



# Task-Driven Programming Pedagogy in the Digital Humanities

David J. Birnbaum  and Alison Langmead 

**Abstract** In this chapter, we advocate for a task-driven approach to teaching computer programming to students of the digital humanities (DH). Our perspective is grounded first in Birnbaum's (2014) plenary address to the University of Pittsburgh Faculty Senate (Birnbaum 2014), in which he argued that coding, like writing, should be taught across the liberal arts curriculum in domain-appropriate ways. This position argued that (1) coding is not an esoteric specialization to be taught solely by computer scientists, and that (2) coding might be taught most effectively in the context of different disciplines. Here, we present a method for embedding Digital Humanities education, and more specifically programming pedagogy, within the long-standing traditions of the Humanities and argue that this approach works most effectively when new learners have access to context-specific mentorship. Our second point of reference lies with oral-proficiency-oriented (OP) foreign language pedagogy. Within an OP model, the ability to communicate in a foreign language is a skill, and the primary goal for learners who seek to acquire that skill is not an academic understanding of the grammar of a language, but, instead, the ability to function successfully within realistic contextualized human interactions. Seen from this perspective, computer-programming curricula organized around the features of the programming language might be compared to older grammar-and-translation foreign-language pedagogies. What we advocate instead is that the ability to *use* a programming language (*programming proficiency*) is best acquired in the context of performing contextualized, discipline-conscious tasks that are meaningful to humanists, an approach that has parallels to OP language learning.

**Keywords** Digital humanities · Computational humanities · Programming pedagogy · Humanities pedagogy · Interdisciplinary · Computing education

---

D.J. Birnbaum (✉) · A. Langmead  
University of Pittsburgh, Pittsburgh, PA, USA  
e-mail: [djbpitt@pitt.edu](mailto:djbpitt@pitt.edu)

A. Langmead  
e-mail: [adlangmead@pitt.edu](mailto:adlangmead@pitt.edu)

## Introduction

Computer programming isn't computer science. Computer programming is more like writing. Everyone can learn to do it, and can be given the opportunity to learn to do it in ways that are appropriate for their disciplines ... You don't have to be an English composition professor to know how to write, and how to teach writing, in your discipline. (Birnbaum 2014, p. 2)

Digital humanists perform humanities research, not computer science research and not, except as a means to an end, software development. The Digital Humanities (DH) path from conceiving a research question through identifying and applying appropriate methods to interpreting the results will sound familiar to humanities researchers because it *is* familiar: the DH toolkit may be new, but to the humanities scholar and student, Digital Humanities is a way of doing humanities, not a way of doing computing. In order to do our work, we select the most appropriate tools from *all* of the resources at our disposal, from the written essay to data models to GIS to *tfidf*. Traditional computer-science concerns like data structures and algorithms and computational complexity may underlie some of what we do, but they are not typically our primary objects of study.

We recognize that integrating digital tools into humanities training is not yet widespread practice, but we also believe that this might result from a misconception that digital tools are something foreign that we import from outside the humanities. In fact, many of the programming tasks required to obtain research results in the computational Digital Humanities can be met with training that is narrowly focused, task-driven, and produced within the context of training in the humanities, especially when explicit DH mentorship is available. Humanities scholars without prior technological training or expertise can, without explicit attention to or consideration of many of the scientific issues that are fundamental to the discipline of computer science, easily learn to use and write programs that yield meaningful insights into their data.

Two types of Digital Humanities practice constitute our focus in this chapter: the production of new knowledge through the application of computational tools to the analysis of (not necessarily verbal) cultural texts, and the production, with the assistance of computational tools and methods, of new resources for conducting research. We do not mean to suggest that these constitute an exhaustive or exclusive definition of the practice of Digital Humanities, but because the present volume engages specifically with *computing education* and *computing across the disciplines*, the focus of our contribution is on *computational* Digital Humanities, that is, on embedding computing in the humanities.

### *The Tool We Know*

The tool that humanists have traditionally used both to process and to convey their ideas and interpretations to their audience is the written essay. The written essay is a

masterful tool. Whether in the form of an article or the form of the book, the academic essay has successfully served the humanities for the length of its existence, and it is expected to continue to do so equally as elegantly for the foreseeable future. The rhetorical design of a well-written essay allows an author to lead the reader through a sometimes complex maze of thoughts, connections, and ideas in a clear and organized fashion. The much-maligned five-paragraph essay model, with its “Tell them what you are going to say, then say it, then tell them what you said” motto, may be the most simplistic example of this tool, but it has the benefit of being quickly understood by readers, most of whom will not only easily follow the pattern of argumentation while reading, but also recognize the design for what it is, having been taught the method themselves in school.

This is not to say that producing a traditional written essay is easy. It has a steep learning curve and its own particular perils. From the panic of staring at a blank page (or a blank screen) to managing digressions and asides without losing one’s place to writing oneself into a rhetorical corner that is difficult to escape, the challenges to writing successful analytical essays are omnipresent. Written argumentation is a complex tool that takes skill and practice to master, and we are not expected to master it quickly or all at once.

Our training in how to use this tool begins as soon as we learn to read and write, that is to say, in elementary school. Slowly but surely, throughout K-12 education, we learn the principles of writing grammatically correct sentences that can be put in a particular order so that they might best convey to others the ideas that we have in our heads. Before students reach college, they will have been exposed to—if not mastered—the principles of effective analytic writing. They will then be expected to practice this tool throughout college in different subject areas. Indeed, students will take their ability to produce essays from their Japanese history classes to their Russian fairy tales classes to their Renaissance art history classes, applying this tool to each successive domain and using it to demonstrate and communicate what they have learned of the subject.

From this vantage point, the introduction of the Digital Humanities only at the university level seems greatly disadvantageous.<sup>1</sup> Having spent over a decade learning to use the written essay, college-level students of the Digital Humanities are expected to master not just one new tool, but an entire digital arsenal of techniques in a few short years, and to do so at a time when it has traditionally been expected that they would be using an already familiar, if not yet entirely mastered, written rhetorical apparatus in order to ingest, process, and create increasingly complex content. Introducing digital skills only at the *graduate* level further exacerbates the situation, as these students are assumed already to have acquired a

---

<sup>1</sup>The twenty-first century has seen a welcome attention to teaching *computational thinking* from a young age and in a way that is not explicitly coupled with the simultaneous teaching of programming languages (e.g., Wing 2006). Computational thinking is obviously relevant to programming, and our focus specifically on teaching the use of programming languages in a volume about programming pedagogy should be understood as complementary to, rather than in disagreement with, teaching computational thinking.

basic humanities education, and the focus of their graduate studies is supposed to be on acquiring advanced content knowledge and producing original research results. It has been our experience that asking graduate students to learn the techniques and tools of the Digital Humanities feels as if we are asking them to start a new line of inquiry from scratch at a time when they are under pressure to perform and demonstrate mastery. The written essay is the entrenched foundation of the humanities. Question its centrality, and students can feel lost, voicing their frustration as “taking a step back” or “starting all over again.” So how might students contextualize and repurpose their established skills as essay writers in a new computational context?

