter if their vision is a little out of focus or distorted, thanks to their software? In the case of topic modelling, LDA is the current algorithm of choice, popularized in part by the implementation in the Machine Learning for Language Toolkit (MALLET) along with successful projects such as Robert Nelson's 'Mining the *Dispatch*' (Nelson 2011) and Matthew Jockers 'macroanalytic' approach to nineteenth-century literature (Jockers and Mimno 2013, and elsewhere). It was also prominently featured in a special issue of the *Journal of Digital Humanities* on topic modelling intended to 'catch and present the most salient elements of the … conversation' (Meeks 2013). But the LDA algorithm was preceded by latent semantic analysis, and as with the crypto-graphic hash algorithms MD5 and SHA-1, it seems likely it will eventually be replaced by something else. What does that mean for the scholarly work based on LDA? When Mimno suggests that we trust him to 'do the math', he means literal mathematics in the form of complicated probabilistic calculations, as well as efficiency improvements where those calculations can be skipped or approxi-mated (Yao *et al*. 2009). Mimno is the current supervisor of the MALLET toolkit and the author of the topic modelling component within it; has worked with David Blei, one of the creators of LDA; and continues to refine and develop the algorithm, often in collaboration with humanities scholars. It seems eminently reasonable to trust him. But if we are trusting him to 'do the math' for us, then how do we give him credit? To what degree is he a collaborator on or contributor to any scholarly work that uses topic models generated by MALLET? If we are relying on Blei's LDA and the notion of these collections of words that he termed 'topics', then what sort of contributor is he to research that builds on this approach? When we are trusting Mimno and Blei with all these calculations and algorithms, we should at least clearly document the use of 'Bleian topics' the way we would any other theoretical grounding or approach.[3]

Mark Marino, in his proposal that software be appraised and interpreted based on critical theory as 'Critical Code Studies', notes that software 'frequently has multiple authors, mostly uncited' (Marino 2006). He compares using a common algorithm to using a screw, and notes that 'mechanics do not cite the inventor of the screw every time they use one, although programmers at times attribute code to particular sources' (Marino 2006). I am not sure how many mechanics are in the habit of citing their sources or materials when they build something, but perhaps it would actually make sense to document the use of a screw rather than a nail when the screw was still a new technology, with some rationale or explanation for the choice. And in the world of software, there are so very many algorithms, and various implementations of those algorithms in different languages and frame-works (and in some cases, a choice of one platform over another could trigger a bug or behaviour that is not present in others). How can we possibly determine when an algorithm is so 'common' that it does not need to be cited? Certainly, a hash algorithm like MD5 is pervasive enough that we can probably consider it to be a 'common' algorithm (although it is certainly easy enough to name, and likely to be mentioned when used). But what about something like LDA? In good scholarship, it is important to document our sources, and that should include

tools and processes to some degree. But how much detail is enough? How far down the technology stack should those details extend?

Even if by some miracle we find a piece of software that works flawlessly, there are still assumptions embedded in the code or in the approach that will always affect results derived from that software. Bethany Nowviskie writes that 'even the most clinically perfect and formally unambiguous algorithmic processes embed their designers' aesthetic judgments and theoretical stances toward problems, conditions, contexts, and solutions' (Nowviskie 2015). Consider the pervasive 'desktop' metaphor for a graphical user interface (GUI), which organizes operations and file content based on the notion of paper documents on a workspace. The desktop is not the only option for an interface metaphor; for instance, as early as 1996 others proposed it be replaced by 'lifestreams', organizing content as a 'time-ordered stream of documents' (Fertig *et al.* 1996). Now, on mobile devices, the specifics of files and data have become more and more abstracted, so that the user does not need to think about where and how particular files are stored, or perhaps even think of content as files. How can we possibly determine how much these basic organizational principles affect the applications built on top of these interfaces, and how it consequently impacts the scholarship that is based on or makes use of such software? Who is to say whether the hierarchical organization of files into folders has had some influence on the hierarchical structure of XML, which we often see referenced in discussions of the limitations of TEI for describing texts, since it enforces a single structure (summarized in Schloen and Schloen 2014, 21). One could even argue that the organization of pages and URLs within websites are based on the same hierarchy of the file system that resulted in the folders and files of the desktop GUI. Perhaps the notion of content as sequential, filterable streams might have inspired different conceptual models.[4]

Given the layers of complexity in the world of software and the hardware it runs on, it seems inevitable that we must trust others to do to the math at some point, but perhaps we should be more cognizant of that fact. What checks are in place, and what credit should be given for software development that is done as part of or alongside scholarship? How is the software associated with scholarly work reviewed and assessed, and who is doing that work?

