lished infrastructure.

References

Anonymous (2008). *Supporting Digital Scholarly Editions: A Report on the Conference of January 14, 2008*. http://www.virginiafoundation.org/NEH%20Workshop%20Report%20FINAL-3.pdf (accessed 14 November 2008).

Cummings, James (2007). The Text Encoding Initiative and the Study of Literature. In Siemens, Ray, and Schreibman, Susan (eds), *A Companion to Digital Literary Studies*. Oxford: Blackwell, pp. 451–76.

Digital Humanities Observatory. http://dho.ie (accessed 14 November 2008).

Finneran, Richard J. (ed) (1996). The Literary Text in the Digital Age. Ann Arbor: University of Michigan Press.

Interedition. http://www.interedition.eu/index.php/Main_Page (accessed 14 November 2008).

McGann, Jerome (2002). Literary Scholarship in the Digital Future. *Chronicle of Higher Education*, 13 December: B7–B9.

Price, Ken (2007). Digital Scholarly Editions. In Siemens, Ray, and Schreibman, Susan (eds), *A Companion to Digital Literary Studies*. Oxford: Blackwell, pp. 434–50.

Schreibman, Susan (2002). Computer-mediated Texts and Textuality: Theory and Practice. *Computers and the Humanities*, 36: 283–93.

TextGrid. http://www.textgrid.de/ (accessed 14 November 2008).

Critical Code and Software Studies

Chair: Marc Marino
University of Southern California

For the future of Digital Humanities, software cannot remain either a black box or some sacred text that only the high priests can access. Software intersects with and coordinates so many aspects of our lives, and a few lines of source code can determine processes all around us. In 2008, The University of California San Diego organized the *Softwhere Studies* workshop to further the development of Software Studies, the critical analysis of the cultural circulation of computer software. Critical Code Studies is a subset of Software Studies that uses critical theory to explicate the extra-functional significance of computer code, exploring not merely what the code does but what it means. This panel offers a series of talks that take up a Software Studies or Critical Code Studies approaches to demonstrate the role of reading code and software using the interpretive approaches of the humanities. Such moves will lead not only to the tearing down of disciplinary boundaries but also a vital understanding of the way software and source code structure our political and personal lives. After a capsule definition of critical code and software studies, panelists will present their papers.

Paper 1: Expressive Processing

Noah Wardrip-Fruin
University of California, Santa Cruz
nwf@ucsd.edu

The forthcoming book *Expressive Processing* (MIT Press, 2009) uses its title's term to point toward two important issues for humanities scholars.

First, "expressive processing" encompasses the fact that the internal processes of digital media are designed artifacts, like buildings, transportation systems, or music players. As with other designed mechanisms, processes can be seen in terms of their efficiency, their aesthetics, their points of failure, or their (lack of) suitability for particular purposes. Their design can be typical, or unusual, for their era and context. The parts and their arrangement may express kinship with, and points of divergence from, design movements and schools of thought. They can be progressively redesigned, repurposed, or used as the foundation for new systems - by their original designers or others - all while retaining traces and characteristics from prior uses.

Second, unlike many other designed mechanisms, the processes of digital media operate on, and in terms of, humanly-meaningful elements and structures. For example, a natural language processing system (for understanding or generating human language) expresses a miniature philosophy of language in its universe of interpretation or expression. When such a system is incorporated into a work of digital media - such as an interactive fiction - its structures and operations are invoked whenever the work is experienced. This invocation selects, as it were, a particular constellation from among the system's universe of possibilities. In a natural language generation system, this might be a particular sentence to be shown to the audience in the system output. From the output sentence it is not possible to see where the individual elements (e.g., words, phrases, sentence templates, or statistical language structures) once resided in the larger system. It is not possible to see how the movements of the model universe resulted in this constellation becoming possible - and becoming more apparent than other possible ones.

Paper 2: What Counts as Code to Criticize? Interpreting flow control and natural language programming

Jeremy Douglass
University of California, Santa Barbara
jeremydouglass@gmail.com

Sites of interpretation in the digital humanities range from the signifiers at the software interface all the way down to the electromagnetic phenomenology of hardware mechanisms. Critical Code Studies situates interpretation at the site of reading and writing source code. These human-machine interlanguage exists both between and prior to the hardware and its interface, often with many other interpretive sites above (application, OS, emulator, etc.) and below (assembly, machine code, etc.). As interlanguage, code carry a variety of philosophical presumptions, aesthetic overdeterminations, and historical traces that may benefit from exegesis. But what counts as code to criticize? This presentation considers critical interpretation in relation to two cases of code that differ substantially from the paradigmatic use of languages such as C/C++ and the attendant assumptions about the relationships between code, programmer, compiler, and software. The first case, natural language programming, emphasizes the expressive power of code and its accessibility to non-programmers, while it simultaneously risks obscuring the deep structures that connect compiled code to software praxis. The second case, flow control programming, uses visual spatial relationships rather than text as the top-level paradigm for specifying software action. Both cases expand our idea of what aspects of rhetoric and visual culture might be brough to bear in humanistic code criticism.

Paper 3: Hacktivism and the Humanities: Programming Protest in the Era of the Digital University

Elizabeth Losh
University of California, Irvine
lizlosh@uci.edu

Despite the dismay of university administrators, who are often hesitant to recognize computer programming as a form of campus free speech, students continue to deploy code to further activist agendas Yet media reports about hacktivist activities by undergraduates and graduate students rarely place these computer-coded expressions of protest in the context of other kinds of campus activism or critical engagement. Whether the student is generating electronic boarding passes to protest Homeland Security policies or data mining for self-interested Wikipedia edits by corporations and politicians, the attention goes to the programmer's identity as a hacker rather than as a student. Although Siva Vaidhyanathan and others have issued a manifesto to promulgate "critical information studies" as an interdisciplinary field of study that could serve as the logical successor to the areas of academic inquiry that arose from the protests of the nineteen-sixties and seventies, which is "needed to make sense of important phenomena such as copyright policy, electronic voting, encryption, the state of libraries, the preservation of ancient cultural traditions, and markets for cultural production," advocacy for these issues in the university setting does not achieve the kind of visibility that was associated with previous movements that assembled crowds of individuals for face-to-face interactions in physical public space to achieve an end to the Vietnam War, milestones on civil rights issues, affirmative action, or divestment in South Africa. This presentation builds upon the models of critical information, critical code, and software studies to examine the analytical frameworks that might reinvigorate the program of the political explication of coding spaces.