## Functional Requirements for Getting Started on a DH Problem

Students who find themselves in this position deserve our empathy; as newcomers to Digital Humanities, they will have much to learn that might seem entirely alien and alienating. In fact, what is new is often much more the digital context than the actual research practice. After all, Digital Humanities is still humanities,<sup>2</sup> and, “[l]ike writing, programming provides a way to think in and through a subject” (Ramsay 2012, p. 229). Even if it does have its own particular difficulties and pitfalls, the research process of performing Digital Humanities is not fundamentally different from traditional approaches, and this can be emphasized to new entrants in the field. If newcomers are fortunate enough to have a DH mentor—and we would like to suggest that providing access to this type of guidance is absolutely critical to the nourishment and growth of Digital Humanities—the mentor can guide them through this process, pointing out what is authentically new and what is primarily a reframing of traditional research methods within a new context.<sup>3</sup> Here is a sketch

---

<sup>2</sup>Dave Perry discusses alternative ways of engaging in Digital Humanities, and what unites those perspectives is a contextualizing of digital humanists primarily as humanists who engage with digital methods or materials, rather than as computer users who are interested in the humanities (Perry 2012).

<sup>3</sup>Our discussion in this chapter presumes a DH mentor because we are writing about curriculum and about pedagogy, but the DH mentor may be a teacher, a workshop facilitator, a professional colleague, or a fellow learner. Not all would-be digital humanists have equal access to training and education, and especially in contexts where one-on-one DH mentors may not be available, new learners will find themselves welcome in online DH communities, such as the Digital Humanities Questions and Answers Board of the Association for Computers and the Humanities (<http://digitalhumanities.org/answers/>), which also runs an organized mentoring program (<http://ach.org/activities/mentoring/>), or the TEI-L mailing list run by the Text Encoding Initiative (see <http://www.tei-c.org/Support/> for information about subscribing and searching the archives). For further information about mentoring opportunities, see also Lisa Spiro’s “Opening up Digital Humanities education” (Spiro 2012, esp. “Coaching,” pp. 353ff), which extends her earlier work (Spiro 2011).

of the typical process of pursuing computationally inflected research in the humanities<sup>4</sup>:

1. Have a humanities-based question that you would like to explore.
2. Be familiar with not only the scope, but also the types of evidence that are available to you.
3. Be aware of one or more digital approaches that will be effective in helping you explore your question. Select the approach that is best suited to the available evidence.
4. Identify a digital tool that implements your preferred approach and that is suitable for your needs, skills, and time commitment. If no suitable tool is available, create one, which may mean first acquiring the skills to create one.<sup>5</sup>
5. Acquire the tool and install it onto a machine you know how to control.
6. Using an iterative approach, learn to use the tool well enough that you can apply it effectively to your question.
7. Interpret your results thoughtfully, appropriately, and thoroughly.

Creating such a list has the unfortunate rhetorical effect of suggesting that its order might somehow be dogmatic. This is not the case here, but we do wish to argue that this particular order makes coherent sense, and our discussion below is organized around it.

### ***Step 1: Have a Humanities-Based Question***

In any type of academic research, beginning with a question in mind is not necessarily self-evident, nor is it always the best place to start. It has been our pedagogical experience, however, that humanists who are new to Digital Humanities often have the greatest success by beginning with a research question. Trevor Owens wrote an insightful piece in 2014 entitled “Where to Start? On Research Questions in the Digital Humanities,” which briefly elucidates the intricate dance between tools and questions that scholars, including digital humanists, perform in practice (Owens 2014).<sup>6</sup> It has rarely been the case that research questions spring fully formed from the scholarly brain, like Athena from the head of Zeus. As

---

<sup>4</sup>These steps have been honed through practice and iteration over the course of the past few years in Alison Langmead’s pedagogical work teaching the Digital Humanities to graduate students in both the School of Information Sciences and the Dietrich School of Arts and Sciences at the University of Pittsburgh (Langmead 2015, 2016).

<sup>5</sup>See “Yes, you can build your own tools,” below.

<sup>6</sup>This particular conversation about beginning with tools or questions was instigated by a Tweet from Tom Scheinfeldt asking the community for advice for new DH’ers (Scheinfeldt 2014) and its concomitant replies. Also relevant is Scheinfeldt’s contribution to the 2012 *Debates in the Digital Humanities* volume (Scheinfeldt 2012).

humanities researchers, we work with the knowledge and tools we have, from the archives to the essay, to craft our questions and our responses, and the questions themselves frequently evolve as part of that process. Yet starting from a research question can serve to contextualize and motivate the acquisition of technological skills. As Simon Mahony and Elena Pierazzo write, “digital humanities *teaching* needs to be relevant to the students’ studies or research interests” (Mahony and Pierazzo 2012, p. 224).<sup>7</sup> In a DH context, research questions scope out a terrain and give humanists an investigative center from which they can explore unfamiliar digital pathways. They can serve as a touchstone to return to when the path ahead seems unfamiliar.

On the other hand, as scholars become more familiar with the scope of digital approaches that exist, playing around with the tools can then take on its own importance. Perhaps the best-known paean to allowing research questions to grow and change in response to non-directed play is Stephen Ramsay’s “The Hermeneutics of Screwing Around; or What You Do with a Million Books,” which asks “whether we are ready to accept surfing and stumbling—screwing around, broadly understood—as a research methodology” (Ramsay 2010, p. 7).<sup>8</sup> Here, Ramsay discusses the joys of search, but also the joys of browse, as scholars allow themselves to consider alternatives and see where their tools might lead them that they might not have visited otherwise. The focus of Ramsay’s essay is on the exploration of existing reference resources (he contrasts traditional, organized bibliographic searching to impetuously following serendipitous links in a web browser, unconstrained by an original research question), but the context can be adapted from discovering existing content in books or on the Web, without a consistent, directed goal, to creating new exploratory views and visualizations with analytical digital tools and programming languages, in this case without a consistent, directed research question.

It is our experience that this exploration (or play) is most productive when fluency (or, at least, operational comfort) with some digital approach has already been achieved. For DH newcomers who have not yet developed this fluency, the role of the DH mentor in the formulation of a research question may be important for avoiding false starts and dead ends, whether the guidance provided is general and limited, concentrating on perspectives and methods with broad application in Digital Humanities, or domain specific, in situations where the mentor happens to share a research interest with the learner. The process of creating an insightful question in the humanities—with or without a digital inflection—is both challenging and of paramount importance.

---

<sup>7</sup>Emphasis added. We return to the crucial role of domain-relevance in “Task-driven programming pedagogy,” below.

<sup>8</sup>See also Jentery Sayers’s work, which compares teaching code to teaching literature or language (Sayers 2012, esp. pp. 289–91).

## ***Step 2: Be Familiar with the Scope and Types of Evidence You Have Available***

Step 2, or knowing what type of evidence you can bring to bear on your question, does not deviate from the humanities research tradition either, although the digital context does call scholars' attention to both the content of the evidence that they can use to make their arguments and its structure or format. Newcomers to Digital Humanities may not know initially which types of evidence will be most productive in a digital context, but the more they know about the data they have at their disposal, the more effective their engagement with that data will be. DH mentors can again be of great help at this stage, because they can facilitate a conversation about the types of evidence that are best suited to digital approaches, and can help the new DH scholar understand the time and expense involved in certain digitization processes—for example, the highly time- and labor-consuming task of extracting fielded data from a narrative text.