## Terms of use

Perhaps we could begin to approach the problematic position of software within scholarship by developing a common terminology for articulating and describing the roles that software plays in scholarly work, as well as some of the models that are used to create and develop that software. Such a terminology might also help us to better articulate the goals of any particular piece of software, the significance it bears for the scholarship it attends, and to determine what level of scrutiny or review is appropriate.

Software can be involved in any phase of scholarly work, from the beginning of a project to the end, whether for collecting or preparing data or materials, analyzing data or text through statistical, visual, or geographic methods, developing and comparing results, revising and revisiting work, and eventually sharing

results through publication, in whatever form that takes. Is the software essential in these various phases, and what influence does it have on the results? To go back to the car analogy, are we arriving at a destination we could have reached by other means, although perhaps more slowly? For some areas of research this is simply not the case; researchers are doing work that is hard to imagine without software to process vast amounts of data, or to inspire innovative new approaches. For example, Underwood and Sellers' study of trends in genre and literary diction from 1700 to 1850 analyzed 4275 works from three different sources (Underwood and Sellers 2012). On a different tack, Amanda Visconti's crowd-sourced *Infinite Ulysses* draws inspiration from 'online communities such as Reddit and Stack-Exchange' (Visconti 2015). In other cases, the same work *could* be done by other means; but realistically it would *not* be done without computers either because it would not occur to us to take a particular approach, or because doing it without automation would be too painfully slow and expensive.[5] According to a 2014 survey, 69% of UK researchers say their research would not be practical without software (Hettrick 2014). But in spite of the wide ranging use and significance of software to scholarly endeavour, we have yet to develop a precise vocabulary to articulate the significance of the digital nature of the scholarship.

Consider just one of these phases of scholarship that is aided and influenced by software: the seemingly straightforward case of publication. These days it is common for the publication or dissemination of scholarly work to take on digital form — whether through a fairly direct digital remediation of traditional paper publication formats using PDF documents; or standardized XML formats for inclusion in subject-based archives, such as PubMed Central in the USA; or more forward-thinking and experimental works that attempt to take greater advantage of the capabilities of digital publication. For this last category, software is intended to be more than a practicality and becomes a necessity. For instance, the journal *Vectors* claims to 'publish only works that *need* … to exist in multimedia' (Vectors Journal: Introduction 2015, original emphasis); the journal *Southern Spaces* aims to provide 'a form for innovative scholarship by taking advantage of the Internet's capabilities … and facilitating new ways of organizing and presenting research' (About Southern Spaces 2015); and the web authoring and publishing platform Scalar is designed to allow authors to take advantage of the 'unique capabilities of digital writing, including nested, recursive, and non-linear formats' (About Scalar 2015). For another case, consider *Socratic and Platonic Political Philosophy* by Christopher P. Long; the author describes the 'enhanced digital book' that is available from Cambridge University Press as 'an attempt to put the argument of the book into practice by creating a digital ecosystem in which readers are invited to engage publicly' (Long 2014). Even in the simplest cases, the mere fact of digital publication and availability can have a significant effect on the transmission or reception of scholarly work; in the more ambitious cases the digital form of the publication can take on some of the weight of the argument. But how do these publications credit and document the work that is done to implement these digital forms, much less test or review that work?

If we want to think of digital publishing as analogous to traditional, physical publication, consider that publishing companies always put their names and

logos on the works that they produce, and authors often include acknowledge-ments thanking editors and reviewers who helped shape the work. But the labour required to turn a manuscript into a physical printed copy is nearly invisible — even though the choices of font, layout, margins all have an impact on the reader's reception of a work. In those instances, where authors or editors claim that the digital form is essential to a work or embodies the argument, should not it be even more important to make that work visible and transparent? For the examples mentioned above, this is somewhat inconsistent. Pieces published in *Vectors* include both author and designer statements, but journal listings and editorial statements indicate that the piece is *by* the scholar, and the technology that drives the journal makes the actual code into a 'black box' that can only be inspected by its interface. *Southern Spaces* has no credits or technical information that I can find indicating the labour of compiling and creating the digital form of the articles, much less any description of the technologies used to run the site or prepare the materials, although some video pieces do credit the producers of the video; the site has a list of editors and editorial staff, but no mention of software or tech-nologists, and even specific editorial credits on individual pieces are missing. As with print publication, the labour of creating and publishing the digital content is rendered effectively invisible. Long's philosophy work is available as a print monograph, and the digital edition is described as 'enhanced', a term that suggests the digital version adds some extra and, presumably, non-essential functionality. The 'semantic web authoring tool' Scalar is a second-generation project, a successor to the experimental journal *Vectors* from The Alliance for Networking Visual Culture (ANVC), is intended to allow for long-form, impro-visational scholarly arguments. Tara McPherson, the ANVC lead, critiques digital humanities work that 'proceeded as if technologies from XML to databases were neutral tools' and questions whether the 'structures of digital computation' cordon off 'racial visibility and knowing' raises the question that (McPherson 2012, 142–3). And yet, surprisingly, over a third of the pieces listed in the Scalar showcase (Alliance for Networking Visual Culture » Showcase 2015) have no technical acknowledgements or only mention Scalar without specifics, and several others list names without specific roles. Scalar is at least an open source project, so the underlying code can be viewed. Since the project is hosted on GitHub we can easily see who has contributed to the code (Fig. 1),[6] although that really provides very little in the way of understanding the vision, decisions, and rationales that went into the software. If the digital form of publication is truly significant to the scholarship being presented, as each of these digital publications and platforms contend, then should not it matter who does the work of creating the digital edition? If the technologies are not neutral, then should not they be documented and exposed? And should not we have standardized ways to document and recognize that work? If the technologies are not neutral, then should not they be documented and exposed as well? Should not we consider whether and how the work and concomitant technologies ought to be reviewed alongside the more traditionally scholarly content? These concerns have been raised before, often in the context of evaluation, including work to 'draft a "Col-laborators" Bill of Rights' in order to provide 'fair, honest, legible, portable, and

