Scholarship in interaction: case studies at the intersection of codework and textual scholarship

Zundert, J.J. van

Chapter 9

Conclusion

The overarching claim I make with this dissertation is that there is a knowledge space between software engineering and textual scholarship that is undertheorized, academically underdeveloped, and intellectually undervalued. The following summary of the most important conclusions of the prior chapters will serve to substantiate this claim. The primary subject of this work as a whole is the interaction between software engineering and textual scholarship, as explained in the introduction (chapter 1). The second chapter, delving into hermeneutics, provided a theoretical background, while chapters three and four are essentially case studies of this interaction, pertaining to the creation of digital scholarly editions and the creation of related tools in the context of the Huygens Institute. The following chapters reflect on this and similar work and are more concerned with questioning academic authority, critique, and evaluation in the context of software development in textual scholarship. The research underpinning this dissertation is strongly interdisciplinary work and it will probably be clear by now that the main argument and conclusions of each chapter do not fall neatly inside this or that disciplinary category. For clarity however, I have tried to gather the conclusions of the various chapters under the headings of three disciplinary categories or perspectives.

9.1 The Textual Scholarship Perspective

Read as a theoretical and practical study of the problems that exist with the currently prevalent way of expressing scholarly digital editions in textual

scholarship, my work provides a rationale for a practical solution. Chapter 3 and 4 theorize from experience the limitations of markup models. These chapters add to the intellectual work that arose in the context of "the problem of overlap" in markup languages. On the practical level that problem may seem trivial to software engineers and digital scholars – and they might interpret it as a mere conflation of text structure and the structure of XML. However, the application of XML as a form of markup for text structure in a purely hierarchical fashion, as it was adopted by the TEI based on a post hoc rationale by DeRose et al. (1990), caused a lively intellectual debate about text, structure, meaning, and digital affordances for their expression (e.g. Buzzetti 2002; DeRose 2004; McGann 2004 etc.). On the basis of this literature and my own work I conclude that hierarchical markup is an inadequate fit to capture and express the multidimensional nature of text. In chapters 5 and 7 I propose graph models as a technical solution that exceeds any hierarchical markup model in expressiveness. These chapters also report on some of the practical aspects of my work that contributed to the development of the graph model as a more versatile descriptive model for form, structure, meaning, and interpretation of text and documents. It is exceedingly gratifying to see that the hard and experimental work by me and others with regard to graph technologies is slowly finding a toehold in textual scholarship, as indicated by various publications (e.g. Haentjens Dekker and Birnbaum 2017; Andrews et al. 2018; Neill and Kuczera 2019; Efer 2017) and conferences such as the annual Graphentechnologien/Graph Technologies conferences since 2018, initiated by the Akademie der Wissenschaften Mainz, and the 2019 Workshop on Scholarly Digital Editions, Graph Data-Models and Semantic Web Technologies at the Université de Lausanne).

9.2 Science and Technology Studies Perspective

This dissertation is also the result of Science and Technology Studies labor. The STS perspective calls attention foremost to the coding work that is related to graph models and especially the tools producing the graphs, such as CollateX (chapter 5 and others) and Stemmaweb (chapter 7). The software engineering work involved with these graph-oriented approaches pro-

vides a more precise example of how actual software code impacts textual scholarship. A paramount difference exists between the "xmlificaton" and digitization work in textual scholarship and such codework approaches. I found (cf. chapters 3 and 4) that the use of markup results only in a shallow form of remediation which accepts digital technology predominantly as a way to reproduce a digital metaphor of the book on a computer screen. However, the algorithmic work involved with CollateX and Stemmaweb signals a partial shift of agency and control related to scholarly editing. Part of textual scholarship's agency and authority shifts, in more covert than overt ways, first from scholarly editor towards engineer and second, most interestingly I think, from engineer to code (see for this chapter 5, but foremost the section "The author and the engineer" in chapter 6). This shift of agency and control is not as absolute as the title of Lev Manovich's work *Software Takes Command* might suggest. Manovich (2013), like Berry (2014), Coyne (1995), Morozov (2013), and others take rather grand perspectives, arguing how software and digital information may affect societies as a whole. This sometimes truly suggests images of software as an omnipresent "invisible hand". My STS work examines the shaping power of software code at a decidedly smaller scale, and finds that code does not truly take over. Rather, some components of scholarship are trading places. As tedious and error-prone tasks, their control and the associated agency are delegated from the scholar, via the software engineer, to software code.