## ***Steps 3, 4, and 5: Identifying a Digital Approach and Identifying and Acquiring a Digital Tool***

Neither step 3 (being aware of your digital options) nor step 4 (selecting the most promising of those options) needs to deviate from the process that a humanist might use when producing a traditional written essay, outside the digital context. Furthermore, all three of these steps, including step 5 (using your chosen digital tool), could be considered identical to their non-digital counterparts if one simply replaces “digital approach” with “rhetorical strategy” and “digital tool” with “essay format.” However, at these steps, the learning curve for understanding that the various digital approaches will require of a scholar is steeper than that of selecting a rhetorical strategy, in part because a variety of rhetorical strategies have been introduced to scholars throughout their educational careers in ways that digital approaches have not, but also in part because digital approaches are not truly one type of tool. The digital computer was touted as the universal machine for a reason—it can process any idea that can be written as an algorithm. This has created a vast domain of options in a context where different digital techniques require vastly different sets of skills. For example, the scholar who wishes to implement topic modeling will need a different set of learning experiences from the scholar who wishes to use GIS to map some form of geographic change over time. Types of digital methods differ more from one another than types of written essays—or, at least, that will be the perception of a scholar who is new to digital technologies but experienced in writing essays.

It is especially in steps 3, 4, and 5 that the role of the DH mentor can come to the fore. Since the decision space inherent in step 3 is extremely large, there is no reason to assume that any given humanists will know what sorts of digital approaches they might bring to bear on their questions, much less what precise tools they will then need to know how to use—whether that be MySQL, Python, Excel, or something else.<sup>9</sup> DH mentors are important at this juncture to offer ideas and explanations, as well as—most critically—to steer scholars away from well-known pitfalls, such as when the approach seems exciting at first, but the available data are either lacking or inappropriate to it. DH mentors also need to balance the pre-existing strengths and skills of the scholar with the training and time available to progress from ideation to actualization of the research methodology.<sup>10</sup>

Step 5, acquiring the tool and installing it on a machine that the researcher controls, is a question not only of time and desire, but also often of financial resources. Familiarity with command line interfaces is something that any DH researcher with a laptop can acquire, but having access to a Linux machine fully connected to the Internet is not merely a question of hard work and persistence.<sup>11</sup> Many high-quality DH projects operate entirely with free (that is, no-cost) software, but that is of little use to a newcomer who is baffled by the documentation (or perhaps by the absence of documentation). This is another area where the DH mentor can be of great assistance. Selecting approaches and tools that are within the current logistical constraints is a critical component of DH research. Much frustration and many abandoned projects can be avoided by matching what is desired in the abstract to what is possible in the real world. We will say more about selecting tools below, where we also contrast selecting tools with building your own.

---

<sup>9</sup>Two of the most extensive inventories of tools available to the Digital Humanist are Alan Liu's DH Toychest, <http://dhresourcesforprojectbuilding.pbworks.com/w/page/69244319/Digital%20Humanities%20Tools>, and the DiRT Directory, <http://dirtdirectory.org/>. Together, these resources list hundreds of tools among which humanists can browse and search, a quantity that can feel overwhelming, especially to someone new to the field [and Liu's Toychest even includes a section entitled, "Other Tool Lists" (<http://dhresourcesforprojectbuilding.pbworks.com/w/page/69244319/Digital%20Humanities%20Tools#othertoollists>), suggesting that the list could be extended]. Hypertext was designed to allow for an infinitely extensible web of logical connections, which means that it is well suited to representing the realm of interconnected lists of DH tools.

<sup>10</sup>Time constraints play an obvious special role in learning environments that are tied to an academic calendar. In our courses, where students must progress from no prior technological knowledge or experience to publishing a completed project on the Internet at the end of a single fifteen-week semester, we often encourage proof-of-concept implementations. In situations where the project as conceived would require data preparation at a scale that is not realistic within an academic semester, reframing the goals as a proof-of-concept implementation allows the learners to prioritize mastering new tools and skills while working with small, illustrative data, which they may or may not then augment after the conclusion of the course.

<sup>11</sup>The availability of hardware and Internet connectivity is mediated economically, and not all learners will have access to first-world resources. See <http://go-dh.github.io/mincomp/> for information about *Minimal computing*.



## ***Steps 6 and 7: Learning the Tool and Interpreting the Results***

Steps 6 and 7 are, again, no different from traditional humanities methodologies. Knowing your question, understanding your evidence, your preferred methodological approach, and the tool(s) you will use to perform your work is a responsible way to undertake both digital and non-digital research projects. Acquiring the tools and then taking the time to learn how to work with them in the context of your own domain are equally important—and, for the humanist scholar, equally familiar—steps on the path to creating effective, creative, thoughtful research. Learning how to interpret the results (including the output of the tools you have used) in an attentive, appropriate, and thorough manner is the work of the humanities itself. It is no different from any other way of performing humanities research.<sup>12</sup> Being a beginner at an otherwise advanced stage of one's education can be a frustrating experience; what can help ameliorate the frustration is recognizing that even a novice in the D of DH can draw on a substantial education and background in the H.

## **Yes, You Can Build Your Own Tools**

As we argued earlier, computationally inflected research in the humanities, like research in the humanities in general, typically seeks to engage with, explore, and answer research questions. In some instances, the principal output of the research may be an analytic report, while in others the result supports analytic inquiry by others without foregrounding its own analysis. This second type of output is a familiar paradigm in non-digital humanities scholarship, such as scholarly editions of texts, where scholarly analysis on the part of the editor informs every stage of the selection, transcription, evaluation, analysis, and presentation of primary material, but where those scholarly interventions and interactions are embedded in a research resource other than a narrative essay. Transplanted to a digital context, in a research report we might ask and answer our own question using digital tools and methods and present the results of that analysis in narrative form, while in a digital edition our goal might be to create new resources that will enable others to explore and query materials in ways that would not be possible without the use of digital methods. In either case, the application of digital tools and methods is fundamental to conducting or facilitating humanistic inquiry.<sup>13</sup>

---

<sup>12</sup>Stephen Ramsay and Geoffrey Rockwell explicitly compare the role of writing and coding (which we understand broadly to include not only programming, but also markup) in the conduct and performance of scholarship in their contribution to the 2012 *Debates in the Digital Humanities* volume (Ramsay and Rockwell 2012, esp. the concluding paragraphs).

<sup>13</sup>As Matthew Kirschenbaum writes, “[c]omputers should . . . be understood as engines for creating powerful and persuasive models of the world around us. The world around us (and inside us) is something we in the humanities have been interested in for a very long time” (Kirschenbaum 2009, p. B10).

So where do the tools of Digital Humanities come from? Much as we argued earlier that scholarly inquiry is often most productive when it is motivated by an initial research question (even if that question later changes in response to serendipitous discoveries), our experience has been that innovative digital humanities research is most likely to emerge when the research question also precedes the identification and selection of tools. We fully recognize that gaining a personally useful understanding of the universe of possible digital tools and approaches is a difficult and time-consuming prospect. Moreover, selecting from this large set of available options is no less difficult, especially for newcomers to the field. It is this yawning domain of open opportunity that can feel daunting to humanists whose methodological boundaries once seemed so clearly set. Having a solid research question, as mentioned above, can be a touchstone, and DH mentors can serve as critical guidance, even lifelines, in this process. However, we also believe that the very practice of computer programming itself can empower researchers to take control over this decision space, and training humanists to construct narrowly focused, task-driven digital tools of their own provides a critical pedagogical opportunity.