prominently displayed crediting mechanisms' (Nowviskie 2011). But the Collaborators' Bill of Rights is still not widely adopted, and these problems are not yet resolved. (Figure ).

In addition to terminology for describing and standards for addressing the involvement and significance of software to scholarly efforts, we should also make a greater effort to document the processes and models used to create,
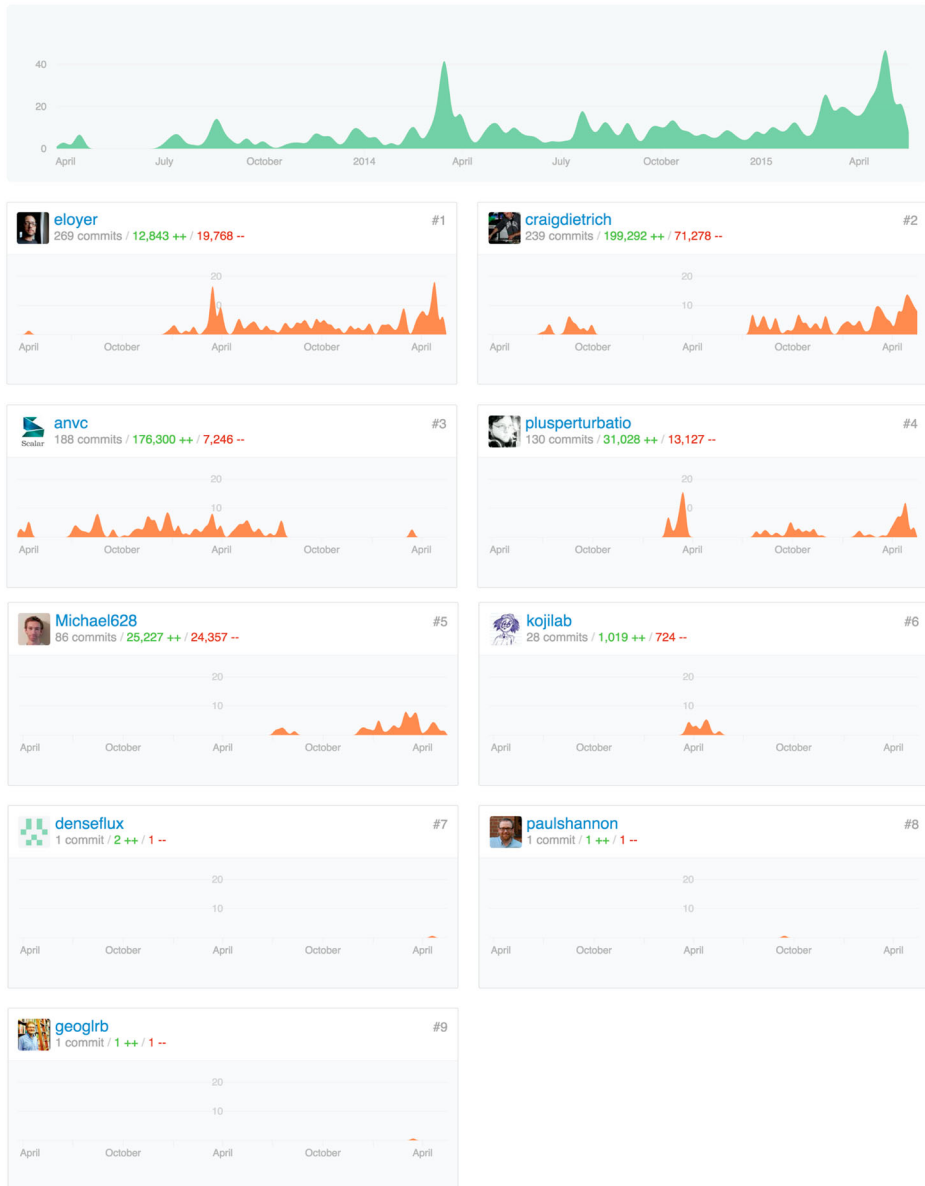


FIGURE 1    GitHub contributor graph for Scalar. https://github.com/anvc/scalar/graphs/contributors.

develop, extend, and test the software, since that may also have a bearing on the resulting scholarship.

