**ROCKWELL — Talking**

<2 figures>

**Talking about Programming in the Digital Humanities**

Rockwell, G. and Sinclair, S.

In the first months of the HUMANIST discussion list a discussion erupted around the value of programming languages in humanities computing courses. Do computing humanists need to know how to program, or is learning expert use of applications enough? What is the point of learning to program? (McCarthy, 1987). This discussion can be traced back at least as far as the 1986 ACH/Sloan Workshop on Teaching ‘Computers and the Humanities’ Courses, where participants discussed ‘most vigorously of all whether programming should be taught, or only package programs’ (Sperberg-McQueen 1986). The issue has really never gone away as it is about disciplinary formation or ‘what it is we want our students to learn, the nature of the world into which we are sending them, and the relationship both of technology and (more fundamentally) the algorithmic approach to problem-solving’ (McCarty, 1987, 1–2). Based on a reading and analysis of a substantial collection of historical documents (journal articles, association newsletters, books, Listservs, etc.) in this paper we will examine how the humanities computing community discussed programming and how that discussion reflects changing views of the field over time. Specifically we will

• Outline a three-part history of programming in the digital humanities starting with a concording phase when programming was not important to using computers.

• Discuss the turn toward teaching programming as a way of teaching computational thinking.

• Look at how the emergence of the Web changed the discussion. We will argue that as the Web became important, scripting skills suitable for building websites replaced older programming languages and it became possible to imagine the digital humanities being defined by the ability to program.

• Conclude with some reflections on how the case for programming has been made more recently.

**Humanities Computing as Concording**

I’ve tried to stay free of programming. I’m perfectly innocent of any knowledge of any programming language and I feel I must remain that way if I am going to continue to function as a scholar in literature; my heartfelt advice to any of you younger people embarking on this whole venture is to do the same—to ask programmers to do for you what they know how to do and what would be costly and painful for you to learn—to learn to talk their language but not to get involved in programming research or machine research. I know the best of you will not take this advice and will, therefore, break new barriers and so on, but I persist in offering it. (Parrish, 1969, 24–25)

Programming in the early concording years of computing in the humanities was something often left to specialists. Stephen Parrish, who was general editor of the Cornell Concordance Series, saw himself as a ‘scholar in English literature who drifted laterally into the making of concordances’ (1969, 16). He stayed away from programming as a distraction from scholarship. This meant that concording ‘required an organized, teamwork approach, characterized by a substantial budget and a university computing centre’ (McCarty, 1993, 52–53). Parrish is not the only one to worry about programming distracting scholars, and in the presentation we will discuss other examples as a way of teasing out how programming was not considered of scholarly value.

**Programming and Reasoning**

By the 1980s, things had changed and programming had become important, as can be seen in the way programming was included in a majority of courses. Surveys like Joseph Rudman’s ‘Teaching Computers and the Humanities Courses: A Survey’ (1987) report 175 courses teaching a programming language while 131 were ‘Applications Only’. In that survey, the most popular languages taught were BASIC (in 75 courses reporting), Pascal (30 courses), Prolog (15 courses), Lisp (13 courses), and SNOBOL (10 courses). What is impressive is the number of responses that indicated they had some sort of course for humanities students. We often assume that computing in the humanities is a recent, post-advent-of-the-Web thing, but given the number of courses, the discussions on HUMANIST, and articles in journals like *Computing and the Humanities*,it is clear that programming was becoming an acceptable activity, especially for those in support staff positions.

Despite the popularity of programming languages like BASIC and Pascal for courses, the languages about which humanists wrote in the 1980s tended to be languages like SNOBOL and Icon—all languages that were good for string (text) manipulation, as can be seen in the attention they received in books for teaching programming to humanists like John Abercrombie’s *Computer Programs for Literary Analysis* (1984)*,* Susan Hockey’s *SNOBOL Programming for the Humanities* (1985)*,* and Alan Corré’s *Icon Programming for Humanists* (1990)*.* Then there are the discussions on HUMANIST and reviews of languages like Mark Olsen’s self-explanatory ‘Beyond SNOBOL: The Icon Programming Language’.

As an aside, one could argue that SGML (Standard Generalized Markup Language) and later XML (Extensible Markup Language) are also forms of programming—meta-languages with which one can create descriptive languages with which to rigorously describe texts for scholarly electronic editions. These were popular in humanities computing, especially after the Text Encoding Initiative began to provide guidelines for the encoding of texts in the late 1980s. How many humanists were first introduced to computing in the humanities when asked to develop a DTD (Document Type Definition) and encode a text?

It is interesting that many of the discussions about programming in the 1980s and early 1990s circle around the teaching of it and that this issue is reported as contentious. No one believed that the ability to program was essential for a computing humanist, in part because so many couldn’t, but proponents of programming argued for it to be taught as way of teaching computational reasoning or problem solving. They also argued that there were social and ethical issues that could be understood through learning computing. You learned to program so as to understand what the computer could do or to be able to talk to programmers. In the presentation we will look at summative discussions of the issue like McCarty’s (1987) and a lovely balanced essay by Nancy Ide, ‘Computers and the Humanities Courses: Philosophical Bases and Approach’ that came out of the 1986 ACH/Sloan Workshop. We will also speculate as to why there was this shift towards including programming and arguing for its importance. Whatever the reasons, the discourse had changed and computing humanists were beginning to take an interest in programming and teaching it.