Once we have identified a research question and an approach, if a tool exists that is truly appropriate for our purposes, it would be foolish not to consider using it, but if not, before we reject an otherwise exciting research question because nobody has built the tool for us, we should consider building it ourselves. As Joris van Zundert reminds us, at least with respect to some DH software:

[T]ool building is not a mere research-independent act to enable data processing. Rather, it is the act of modeling humanities data and heuristics as an intrinsic aspect of research. Tool and software development thus represent in part the capture and expression of interpretations about structure and properties of data, as well as interactions with that data. (van Zundert 2012, pp. 165–186)<sup>14</sup>

From this perspective, creating tools, and not merely using them, can function as an interpretive aspect of performing humanities research. One of the authors of this chapter (AL) has had the recent experience of teaching introductory courses in the Digital Humanities to graduate students in the Information Sciences as well as in the Humanities, and her observations confirm our intuition that the humanists tend to have an easier time forming a research question, while the information scientists tend to have an easier time becoming familiar with the tools. From a teacher's perspective, then, beginning with a question may work best when you are teaching humanists. Thinking about tools and their applications to the humanities, on the

---

<sup>14</sup>In another essay, Joris van Zundert and Ronald Haentjens Dekker explore in more detail the extent to which the creation of *software tools* (not digital editions or other end-result publications) can be considered humanities *scholarship*. Their analysis of the question distinguishes *enabling* and *performative* aspects of software, arguing that the latter embeds more scholarly assumptions and decisions, and may therefore be seen as having a more scholarly nature (van Zundert and Haentjens Dekker 2015).

other hand, can be a great way to get information scientists to understand the interpretive complexity of humanities data.<sup>15</sup>

We have used the term “tool” to refer to pre-existing software packages, but van Zundert’s observation encourages us to consider whether developing tools can itself constitute humanities research, and not just a way of enabling us to conduct research once the development has been completed. Andrea Laue, citing Karl Marx and Lewis Mumford, offers the following insight:

Marx writes that the origin or impetus of movement is the essential difference between a tool and a machine: with tools, movement originates in the laborer; with machines, movement issues from the mechanism itself (1867: 409). Working in an environment with machines requires education, that the body be trained to move along with the uniform and relentless motion of the mechanism (1867: 408). Conversely, the tool allows the laborer freedom of movement, an opportunity for motion independent of the mechanism. Working from Marx, Mumford associates tools with flexibility and machines with specialization. The essential difference, according to Mumford, is the degree of independence of operation. To summarize, a man works with a tool as an extension of his own body (and perhaps his own mind); in contrast, a machine employs a man as an extension of its own mechanism. (Laue 2004)<sup>16</sup>

Laue’s formulation invites us to consider software products developed by others that we may then employ not as tools, but as machines, insofar as they circumscribe the flexibility and independence of the researcher. This type of limitation is most obvious with respect to the actions we can perform, but, if we let them, black-box tools can also restrict the research questions we are able to ask to the domain of actions that others have chosen to facilitate for us.

Employing programming languages to build our own analytic tools, on the other hand, is consistent with Laue’s (and Marx’s and Mumford’s) definition of tools as enabling researchers to do whatever they want, instead of operating in an environment where the preexisting software (or perhaps its developer) has defined (and thus constrained) the terms of engagement. Moreover, knowing the process of producing computer software can also allow humanists to work more effectively within, and sometimes around, the initial user expectations for existing software packages—that is to say, it can teach humanists both to create their own tools and to

---

<sup>15</sup>Perhaps surprisingly, although a programming or other technical or technological background might be expected to (and often does) convey advantages in mastering new computational methods, our students with a strong computer science or information science background have sometimes also been the most resistant to learning new technologies, insisting on the greater ease of using the tools and methods they have already mastered even when those may not be as appropriate for their tasks as those we introduce in our courses. Assuming no difference in the quality of the end product, it makes sense in a production environment to get the job done as efficiently as possible, and avoiding a new learning curve is a sensible consideration. What surprises us is the invocation of that argument in a *classroom*, where, after all, learning to do something one does not already know how to do is largely the point of the educational enterprise.

<sup>16</sup>We are grateful to our colleague Aaron Brenner for bringing this citation to our attention.

hack others that are handed to them. We will continue to use the term *tool* in this chapter in its vernacular meaning, that is, to refer to both software products and computer programs that humanists develop themselves for their own research purposes, but the distinction between machines and tools underlies our advocacy for a digital competence for humanists that embraces programming, and not only the use of existing software packages.

## Languages and Humanities Research

Humanists who have never thought of learning a programming language might consider that acquiring a reading knowledge of research languages—that is, human languages used in scholarly publications in our fields even where those languages are not themselves our primary object of study—has long been a required component of professional training in many humanities disciplines. Nonetheless, there is at least one reason that humanists who accept the acquisition of professional reading competence in research languages as part of our basic training may have an instinctive perception of learning a programming language as something alien. Just as the use of the analytic essay has been embedded in our traditional training, humanists learn to read human languages because it has been part of our education and professional training, an education and training that typically will not have prepared us to know how to acquire technical skills. At their most basic level, though, research languages and programming languages are both skills humanists may need to acquire in order to conduct basic humanities research about something else. So how might humanists, deeply embedded in a training that is already replete with requirements and traditions, nonetheless learn to create and work with their own digital tools, that is, to write computer programs that enable new types of humanities research?

One superficially appealing but ultimately unsatisfactory answer to this question involves an appeal to collaboration, where the humanist formulates the research agenda and a programmer builds the tools. We are supportive of collaboration (see below), but it should not be our first recourse for at least three reasons. The first reason is that a more accurate term for the preceding description would be *compartmentalization*, rather than *collaboration*, and it entails a risk that the humanist will not learn much about how computation can serve humanistic research and the programmer will not learn much about how humanists formulate and think about research questions. The second reason is that, even under the best of circumstances (intelligent, intellectually curious, and professionally generous colleagues), the risk of missed opportunity is great because the humanist may not know what is possible computationally and the programmer may not understand what is interesting to a humanities scholar—that is, it may be that neither knows how to ask the questions that would bridge the divide. The third reason is that this sort of

compartmentalization reinforces the common and self-defeating assumption that computation is fundamentally external to humanistic inquiry and humanistic ways of thinking. All academic inquiry relies on tools and methods and methodologies, and part of our professional preparation (not only in our graduate-student days, but throughout our careers) involves learning to use those tools and methods and methodologies to conduct our research. We already learn to approach cultural objects from a variety of perspectives, some more intuitive and natural for us than others, and engaging with cultural objects computationally is ultimately just another perspective, and one that need not be regarded as so fundamentally alien to humanistic methods that we must subcontract others to perform it for us. There are, to be sure, times when we consult or collaborate with computational professionals, just as we consult and collaborate with non-computational colleagues in traditional but (for us) ancillary humanistic disciplines where the amount of knowledge required is greater than we can acquire ourselves. But Ted Underwood encourages us to consider the absurdity of outsourcing all of the computational work in humanities research by turning the tables: “Expecting computer scientists to do all the coding on a project can be like expecting English professors to do all the spelling” (Underwood 2014, n.p.).