One familiar mode of software development is something we might call the 'contractor' model. This term could indicate that development work is literally contracted out to an outside software development firm, but it may also apply when software developers are considered service providers rather than collaborators. This model can certainly be successful, and any number of scholarly projects have used this approach; but in my experience success requires communication and engagement, not just skill and expertise. For instance, I had a colleague who discovered that something as seemingly straightforward as creating a database interface for collecting data on poetry magazines revealed unexpected assumptions on one side and a lack of subject expertise on the other. The resulting problems ranged from simple annoyances, like awkward naming conventions and unclear organization, to larger concerns. For one set of fields, poor interface choices made it harder to enter the data correctly and resulted in duplicate records. In another case, the data model for locations was not properly thought out to handle the wide range of specificity present in the content, which included everything from street addresses to mentions of cities and countries. An engaged technical partner can help a less technical researcher think through the implications of their approach and possible limitations or concerns ahead of time that could affect their process or results. In this case, the user interface had an impact on the data collection, and any errors that are not caught and corrected will have an effect on any subsequent analysis. Even in something as simple as a data-entry site, the software developer creating the interface has to make any number of minute decisions about what fields should be displayed; what kinds of filters and searches should be available; and how to provide access to the data or display the connections between various kinds of content. All of these small, seemingly insignificant decisions accumulate into potentially profound impact on the labour and scholarship that is done on the basis of or with the use of that software. Some developers will make better choices than others, either instinctively or based on experience, skill, and tacit knowledge from other projects. Consequently, results in a contractor-based software development model can be variable, to say nothing of how they may be profoundly affected by time and funding constraints. Such limitations may not afford the opportunity for iteratively improving the software to meet scholarly expectations.

Another familiar model is the developer/scholar, who creates his or her own software along with their research. For examples of this, consider Ted Underwood or Matt Jockers, who work in the realm of large-scale literary analysis, and both of whom have developed packages in R for their work, among other software. Another example is Amanda Visconti, who created the 'participatory digital edition' *Infinite Ulysses* as part of her dissertation (Visconti 2015). This approach obviously has some clear benefits; any failure to translate the mental scholarly model into code cannot be blamed on communication difficulties that could be the culprit in other models, where team members have different vocabularies and competencies. Scholars who create software alongside more typically 'scholarly' endeavours should understand the code, and will of necessity be aware of at least

some of the other technologies and tools that they are relying on. To go back to Mimno's playful comment, they do not have to trust anyone else to 'do the math' for their research, although almost certainly they will be relying on tools, software, and algorithms created by others. All of the immediate credit or blame resides with the developer/scholar. This model has some downsides, however. A scholar operating as a solitary developer is less likely to have a strong technical support network or team, which can provide structures for and knowledge of industry best practices such as unit testing, code review, documentation. While these are not always found in other development environments, they are much more common in professional software development settings. If there are errors in the code, there might not be anyone else to catch them, because that code may never be reviewed (depending on the particular discipline), and in other cases it is never made public. In one scientist's case, 'the discovery of the bug in his software led to the retraction of three *Science* papers' (Goble 2014).[7] While a retraction is never desirable, it is still a better outcome — at least for the sake of knowledge — than not discovering a flaw, or discovering one but not making it public. Another drawback to the developer/scholar model is that it requires a select set of skills and knowledge, and not all scholars have or want to invest in acquiring the technical expertise. Returning to that 2014 Software Sustainability Institute survey of UK researchers, of a reported 56% of researchers that are developing their own code, 21% reported doing so with no training at all, and many others with limited training (Hettrick 2014). In the humanities in particular, it is still uncommon for people to have formal training in both humanities disciplines and software development or engineering. Perhaps, as Matt Kirschenbaum suggests, it is important to have more scholars begin formal study of technical languages to develop self-reliance and 'procedural literacy', just as scholars have studied foreign languages so that they need not rely on solely translations (Kirschenbaum 2009). However, in the sciences there is already concern about taking this too far. A researcher who becomes the 'resident software expert' may 'begin to write just code, and not papers for publication', and may eventually leave academia for industry rather than sacrifice career goals and salary (Hettrick 2015). Therefore, it is important not to privilege this particular approach as the only or best model for scholarly software development.