It should be stressed that this process of delegation progresses in a rather unobserved manner. I witnessed that there is little explicit methodological reflection built into either the ad hoc processes or the formal software engineering methodology that governs projects where software engineers and textual scholars collaborate. Slightly covertly therefore, and unintentionally, this process of delegation passes over a number of important theoretical, methodological and epistemological questions. A number of these matters were investigated in chapters 5 and 6. The main conclusion I draw from this work is that digital textual scholarship and the associated coding work is in dire need of a framework for evaluation. Although it has earned me several remarks that Critical Code Studies, Software Studies, and platforms such as Cultural Analytics[1] long since provide this kind of evaluation, I stand firmly by my

[1] https://culturalanalytics.org/

contention: code peer review and academic acknowledgement of code work are tremendously underdeveloped in textual scholarship and, by extension, in the humanities. None of these domains and platforms provide precise, code level, peer review and evaluation. Their engagement with the scholarly work is at a level of method that is detached from actual software code and considers only the analytic results obtained through code. Whether that code functioned correctly and in accordance with theory and hypotheses is never evaluated.

The conclusion that the evaluation of coding in textual scholarship is inadequate is important, as it pertains to pivotal concerns about who produces knowledge and how we evaluate that. Currently my case studies in the context of the Huygens Institute, and partly beyond, suggest that most textual scholars conveniently assume that authority and control over scholarly work still resides firmly with the scholars – and literature suggests this holds for a wider scope than just the Huygens Institute (cf. Robinson 2016; Bordalejo 2018). But as I argue in chapter 6, scholarship expressed in code results in the same appropriation of revisionary authorship as scholarly editing does. Software engineers and code both produce knowledge, and as such they must not be denied a proper form of evaluation and crediting in any academic system. Both scholars and software engineers should take full responsibility to find and establish ways that acknowledge the scholarly contribution that code and programmers make. A formal means of acknowledgment must allow software engineers – no matter whether they are scholars, software engineers, or both – to be credited and, moreover, accountable for their work.

Another effect that my STS approach to the problem of interaction between software engineering and scholarship uncovered is the process of paradigmatic regression: affordances of unknown or new methodology may become hidden when they are narrowly used to express concepts and terminology of a prior and more familiar paradigm. The digital interfaces, especially graphical interfaces, that stand between computational model and scholar are strange beasts that play a crucial role in this process. They open up as much as they hide the computational model (cf. chapter 3 and chapter 7). At the same time that they make it useable, they distort the computational model in ways that provide endless opportunities for misinterpretation, misunderstanding, and uninformed or plain incorrect

use of computational tools. Because they are made by humans they are far from neutral, but they still carry something of a halo of perfect impartiality because this is, however incorrectly in itself, associated with digital computing. Computational models are not neutral and their makers are not perfectly disinterested. Interfaces are loaded and situated. I have found little to no evidence that collaboration between textual scholars and software engineers specifically on graphical interfaces results in any significant methodological gain or diffusion in either direction (cf. chapter 3 and especially chapter 4). If anything, graphical interfaces act more like fences that keep epistemologies well apart and inhibit methodological knowledge exchange more than they enable it. I think this "interface effect" (Galloway 2012) has been especially pernicious in the case of the scholarly digital edition. The fully representational paradigm that digital textual scholarship writ large has preferred for digital scholarly editing has been reaffirmed consistently by skeuomorphic visualization – that is, making the graphical interface look as closely as possible to a physical book, if possible down to mimicking cover, paper, and ink. Arguably the benefits of the tremendous increase in scale of access to (digitized) sources and knowledge do warrant the equally tremendous economical, technical, and personnel resources that representational digital scholarly editions require, even if institutions and scholars still struggle to establish sensible strategies for the sustainability of scholarly digital objects and infrastructure. However, the strong regression towards the representational paradigm – powered and reaffirmed continuously by graphical interfaces – has all but eradicated the appreciation for actual methodological affordances that might be found in a genuine interaction between computing and textual scholarship. To many computing and textual scholarship may seem like irreconcilable foes. Based on my findings however, my contention is that they are the very two components that together underpin a paramount task of methodological scholarship for the next decade: the establishment of a true computational hermeneutics.

9.3 The Humanities Computing Perspective

Obviously this dissertation also results from interdisciplinary work in humanities computing. However, increasingly I have difficulty with the term "interdisciplinary". Like "interface", it implies a separation as much as it suggests some cross-discipline dynamic. I no longer have a use for this separation. In fact, I find it unhelpful and bothersome. The neat disciplinary boundaries that we ourselves maintain place constraints and limits on the use and application of our collective but distributed knowledge. I am a professionally trained programmer, an academically trained textual scholar, and a novice science and technology studies researcher. It is my own hybrid nature rather than the interdisciplinary nature of my work that allowed me to assemble the theoretical knowledge (chapters 2, 4, and 6), practical work (chapters 3 and 5), and experience (chapter 8) that resulted in an understanding of the interaction between software engineering and textual scholarship, informed by multiple perspectives. This in turn allows me forcefully to conclude that there is indeed a terrifying gap between software engineering and textual scholarship. This is the intellectual space I aimed at in the first paragraph of this conclusion – an intellectual space that is undertheorized, underdeveloped, and undervalued. It is the knowledge space of computational hermeneutics.