Our argument is not that every humanist needs to learn to code,<sup>17</sup> but that humanities scholars whose research would benefit from the use of digital methods ought to be given access to more focused, domain-specific opportunities to acquire the knowledge and skills needed to conduct that research. And if the research agenda cannot be seen to completion entirely with existing software tools, humanities scholars should be encouraged to build their own. This does not mean that humanists should become experts in computer science, or even in computer programming, any more than art historians who photograph paintings for use in their research become scholars of photography or, for that matter, that humanists who write scholarly articles become scholars of writing. Moreover, proficiency in software creation enables more effective, authentic *collaborations* with software programmers who are true, fluent masters of their domain. Coding across the disciplines is similar to writing across the disciplines, and just as no researcher would say “I can’t write articles and books because I’m not a word person,” no researcher need say “I can’t write programs because I’m not a computer person.” But if you nonetheless think you are not a computer person because you have not had the opportunity to learn programming the way you have been learning writing all your life, how do you learn to code?

---

<sup>17</sup>We use “code” here to refer to computational processing in programming languages, but also to other computational interventions with cultural texts, such as the use of markup languages (sometimes distinguished as “encoding”, with “coding” reserved for programming). We distinguish this type of coding from other uses of software, such as a word processor for editing text or an image editor for editing graphics, by a conscious focus in coding on controlling the terms that will govern machine interaction with and operation on the object of study.

## Task-Driven Programming Pedagogy

The first step toward learning to code is to recognize that computer programming is not computer science; it is more like writing.<sup>18</sup> Everyone can learn to do it, and can be given the opportunity to learn to do it in ways that are appropriate for their disciplines. We offer humanists years of practice in learning to write; let us give them the chance also to learn to code. The second step is to recognize that learning a programming language is like learning a foreign language, except that it is much easier.<sup>19</sup> A medieval Islamic historian of our acquaintance who works with voluminous textual sources spent ten years learning Classical Arabic so that he could conduct his research. He spent just a few months learning to perform targeted tasks in the Python programming language that make him a more effective and successful historian. He is not a computer scientist, or even a specialist in Python; he is a humanities scholar who learned to use a tool to conduct his humanities research. The third step is to recognize that computer programming can be likened to cooking. Not everyone can pull off a multicourse meal with a large guest list where the hot food is still hot when served. But anyone can learn to cook, and being able to cook gives you dietary options you would not have otherwise. Similarly, basic mastery of a programming language that is plenty good enough to support real research is within the grasp of any humanist.

In order to teach humanities scholars how to learn to use a programming language, we look for inspiration to the transition in foreign language pedagogy from the grammar and translation model that predominated half a century ago to the oral proficiency model that is common today.<sup>20</sup> The grammar and translation approach is encyclopedic (it aims to teach the grammar of a language comprehensively and in a way that is organized by grammatical topic) and knowledge based: students learn a particular grammatical construction and they learn vocabulary, which they may then practice in written translation or composition or in conversation. In this knowledge-based model, the grammar and vocabulary of the language are the objects of study. The oral proficiency approach, on the other hand, is task-based: language learning is organized not by knowledge units (grammatical topics, vocabulary items), but by communicative tasks, such as conversing about your family, about your studies, about your hobbies, about current events, etc. Learners

---

<sup>18</sup>This paragraph is based on David Birnbaum's March 2014 address to the University of Pittsburgh Faculty Senate (Birnbaum 2014).

<sup>19</sup>The observation that the experience of learning programming languages is similar to that of learning human languages can be found in Janis Chinn's and Gabrielle Kirilloff's "Can humanities undergrads learn to code?" (Chinn and Kirilloff 2012). The authors were undergraduate humanities students and DH teaching assistants when they contributed this essay in January 2012 to *Techne*, the former blog site of the National Institute for Technology in Liberal Education [NITLE].

<sup>20</sup>Information about oral proficiency as a perspective on and methodology in second language acquisition and assessment is available at the University of Minnesota Center for Advanced Research on Language Acquisition (CARLA), <http://carla.umn.edu/assessment/MLPA/CoSA.html>

acquire vocabulary and grammar, of course, but the learning is organized around using the language in communicative contexts, rather than around knowing grammatical or lexical facts. For full near-native mastery of a language one eventually needs to acquire the grammatical knowledge and breadth of vocabulary of an educated native speaker, but one can communicate in many meaningful situations with less than full near-native mastery of a language. What is distinctive about oral-proficiency-oriented language learning, then, is the focus on being able to participate in communicative situations, rather than on learning facts about the language.

Textbooks for learning programming languages (including teach-yourself books) are often organized like reference grammars of human languages, with chapters like “data types” (treating, one after another, strings and integers and doubles and floats and lists and multidimensional arrays, etc.) or “control structures” (if-then-else, for, while, until, etc.). And when we look at the top hits of an online search for a combination of “syllabus” and “computer programming with ...” (filling in the name of a programming language), many courses are organized in the same encyclopedic way.<sup>21</sup> But, just as in the case of the movement away from a grammatical focus in language teaching to an oral-proficiency focus, there is another type of programming textbook and another type of course: the task-based, proficiency-oriented one. Books of this sort sometimes include the word “cookbook” in the title,<sup>22</sup> and what characterizes these books (and courses that follow the same model) is that they are organized not around learning, say, all of the numeric data types and then all of the non-numeric simple types and then all of the complex types, but around accomplishing specific coding tasks. Both types of textbooks include coding exercises, but the difference in perspective is crucial: in an encyclopedic textbook or course, the exercises exist in order to illustrate and practice specific features of the language, and the task may be contrived to provide an opportunity to practice those features. In a proficiency-oriented textbook or course, though, the exercises are—from the beginning—about learning to get things done in the language, and in a course for humanists, those tasks should be something that make sense in the context of humanistic inquiry. As Clifford Anderson writes about a proposed textbook for digital humanists, “the sample applications in this proposed textbook should center on narrative documents and, for the most part, avoid mathematical examples. In other words, if you plan to teach recursion, build an algorithm to validate palindromes rather than solve the Fibonacci sequence” (2014, n.p.).

---

<sup>21</sup>See, for example, the textbook to accompany Princeton University’s “Introduction to Programming in Python,” (Sedgewick et al. 2015). This material is described as “a textbook for a first course in computer science for the next generation of scientists and engineers” on the “booksite” found at <http://introcs.cs.princeton.edu/python/home/>. As may be appropriate for that audience, the approach to teaching Python is organized around computer science concepts and Python features, which it illustrates with examples and applications.

<sup>22</sup>See, for example, David Beazley and Brian Jones’s 2013 *Python Cookbook* (Beazley and Jones 2013).