Some of my own experience has been with what I would describe as a 'co-author' model developed for software-based scholarship. In the 'Belfast Group Poetry | *Networks*' project, Brian Croxall and I effectively worked as co-authors, providing different expertise; we each took the lead on a different part of the project and deferred in other areas. While I did the majority of the software development, Brian was integral to the process in his role of testing code in development, helping to make decisions when choices came up, and uncovering discrepancies in the data that pointed at errors in the data processing. We also co-authored the analytical essays for the site, although the distribution of labour was different there than for the software code.

Yet another model to consider is the scholar as user of a software package designed to assist in a particular aspect of scholarly endeavour. Scholars now have available a wide range of tools available to them. In the sciences and social

sciences, statistical programs like Matlab, R, and SPSS are widely used (Hettrick 2014). Digital Humanities scholars often use software tools such as Voyant for text analysis, Gephi for network analysis, and MALLET for machine learning and topic modelling. These tools have varying degrees of user-friendliness and learning curves. Voyant has a very simple interface that makes it easy to get started. In Gephi, anyone with a spreadsheet can make a network graph, although learning how to use the interface and make use of all the embedded analytical and visualization algorithms can take much longer. In contrast, the difficulty in preparing texts for use with MALLET and analyzing the results poses a significant barrier to entry. But even in the simplest cases, choices are being made on behalf of the user — for instance, the default stop words in Voyant or MALLET. In many cases, these preselected choices can be configured and modified, but research shows that developers and designers are far more likely to customize their settings than average users, who are more likely to assume that default configurations have been carefully researched and selected (Spool 2011). This is a disconcerting notion, if it holds true for scholars as users of software. These necessary assumptions and choices make it all the more essential that even the seemingly small details of configurations and settings used in a software tool for scholarship should be documented and explained clearly.[8] It is important that humanities scholars engage with and question the software they use — that is, after all, the crux of their training. If providing a deep technical grounding for scholars is impractical, then at the very least software developers must work to give them a critical framework and models to approach the software they use more thoughtfully.

## [Re]viewing the code

A few years ago, I attended a reading by the poet Li-Young Lee. At one point he read a poem that I had inadvertently memorized due to repeat readings. At that point I had been working as a software developer for a few years, and it struck me that I had already developed a 'corpus' of work rivalling Lee's oeuvre, at least in sheer size. And yet no one will ever read my code or inadvertently memorize any of my lines, nor should they. In fact, in practical terms, the better I do my job, the less likely it is that someone will read my code, since the most likely reason for someone to trawl through code is because they have to do so in order to find and fix a bug.

  When software is integral to scholarship, how should the software be reviewed? As we have seen above, software is not always documented or credited clearly, and often the code cannot be read at all, much less examined in detail. Some might think that Critical Code Studies could be of use here, with the approach to the software as artefact and the code as text, looking at the human context as well as the machine. However, this approach takes time and expertise, and not every piece of code merits this kind of study. This is a valuable approach for historic and culturally significant pieces of software,[9] but perhaps few pieces of scholarly software deserve that approach — and even if some do, we may not yet know which will be long lasting or influential enough to become significant

and historic. From a practical standpoint, not every piece of software written to support scholarship is intended to last, be reused, or even be used by others. But even single-use code should follow some standards, so that research can be replicated and vetted, and research software cited. If we had a common terminology for the roles of scholarly software and models of development, perhaps we could arrive at simple criteria for determining what should be reviewed and how. In most cases, this review need not take into account elegance of approach or speed, but rather the extent and accuracy of assumptions. However, we cannot review what is not exposed, what has never been made viewable.

As part of my project with Brian Croxall, 'Belfast Group Poetry|*Networks*' (Belfast Group Poetry|*Networks* 2015) we took the time to document the automated data processing involved in the project (Koeser 2015). The project makes use of linked open data, harvesting information from archival finding aids for Irish Literature collections at Emory University's Stuart A. Rose Manuscripts, Archives, and Rare Book Library along with information from XML-encoded versions of the typescripts from the Belfast Group meetings. Our work connects the data and generates network graph visualizations as a way of investigating the relationships within a particular literary community. In this data-processing document, we detail the steps taken in the software as it collects and massages that data for presentation on the website and display in network graphs, including, rather critically, the places where we had to intervene to include people we considered to be significant and who would be otherwise invisible (or nearly invisible) based on the data and processes we used. We found it particularly critical to take the time to document and explain this part of the project, something that is not always done. However, the rest of the technology we have used in constructing the site is simply mentioned and linked on a credits page. We have not written anything on why we use the force-directed network graph layout algorithm in preference to other graph layouts (although it is a common enough choice for work of this type), and we only provide a brief statement about what the alternate chord diagram layout provides as a different representation of the same data (Fig. 2). The source code for the website and some of the tools that were used to create it are available on GitHub (Koeser *et al.* 2016), and those are linked and available, but it is difficult to make it clear that the software and methods are a significant part of the scholarly output of the project.[10] When we presented a preliminary version of this work at the 2013 Digital Humanities Conference in Lincoln, Nebraska, we included references to the software in our abstract (Koeser and Croxall 2013), but not with any expectation (or, realistically, desire) that the code would be reviewed alongside the more typically scholarly components of the project. This is a problem the *DH Commons* Journal is attempting to address by providing a venue for mid-stage project review. Yet the editors admit it is often a struggle to find reviewers who can assess both the humanities contributions and also the 'scholarship embedded in their methods or technical practices' (Issue 1 | DH Commons 2015, 1). That difficulty is one reason why this approach is still relatively new and uncommon for Digital Humanities work.