Mostly by way of chapters 2 and 8 I have argued in more detail how central hermeneutics is to textual scholarship. An understanding and practice of computational hermeneutics becomes urgent in a society and culture increasingly producing its textual legacy through digital text and its digital-native relative: code. It is therefore both surprising and disappointing how little interest there is either in computer science, software engineering, or textual scholarship for the computability of hermeneutics. Scholars have called attention to digital hermeneutics (e.g. Capurro 2010; Frabetti 2012; Meister 1995; Thaller 2018), but a rigorous theoretical and applied program of scientific investigation involving the three related disciplines – computer science (including software engineering), textual scholarship, and science and technology studies – remains wanting.

I have found four main factors that inhibit the various disciplines from pro-

gressing productively into this space. The first is the fully representational paradigm and associated technologies of markup that digital textual scholarship has championed as its prime philosophy. A computational hermeneutics for textual scholarship at the very least requires scholars to appreciate the differences in nature and behavior of digital text, its relation to code, and especially the precise performative nature of code with its dual guise of text and executable. However, textual scholarship appears singularly interested in bookish screen essentialism. The fully representational paradigm gives rise to the second main factor: an exaggerated attention to (XML) encoding of texts in what little digital humanities education there is in textual scholarship. Although ever more digital humanities minors and masters have sprung into existence, attention to code and coding is – especially in the Dutch situation – negligible. But code literacy is an essential skill for the next generation of textual scholars (cf. Vee 2013).

Third, computer science (and especially its practical counterpart, industry level software engineering) and textual scholarship sustain a rhetoric that causes a myopic understanding of computing and its uses for textual scholarship. Scholars predominantly sustain a rhetoric of reductiveness about computational methods. While it is true that machine learning, natural language processing, and stylometrics, for instance, are currently rather reductive techniques, this does not mean at all that code must ever be reductive. However, merely pointing this out continually (Johanna Drucker is a well known proponent of this type of rhetoric) is only of limitedly help. Technologists aggravate this situation by emphasizing a rhetoric of speed and scale, by mystifying code (which is actually a rather straightforward semiotics to describe objects and actions), and by sustaining a lack of interest in the potential hermeneutic forms of coding. Attention for possible hermeneutic forms of computer code in scholarship is equally minimal, with the odd notable exception, such as Manfred Thaller (2018).

Lastly, institutional knee-jerk reactions hardly help. It is still too difficult to amass academic credit for digital objects and coding work. This problem is annoyingly persistent. Added to this is an institutional preference for outdated organizational philosophies and structures. Nowhere do I see this dividing force more up close than in the Huygens Institute, now part of the Amsterdam Humanities Cluster. Continuous reorganizations

have reaffirmed the boundaries between scholarship and research on the one hand and software engineering on the other. The organizational charts of the successive Constantijn Huygens Institute, Huygens Institute, Huygens Institute for the History of the Netherlands, and the Humanities Cluster have consistently reaffirmed through old-fashioned organizational boundaries who is allowed to produce – or rather who will be credited with having produced – scientific knowledge, and who is not. These boundaries have consistently separated scientific programmers from their scholarly colleagues. Rather than being embedded in research departments they have been increasingly put at an ever-increasing distance from researchers. The separation resulting from such a professionalization of IT services also results in token reaffirmation of these boundaries through ticket systems, administrative resource management requirements, product managers, and so forth. Rather than providing for interdisciplinary work and hybridization of knowledge and skills, this organizational inertia obviously does little to contribute to methodological innovation.

However, I want to close this dissertation on a more upbeat tone. I repeat that I am a professionally trained programmer, an academically trained textual scholar, and a novice science and technology studies researcher. Through my studies and experiences at the Huygens Institute and beyond, over more than fifteen years, I have learned an enormous amount about the interaction between software engineers and textual scholars. The same context, problematic as it sometimes may be, challenges me to keep learning every day, which is a blessing that I am tremendously thankful for. STS method has taught me how to reflect analytically on this ongoing work. These reflections have enabled me to venture beyond my initial personal preoccupations, to look harder, and to question more earnestly. It has been the multidisciplinary context that brought the contours of a "computational hermeneutic void" to my attention. My hope is that this dissertation contributes to an acknowledgement of the intellectual space that exists between computer science and textual scholarship. My expectation is that one day textual scholarship, computer science, and software engineering will truly meet each other in this space. My wish is to explore that space and to understand this beast called "computational hermeneutics" while walking with it.

–Joris van Zundert
Utrecht, 9 September 2019