Insofar as validating palindromes is also not a task commonly needed in actual DH research, we would go beyond Anderson and argue that teaching recursion should not be a course goal for which we then seek out a humanities-friendly task. The time to teach recursion is when you need it to perform a task that makes sense in the context of a real DH research question. For example, the time to teach recursion in an XSLT course might be the time students need to perform a task that would require them, in a procedural programming language, to modify the value of a variable.<sup>23</sup> That is, the task should not be invented to illustrate the method (in this case, recursion); rather, if the motivation is completing a natural humanities research task, it encourages not only learning the method, but also remembering why and where it is useful for digital humanists.

Encyclopedic textbooks play an important role in the study of human languages in situations where the audience is linguists for whom the language itself *is* the primary object of study, which is a different audience than people who want to learn to communicate in a language. Similarly, encyclopedic textbooks of programming languages make obvious good sense where the emphasis is as much on computation and on the programming language itself as it is on performing specific tasks. Furthermore, insofar as encyclopedic textbooks may include clear explanations, they may nonetheless have value as reference manuals even for proficiency-oriented learners. Our reservations are not about encyclopedic organization in general, but about letting it serve as the structural core of an introductory syllabus. In the computational methods courses we offer to undergraduate and graduate students, we emphasize that learning to write programs and use digital methods in order to conduct research in the humanities requires learning three broad, general things:

- **Algorithmic thinking.** Algorithmic thinking in a humanities context means that, for example, if you want to find out which characters speak in which act of a Shakespearean play, you can ask one question in a loop over the acts instead of five separate but almost identical questions, one about each act (Birnbaum 2015). And it also means that if you want to create a word-frequency list for a text, you need to recognize that task as consisting of small subtasks, such as breaking the text into words, identifying the distinct words, counting the occurrences of each distinct word, etc.<sup>24</sup> Digital humanists may someday need to know about big-O complexity and other foundations of algorithms as understood in computer science, but what humanists need to acquire immediately about algorithms is the ability to distinguish what the human does better than the computer from what the computer does better than the human, and the

---

<sup>23</sup>XSLT is a declarative language that does not permit the redefinition of a variable. The use of recursion as an alternative to iteration in XSLT is discussed and illustrated in Michael Kay's *XSLT 2.0 and XPath 2.0* (Kay 2008, pp. 992–1000).

<sup>24</sup>A humanist new to digital methods is likely to start by tokenizing on white space, whereupon the appalling initial output quickly reveals the need to decide how to handle punctuation, contractions, upper and lower case, etc. Overlooking those sorts of issues initially isn't an error; it's a natural part of a strategy that closes in on a solution by starting with the obvious and letting specific erroneous results guide the further development.



ability to break large, vague tasks into small, specific tasks. This requires learning to be explicit and precise in situations where humans may not otherwise have to be,<sup>25</sup> but it is not computer science.

- **Looking stuff up.** A proficiency-oriented approach to learning to code in a world where a lot of information is encyclopedic means learning to do what professional developers do, that is, to look stuff up, whether by treating the encyclopedic textbook as a reference manual, through a well-formulated Google query, or by engaging in targeted searching in on-line communities like StackOverflow. Digital technologies change too quickly for it to be practical to learn something comprehensively and then practice it for the rest of our careers. Learning to perform computational Digital Humanities in a way that lets us get our work done does not mean learning an entire programming language from a textbook. It means learning how to break down a big task into small ones (the algorithmic thinking part) and then learning how to look up how to do the small ones. This lesson is as valid for creating your own tools as it is for learning how to play in and around someone else's.
- **Incremental development and iteration.** When humanists write scholarly articles, some work from an outline, while others write an entire first draft and then go back and revise. But nobody writes a computer program as an entire first draft because when it breaks, as it inevitably will, finding the errors turns into a guessing game. Working from a skeleton with stubs (corresponding to the small tasks identified when thinking about the logic of the problem), and writing, testing, and debugging incrementally is a new paradigm for many humanists, but it is not hard to learn. If you do only one thing at a time and then test it, if it breaks your code, you know where the error lies and you can fix it. There is a common assumption among new learners that digital methods will work right away if you just “do it correctly.” This is rarely, if ever, the case because, no matter how much experience we have, we rarely, if ever, do it correctly the first time. Incremental development and iteration are key to the computational Digital Humanities.

Teaching humanists to code can be tackled in the same way we approach our own programming tasks: by focusing on algorithmic thinking, looking stuff up, and robust, iterative, incremental development. As instructors and DH mentors, we introduce our students to the methods and the sorts of tasks that digital humanists have to complete, we guide them to the reference resources that we find most useful in our own work, and we review their code and help them distinguish patterns from anti-patterns. What we do not do is first teach them abstractions like numeric datatypes or control structures and then give them non-contextualized (or even contextualized) exercises for practice. Our teaching is like our own on-going learning: it is organized around scaffolded goal-driven tasks that resemble what we use in our own work.

---

<sup>25</sup>For example, a recipe that tells a human to scramble eggs doesn't have to tell the human to break the shells first, but a computer program that reads a file from a disk may have to open the file explicitly first.

## Examples Matter

Embedding this heuristic understanding of what it takes to code in the context of research within the curriculum of the humanities itself is important for successful DH pedagogy. For best effect, therefore, identifying tasks that humanists might need to accomplish within the context of humanist inquiry is critical. As mentioned above, the Fibonacci sequence is a fascinating mathematical pattern, but is not commonly needed in the study of nineteenth-century photography, and confusion and alienation can ensue when we ask humanists to learn a digital method within a completely unfamiliar context. Examples of this struggle can easily be found in a number of the pre-existing educational supports available now to DH learners and DH mentors, including Lynda.com and YouTube videos. These resources are beneficial to DH pedagogy, but they have the disadvantage of being directed towards an audience that is largely assumed to be working within a business, rather than academic, context. This choice is understandable from the point of view of, say, Lynda.com's business model, but for digital humanists the use of business examples can provide an obstacle to effective learning, especially for newcomers to the field who are not accustomed to abstracting programming (or computer science) principles away from the particular situated examples being used in the demonstrations.

Training videos on data modeling provide an excellent case study of the impact that the choice of examples can have on learners. Lynda.com's "Relational Database Design with FileMaker Pro" teaches this approach to organizing digital information by using the common customer-orders-products paradigm, wherein there are a number of customers who can place a number of orders, each order being made up of a selection of products, any of which can be purchased in any quantity (Ippolite 2015). This is unquestionably a prototypical use case for relational databases, and it affords the trainer the opportunity to explain the vast majority of concepts entailed in designing such databases. Indeed, the course "Relational Database Fundamentals," which is slightly more tool-independent than the FileMaker course, also focuses on business needs, using examples from the fictitious Two Trees Olive Oil Company and the Explore California Tour Company. Business contexts such as these force digital humanists to do quite a bit of translation work; after all, as a rule, humanists are not attempting to model customers, orders, or products, and may even be as unfamiliar with this use case as they are with the concept of one-to-many relationships.