Reviewing conference and publication submissions can already be a daunting, time-consuming process that requires expertise and investment. Proposing
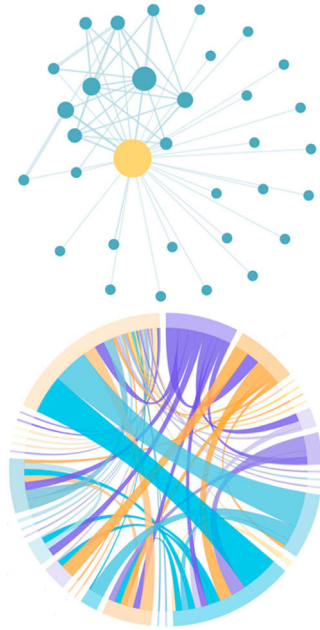
FIGURE 2    Two representations of the same dataset from Belfast Group Poetry|*Networks*.

additional work to that review process seems unrealistic and expensive, especially considering the additional knowledge and expertise that would be required. However, if we could implement some kind of code review alongside the other, more traditional scholarly outputs, it should help to make the role of software and software development more transparent in the work of scholarship. This would require that such work be consistently documented and credited. The code would not necessarily need to be released as open source, or be particularly polished, as long as there are standards and terminology clearly describing the purpose and end goal of the work. If scholars must rely on others to 'do the math' of developing the software that is used in their work, then perhaps it is also logical that they should begin to rely on others to review and assess that portion of the scholarly work, and to check the accuracy and appropriateness of those hidden calculations. But this may be difficult to bring about. Software developers without a scholarly background may not be eager to get involved in the review and assessment process, and those who do may find it difficult to focus on the intellectual content and assumptions rather than software development best practices and implementation specifics. And demanding additional work of those few scholar-developers who do have the expertise would be an additional and undue burden. This is yet another reason why it is important work to advocate for research software engineers to be recognized and provided with a long-term career path that will keep them involved in scholarly work within academia (Hettrick 2015).

Software is ubiquitous, and ever more important to scholarship, but this fact is not always recognized or brought to the fore, particularly in humanities fields.

Software is flawed, makes assumptions, and current algorithms may well be replaced by newer and better versions. We are constantly trusting others to 'do the math' at some point, but we would do well to make sure our trust is well-founded. Scholars should be more thoughtful about the software they use in their work, learn about its limitations, and think critically about the implications for their results. Both scholars and creators of software should take advantage of resources offered by groups like the Software Sustainability Institute and Software Carpentry and seek to extend their work to other disciplines. The people who develop software used for scholarly work and research are important to that work. The code and its authors should both be made more visible; we need to find ways for that technical work to be credited and cited, and reviewed along with the scholarship, so that our trust in those 'doing the math' is clear, explicit, and justified.[11]

## Acknowledgements

## Notes

[1] To be fair, Mimno 'argues that those intending to implement topic modeling should understand the details' behind it (Meeks 2013), and the participation of David Blei, Mimno and other computer scientists in workshops and presentations such as this one underscores their concern that humanities researchers need some understanding of the tools; but this statement indicates there is a point at which they may not be able to follow the details.

[2] Blei's comment was meant somewhat humorously, and evoked laughter in the room, but it was not, I think, intended ironically. As I discuss later, Mimno's background and expertise make it very reasonable to trust him — but I found the statement striking all the same, particularly as it came from an event intended to address humanists' lack of 'a clear understanding of the underlying statistical methods and models' (About | Topic Modeling 2016).

[3] My thanks to the anonymous reviewer who suggested this comparison, writing that 'Blei is the contributor to the same extent that Freud is a contributor if we make a Freudian reading of humanistic phenomena, or that E.P. Thompson is if we make a Thompsonian Marxist reading … he isn't a contributor, but an exponent on an influential method or practice we have followed'.