**Programming the Web**

The emergence of the Web changed the languages that humanists were likely to use for programming and, we will argue, made it possible to make programming a defining skill for digital humanists. Moreover, learning to program for the Web provided a more appealing and transferable skillset; web and application development became normalized and desirable. Because the Web provided such a convenient way to show and distribute digital research, it changed which programming languages received attention or were taught. We can see the shift in these histograms of word frequencies in the HUMANIST Archives. The string processing languages popular in the late 1980s taper off and are replaced with languages like PHP with which one can build dynamic websites.





Figure 1. Two graphs showing the frequency of programming languages in the annual archives of the HUMANIST Discussion Group Listserv, produced with Voyant Tools.

Now programming (and other web skills) went from being about teaching reasoning to being a skill students and humanists could actually use to create humanities products, i.e., websites. Programming became creative and expressive. There was also a convenient on-ramp as students could start creating HTML pages and then learn to use scripting languages like PHP that enhanced the web site until you knew enough to code a scholarly web site.

We will further argue that once web development wasn’t something that digital humanists would necessarily be amateurs at, then it could become a defining skill, and a source of some pride. Digital humanists finally had something that they could be good at and a set of competencies unique to the digital humanities. Digital humanists like Stephen Ramsay could provocatively say, ‘Do you have to know how to code? I’m a tenured professor of Digital Humanities and I say “yes”’ (Ramsay 2011). This led to the ‘hack vs. yack’ discussion that followed about programming and other forms of building (see Nowviskie [2014] for more general context).

**Conclusions: Programming in the Humanities and Disciplinary Formation**

This paper traces the ways programming has been discussed in the digital humanities. We believe that the role of programming was important to the way the field of humanities computing (and later digital humanities) conceived of itself. This is not surprising given the importance of programming to computing in general, but it is interesting to follow the particular ways programming was discussed as the discipline emerged.

We will end the paper by theorizing, or at least speculating, about where programming in digital humanities is going. One direction (already well under way) is towards software studies and the studying of programmed artifacts as works of human art and expression worthy of the humanities. Matthew Kirschenbaum in a fine essay for the *Chronicle of Higher Education* talks about ‘procedural literacy’.

All programming entails world-making, as the ritual act of writing and running Hello World reminds us. . . .

Ultimately what’s at stake is not the kind of vocational computer literacy I was taught as an undergraduate, but what a growing number of practitioners in the digital humanities (and related disciplines, like digital art or game studies) have begun to call procedural rhetoric, or procedural literacy. (Kirschenbaum, 2009)

Another direction is data science. The opportunities for new insights through large-scale text mining, or what Moretti (2007) calls distant reading, have made programming languages like R attractive. There is a return to text processing languages, but now languages that can analyze large corpora or visualize results.

**References**

**Ide, N. M.** (1987). Computers and the Humanities Courses: Philosophical Bases and Approach. *Computers and the Humanities,* **21**(4): 209–15.

**Kirschenbaum, M.** (2009). Hello Worlds. *Chronicle of Higher Education,* **55**(20): B10, http://chronicle.com/article/Hello-Worlds/5476.

**McCarty, W.** (1987). HUMANIST So Far: A Review of the First Two Months. *ACH Newsletter,* **9**(3): 1–3.

**McCarty, W.** (1993). Handmade, Computer-Assisted, and Electronic Concordances of Chaucer. *CCH Working Papers,* **3**: 49–65.

**Moretti, F.** (2007). *Graphs, Maps, Trees: Abstract Models for Literary History*. Verso, London.

**Nowviskie, B.** (2014). On the Origin of ‘Hack’ and ‘Yack’. http://bit.ly/1ttih76.

**Olsen, M.** (1987). Beyond SNOBOL: The Icon Programming Language. *Computers and the Humanities,* **21**(1): 61–66.

**Parrish, S. M.** (1969). Concordance-Making by Computer: Its Past, Future, Techniques, and Applications. In Burelbach, F. M. (ed.), *Proceedings: Computer Applications to Problems in the Humanities; A Conversation in the Disciplines*, Conference at State University College, Brockport, NY, 4–5 April 1969, pp. 16–33.

**Ramsay, S.** (2011). On Building. http://stephenramsay.us/text/2011/01/11/on-building/.

**Rudman, J.** (1987). Teaching Computers and the Humanities Courses: A Survey. *Computer and the Humanities,* **21**(4): 235–43.

**Sinclair, S. and G. Rockwell.** (2014). *Voyant Tools*. http://voyant-tools.org/.

**Sperberg-McQueen, C. M.** (1986). Report on ACH/Sloan Foundation Workshop on Teaching ‘Computers and the Humanities’ Courses. *ACH Newsletter,* **8**(4): 1–2.