YouTube's tutorial videos use similar domains to explicate data modeling. A number of approachable videos have been posted, most of which use business case studies as examples. Gina Baldazzi has contributed an "Entity Relationship Diagram (ERD) Training Video," which has 362,000+ views, and uses a university registrar's database as its example (Baldazzi 2013). Indeed, registration is also a common theme for database concept videos, perhaps because the one-to-many relationship between a student and classes is crystal clear (Glasser 2011). But the most memorable example of these eminently clear, but non-humanities-research-related, contexts for relational database modeling problems belongs to Mr. B's Code

Academy lectures on normal forms ([Mr. B's Code Academy] 2012). His on-the-fly use of Excel to explain the problems at hand is helpful for new learners in many ways, but his choice of a pizza delivery business as his example has led students in the humanities to ask us, "What are the 'pizza toppings' again for my data model of German Conceptual Art in the 1960s?"<sup>26</sup>

In the Lynda.com tutorial first mentioned, "Relational Database Design with FileMaker Pro," the trainer does, only briefly, switch to a slightly more humanist-friendly example, that of modeling the appearances of actors in movies. This example is well suited to discussing the perils of instantiating many-to-many relationships, and affords the humanist a more welcoming entrée into these principles (even if, in this case, a humanist may take issue with the simplicity of the example). At this point in the trajectory of the practice of the Digital Humanities, more pedagogical tools could certainly be created that treat authentic humanities examples at their true level of complexity, allowing newcomers to the field not to stumble over the intricacies of data taken from an unfamiliar domain. Of course, each domain within the humanities might need its own examples—an art historian does use dramatically different data from a scholar of Slavic languages—but this might perhaps be addressed through "user active" tutorials.<sup>27</sup> The best of these new tools could be driven by user-chosen tasks, and would therefore naturally be domain-specific.

A preference for examples drawn from DH research raises another issue that will be familiar from foreign-language pedagogy: the challenge of using authentic materials in the elementary classroom. Authentic examples of language as used by real native speakers (that is, not made up by teachers to illustrate grammar or vocabulary) may contain much that is new to the beginning learner, and the same is true of real DH tasks. How can learners engage with authentic materials without becoming overwhelmed? In foreign-language pedagogy it has become a cliché to "simplify the task, rather than the text," recognizing that learners can perform real-world tasks with authentic materials without understanding every word.<sup>28</sup> In both language learning and DH contexts, this simplification is often implemented through scaffolding, which is another way of describing the process of breaking a large, vague, complex task into small, discrete tasks that can be addressed individually.

---

<sup>26</sup>The pizza-topping model was used explicitly in a DH context by the Text Encoding Initiative in their *TEI Pizza Chef* (Text Encoding Initiative 1999).

<sup>27</sup>Dan Colman, of the website *Open Culture*, defines *user active* tutorials as those where as "users can...design projects of their own choosing" (2016, n.p.).

<sup>28</sup>For example, beginning language students may not be able to understand every word of the listings of film screenings for a foreign city, but those students can typically read the same listings that native speakers read and identify the name of the cinema and the screening times. Beginning language students cannot read the web site at a foreign university as easily as the one at their own, but in the case of many foreign languages they can use international vocabulary to identify courses in which they might enroll without previously having learned the names of those subjects in the new language.

## Conclusions

Our advocacy for a humanities-oriented programming pedagogy for the computational Digital Humanities emerges first from the many ways in which the computational Digital Humanities resembles non-computational research methods with which humanists are already familiar, and we propose exploiting that familiarity to contextualize the new learning. In particular, some of our observations are inspired by the lessons humanists have already learned about foreign-language pedagogy, and specifically about proficiency-based learning and the use of authentic materials even at a beginner level. Furthermore, to avoid the constraints and risks that come with allowing the available tools—those produced by humanists and non-humanists alike—to dictate the entire scholarly agenda, and because the act of programming can be part of conducting research (and not just preparation for conducting research), we consider it imperative to demystify programming and empower humanists to write programs that will do exactly what they need, whether at the basic or advanced level. Specifically:

*The tool we know and the tool we are learning:* Conducting digital research in the humanities has much in common with conducting non-digital research in the humanities. Instruction (by teachers and mentors) and acquisition (by learners) of digital methods in the humanities can be facilitated by distinguishing the genuinely new from that which may be unfamiliar, but which, upon closer inspection and consideration, turns out ultimately to resemble non-digital methods of humanistic inquiry.

*Yes, you can build your own tools:* Conducting digital research in the humanities requires learning to use digital tools. Researchers should start by identifying what they want to accomplish, and if the tools they need do not exist, they should be given the skills and the opportunity to learn to make them. In many cases, coding can be an actual part of hermeneutic practice.

*Task-driven programming pedagogy:* Learning a programming language may be a new experience for humanists, but learning something new in order to conduct our research is familiar. Learning a programming language can be made more accessible through task-driven instruction, with benefits comparable to those introduced by proficiency-oriented curricula in foreign-language pedagogy.

*Examples matter:* Textbooks and video learning resources for digital tools and methods are often based on examples drawn from the business world. Much as today's foreign-language pedagogy makes effective use of authentic texts even at a beginning level, programming pedagogy for humanists can instead draw on genuine DH research needs as a way of contextualizing the learning.

## References

- Anderson, C. B. (2014). On teaching XQuery to digital humanists. In *Proceedings of Balisage: The Markup Conference 2014, Washington, D.C. August 5–8, 2014*, 13. doi:[10.4242/BalisageVol13.Anderson01](https://doi.org/10.4242/BalisageVol13.Anderson01)
- Baldazzi, G. (2013, January 29). *Entity relationship diagram (ERD) training video* [Video File]. Retrieved from <https://youtu.be/fQ-bRllhXc>