[4] Many of the major players in the Web 2.0, such as Twitter, Facebook, and YouTube, and even blogs and RSS feeds to some extent, now provide streams of content ordered by date rather than hierarchical web pages often seen in more traditional websites, but I would argue this is a newer development and not yet as pervasive and influential as the desktop metaphor and associated hierarchical file systems.

[5] The *Historical Thesaurus of the Oxford English Dictionary*, as described by Marc Alexander in his presentation on the history of the English language at the 2012 Digital Humanities conference in Hamburg, is an interesting, and I think a rare, counter-example, where the content of the thesaurus was painstakingly and manually compiled over decades, relying in part on student labour (Alexander 2012).

[6] Even here, there are some caveats: the GitHub contributors page only displays commits from users who have accounts on GitHub; if an account is deleted, that user is no longer visible as a contributor, although their commits are still visible in the history of the code.

[7] Geoffrey Chang's retraction explains that 'an in-house data reduction program introduced a change in sign for anomalous differences' (Chang *et al*. 2006). In another description of the

incident, five papers were affected by 'a home-made data-analysis program' that 'flipped two columns of data' (Miller 2006).

[8] The 2012 issue of the *Journal of Digital Humanities* on topic modelling includes reviews of MALLET (Graham 2013) and Paper Machines (Crymble 2013); both approach and discuss the tools from a user perspective, and make no apparent critique or review of the underlying software code itself, although they do raise concerns about lack of documentation and user support.

[9] For instance, the author-annotated copy of the original jQuery source code (Resig 2015) provides a fascinating look into the beginnings of the now prevalent Javascript library.

[10] Perhaps the most significant, since the approach and the software are the work most likely to be re-used.

[11] This article was drafted on two successive Apple MacBook Pro laptops, usingSublime Text and Microsoft Word, typed using the Colemak keyboard layout. Web content used for research was largely accessed with Chrome, web searches were conducted using Google, and citations were harvested and generated with Zotero. Numerous other technologies, protocols, and infrastructure were used to store, transmit, transform, and display it. No custom software was developed for the creation of this article.

# References

About | Topic Modeling. 2016. Accessed February 2. http://mith.umd.edu/topicmodeling/about/

About Scalar. 2015. Accessed May 17. http://scalar.usc.edu/scalar/

About Southern Spaces. 2015. Accessed May 17. http://southernspaces.org/about

Alexander, Marc. 2012. Patchworks and field boundaries: visualizing the history of English. Presented at the Digital Humanities 2012, Hamburg, Germany, July 18.

Alliance for Networking Visual Culture » Showcase. 2015. Accessed May 18. http://scalar.usc.edu/scalar/showcase/

Belfast Group Poetry|Networks. 2015. *Belfast Group Poetry|Networks*. http://belfastgroup.digitalscholarship.emory.edu/

Chang, Geoffrey, Christopher B. Roth, Christopher L. Reyes, Owen Pornillos, Yen-Ju Chen, and Andy P. Chen. 2006. Retraction. *Science* 314(5807): 1875b–1875b.

Crymble, Adam. 2013. Review of Paper Machines, Produced by Chris Johnson-Roberson and Jo Guldi. *Journal of Digital Humanities*. Accessed April 4. http://journalofdigitalhumanities.org/2–1/review-papermachines-by-adam-crymble/

Fertig, Scott, Eric Freeman, and David Gelernter. 1996. Lifestreams: an alternative to the desktop metaphor. Accessed April. http://www.sigchi.org/chi96/proceedings/videos/Fertig/etf.htm

Goble, Carole. 2014. Better software, better research. *IEEE Internet Computing* 18(5): 4–8. idoi:10.1109/MIC.2014.88.

Graham, Shawn. 2013. Review of MALLET, Produced by Andrew Kachites McCallum. *Journal of Digital Humanities*. Accessed April 4. http://journalofdigitalhumanities.org/2-1/review-mallet-by-ian-milligan-and-shawn-graham/

Hettrick, Simon. 2014. It's impossible to conduct research without software, Say 7 out of 10 UK Researchers. *Software Sustainability Institute*. Accessed December 4. http://www.software.ac.uk/blog/2014-12-04-its-impossible-conduct-research-without-software-say-7-out-10-uk-researchers.

Hettrick, Simon. 2015. Why we need to create careers for research software engineers. *Scientific Computing World*. Accessed November 11. http://www.scientific-computing.com/news/news_story.php?news_id=2737.

Issue 1 | DH Commons. 2015. *DH Commons*. Accessed July. http://dhcommons.org/journal/issue-1.

Jockers, Matthew L., and David Mimno. 2013. Significant themes in 19th-century literature. *Poetics* 41(6): 750–69. doi:10.1016/j.poetic.2013.08.005.