- Beazley, D., & Jones, B. K. (2013) *Python cookbook* [ebook]. Sebastopol, CA: O'Reilly. Retrieved from <http://chimera.labs.oreilly.com/books/1230000000393/index.html>
- Birnbaum, D. J. (2014). *Faculty Senate plenary address*. Presented at the Spring 2014 Meeting of the University of Pittsburgh Faculty Senate, Pittsburgh, Pennsylvania. Retrieved from [http://www.obdurodon.org/slides/2014-03-19\\_senate-plenary.pdf](http://www.obdurodon.org/slides/2014-03-19_senate-plenary.pdf)
- Birnbaum, D. J. (2015, August 23). *Thinking in algorithms*. Retrieved from <http://dh.obdurodon.org/algorithms.xhtml>
- Chinn, J., & Kirilloff, G. (2012). *Can humanities undergrads learn to code?* Retrieved from <http://dh.obdurodon.org/nitle.xhtml>
- Colman, D. (2016, March 7). *Learn how to code for free: A DIY guide for learning HTML, Python, Javascript & More* [Web log post]. Retrieved from <http://www.openculture.com/2016/03/learn-how-to-code-for-free-a-diy-guide-for-learning-html-python-javascript-more.html>
- Glasser, M. [Prescott Computer Guy]. (2011, September 30). *Relational database concepts* [Video File]. Retrieved from <https://youtu.be/NvrpuBAMddw>
- Ippolite, C. (2015, May 12). *Relational database design with FileMaker Pro* [Video File]. Retrieved from <http://www.lynda.com/FileMaker-Pro-10-tutorials/Relational-Database-Design-with-File-Maker-Pro/83839-2.html>
- Kay, Michael. (2008). *XSLT 2.0 and XPath 2.0* (2nd ed.). Indianapolis: Wiley.
- Kirschenbaum, M. (2009, January 23). Hello worlds. *The Chronicle Review*, 55(20), B10. Retrieved from <http://chronicle.com/article/Hello-Worlds/5476>
- Langmead, A. (2015, August 13). *Syllabus for the Ph.D. seminar, The Digital and the Humanities. Fall Term 2015* [Web log post]. Retrieved from <http://constellations.pitt.edu/entry/syllabus-phd-seminar-digital-and-humanities-fall-term-2015>
- Langmead, A. (2016, July 25). *Summer 2016 syllabus: "Digital Humanities," MLIS Program, University of Pittsburgh* [Web log post]. Retrieved from <http://constellations.pitt.edu/entry/summer-2016-syllabus-digital-humanities-mlis-program-university-pittsburgh>
- Laue, A. (2004). How the computer works. In S. Schreibman, R. Siemens, & J. Unsworth (Eds.), *A companion to digital humanities* (Chapter 13). Oxford: Blackwell. Retrieved from <http://www.digitalhumanities.org/companion/>
- Mahony, S., & Pierazzo, E. (2012). Teaching skills or teaching methodology? In B. D. Hirsch (Ed.), *Digital humanities pedagogy: Practices, principles, and politics* (pp. 215–25). Cambridge, UK: Open Book Publishers. doi:10.11647/OBP.0024
- [Mr. B's Code Academy]. (2012, November 25). *Normalisation 3NF: Third normal form example* [Video File]. Retrieved from <https://youtu.be/c7DXeY3aIJw>
- Owens, T. (2014, August 22). *Where to start? On research questions in the digital humanities* [Web log post]. Retrieved from <http://www.trevorowens.org/2014/08/where-to-start-on-research-questions-in-the-digital-humanities/>
- Perry, D. (2012). The digital humanities or a digital humanism. In M. K. Gold & L. F. Klein (Eds.) *Debates in the digital humanities*. Minneapolis: University of Minnesota Press. Retrieved from <http://dhdebates.gc.cuny.edu/debates/text/24>
- Ramsay, S. (2010). *The hermeneutics of screwing around*. Retrieved from <https://web.archive.org/web/20101105171751/http://www.playingwithhistory.com/wp-content/uploads/2010/04/hermeneutics.pdf>
- Ramsay, S. (2012). Programming with humanists: Reflections on raising an army of hacker-scholars in the digital humanities. In B. D. Hirsch (Ed.), *Digital humanities pedagogy: Practices, principles, and politics* (pp. 227–39). Cambridge, UK: Open Book Publishers. doi:10.11647/OBP.0024
- Ramsay, S., & Rockwell, G. (2012). Developing things: Notes toward an epistemology of building in the digital humanities. In M. K. Gold & L. F. Klein (Eds.), *Debates in the digital humanities*. Minneapolis: University of Minnesota Press. Retrieved from <http://dhdebates.gc.cuny.edu/debates/part/3>

- Sayers, J. (2012). Tinker-centric pedagogy in literature and language classrooms. In L. McGrath (Ed.), *Collaborative approaches to the digital in English Studies* (pp. 279–300). Retrieved from [http://ccdigitalpress.org/cad/Ch10\\_Sayers.pdf](http://ccdigitalpress.org/cad/Ch10_Sayers.pdf)
- Scheinfeldt, T. [@foundhistory]. (2014, August 8). I've been asked to compile a list of top 10 pieces of advice for new Dh'ers for a group of public humanities fellows. *Suggestions?* [Tweet]. Retrieved from <https://twitter.com/foundhistory/status/497763193612410880>
- Scheinfeldt, T. (2012). Where's the beef? Does digital humanities have to answer questions? In M. K. Gold & L. F. Klein (Eds.), *Debates in the digital humanities*. Minneapolis: University of Minnesota Press. Retrieved from <http://dhdebates.gc.cuny.edu/debates/text/18>
- Sedgewick, R., Wayne, K., & Dondero, R. (2015). *Introduction to programming in Python: An interdisciplinary approach*. New York: Addison-Wesley. Retrieved from <http://introc.cs.princeton.edu/python/home/>
- Spiro, L. (2011, October 14). *Getting started in the digital humanities* [Web log post]. Retrieved from <https://digitalscholarship.wordpress.com/2011/10/14/getting-started-in-the-digital-humanities/>
- Spiro, L. (2012). Opening up digital humanities education. In Brett D. Hirsch (Ed.), *Digital humanities pedagogy: Practices, principles, and politics* (pp. 331–64). Cambridge, UK: Open Book Publishers. doi:10.11647/OBP.0024
- Text Encoding Initiative. (1999, October 8). *TEI Pizza Chef*. Retrieved from <http://www.tei-c.org/Vault/P4/pizza.html>
- Underwood, T. (2014, March 18). *How much DH can we fit in a literature department?* [Web log post]. Retrieved from <https://tedunderwood.com/category/dh-as-a-social-phenomenon/>
- van Zundert, J. J. (2012). If you build it, will we come? Large scale digital infrastructures as a dead end for digital humanities. *Historical Social Research—Historische Sozialforschung*, 37(3), 165–186. Retrieved from <http://nbn-resolving.de/urn:nbn:de:0168-ssoar-378903>
- van Zundert, J. J., & Haentjens Dekker, R. (2015, October 29). *Code, scholarship, and criticism: When is coding scholarship and when is it not?* Paper Presented at the Digital Humanities 2015: Global Digital Humanities Conference, Sydney, Australia, June 29–July 3, 2015. Retrieved from <http://jorisvanzundert.net/wp-content/uploads/2016/02/CodeScholarshipCriticism.pdf>
- Wing, J. (2006, March). Computational thinking. *Communications of the ACM*, 49(3), 33–35. Retrieved from <https://www.cs.cmu.edu/~15110-s13/Wing06-ct.pdf>

## Author Biographies

**David J. Birnbaum** is Professor and Chair of the Department of Slavic Languages and Literatures at the University of Pittsburgh. He has been involved in the study of electronic text technology since the mid-1980s, has delivered presentations at a variety of electronic text technology conferences, and has served on the board of the Association for Computers and the Humanities, the editorial board of *Markup Languages: Theory and Practice*, and the Text Encoding Initiative Technical Council. Much of his electronic text work intersects with his research in medieval Slavic manuscript studies, but he also often writes about issues in the philosophy of markup. Since 2011 he has taught an undergraduate honors course entitled “Computational Methods in the Humanities”, cross-listed in eight departments, where students learn, hands-on, to use XML and web technologies to support research in the humanities.

**Alison Langmead** currently holds a joint faculty appointment at the University of Pittsburgh between the Dietrich School of Arts and Sciences (DSAS) and the School of Information Sciences (SIS). At DSAS, Langmead serves as the Director of the Visual Media Workshop (VMW), a digital humanities lab focused on the investigation of material and visual culture—historical or contemporary—in an environment that encourages technological experimentation. At SIS, Langmead teaches courses on digital preservation and the digital humanities. In her research, she

designs and produces digital humanities projects that investigate visibility and materiality as a multivalent, interactive process. Langmead is the Principal Contact for the DHRX: Digital Humanities at Pitt faculty research initiative, which represents a transdisciplinary network of scholars at the University of Pittsburgh who use digital methods to study the ways in which humans interact with their environments, whether social or cultural, natural or human-created. Langmead holds a Ph.D. in medieval architectural history from Columbia University as well as an MLIS from the University of California, Los Angeles.