Kirschenbaum, Matthew. 2009. Hello Worlds. *The Chronicle of Higher Education*. Accessed January 23. http://chronicle.com/article/Hello-Worlds/5476.

Koeser, Rebecca Sutton. 2015. Belfast Group Poetry|Networks : About the Data. *Belfast Group Poetry|Networks*. http://belfastgroup.digitalscholarship.emory.edu/network/about/.

Koeser, Rebecca Sutton, and Brian Croxall. 2013. Networking the Belfast Group through the automated semantic enhancement of existing digital content. *Journal of Digital Humanities* 2(3). http://journalofdigitalhumanities.org/2-3/networking-the-belfast-group-through-the-automated-semantic-enhancement-of-existing-digital-content/.

Koeser, Rebecca Sutton, Brian Croxall, and Kevin Glover. 2016. *Belfast-Group-Site: Belfast Group Poetry|Networks Website Initial Release*. doi:10.5281/zenodo.45121.

Long, Christopher. 2014. Birthing the Digital Book. *Christopher P. Long*. Accessed November 23. http://www.cplong.org/2014/11/birthing-the-digital-book/.

Marino, Mark C. 2006. Critical code studies. *Electronic Book Review*. Accessed December 4. http://electronicbookreview.com/thread/electropoetics/codology

McPherson, Tara. 2012. Why are the digital humanities so white?. In *Debates in the Digital Humanities*, NED - New edition, 139–60. University of Minnesota Press. http://www.jstor.org/stable/10.5749/j.ctttv8hq.12.

Meeks, Elijah. 2013. The digital humanities contribution to topic modeling. *Journal of Digital Humanities*. Accessed April 9. http://journalofdigitalhumanities.org/2-1/dh-contribution-to-topic-modeling/

Miller, Greg. 2006. A scientist's nightmare: software problem leads to five retractions. *Science* 314(5807): 1856–7.

Mimno, David. 2012. The details: how we train big topic models on lots of text. presented at the Topic Modeling for Humanities Research, Maryland Institute for Technology in the Humanities, College Park, Maryland, Accessed November 3. https://vimeo.com/53080123

Nelson, Robert. 2011. Mining the Dispatch. *Mining the Dispatch*. Accessed September 7. http://dsl.richmond.edu/dispatch/pages/intro

Nowviskie, Bethany. 2011. Where credit is due: preconditions for the evaluation of collaborative digital scholarship. *Profession* 2011(1): 169–81. doi:10.1632/prof.2011.2011.1.169.

Nowviskie, Bethany. 2015. A game nonetheless. *Bethany Nowviskie*. Accessed March 15. http://nowviskie.org/2015/a-game-nonetheless/

Resig, John. 2015. Genius-annotated version of 'http://ejohn.org/files/jquery-Original.html.'" *Ejohn.org*. http://genius.it/ejohn.org/files/jquery-original.html.

Rivest, R. 1992. The MD5 Message-Digest Algorithm. Accessed April. http://tools.ietf.org/html/rfc1321

Schloen, David, and Sandra Schloen. 2014. Beyond Gutenberg: transcending the document paradigm in digital humanities. *Digital Humanities Quarterly* 008(4). http://www.digitalhumanities.org/dhq/vol/8/4/000196/000196.html

Spool, Jared. 2011. Do users change their settings? User interface engineering. *UIE Brain Sparks*. Accessed September 14. https://www.uie.com/brainsparks/2011/09/14/do-users-change-their-settings/

The Heartbleed Bug. 2015. *The Heartbleed Bug*. Accessed May 8. http://heartbleed.com/

Underwood, Ted, and Jordan Sellers. 2012. The emergence of literary diction. *Journal of Digital Humanities*. Accessed June 26. http://journalofdigitalhumanities.org/1-2/the-emergence-of-literary-diction-by-ted-underwood-and-jordan-sellers/

"Vectors Journal: Introduction." 2015. Accessed May 17. http://vectors.usc.edu/journal/index.php?page=Introduction

Visconti, Amanda. 2015. *'How Can You Love a Work, If You Don't Know It?': critical code and design toward participatory digital humanities*. College Park: University of Maryland. http://dissertation.amandavisconti.com/.

Yao, Limin, David Mimno, and Andrew McCallum. 2009. Efficient methods for topic model inference on streaming document collections. In *KDD*. http://www.cs.umass.edu/mimno/papers/fast-topic-model.pdf.

## Note on contributor

Rebecca Sutton Koeser is a senior software engineer with Emory University Library and IT Services, and has a PhD in English Literature. She has contributed to a diverse array of digital library and digital humanities projects and open source tools, including Belfast Group Poetry|Networks, Serendip-o-matic, and Readux.

Correspondence to: Rebecca Sutton Koeser. E-mail: rebecca.s.koeser@emory.edu; http://rlskoeser.github.io/