Adrian Mackenzie: Cutting Code. Software and Sociality Peter Lang: New York 2006. 215 pages.

Highly mobile and distributed – code is everywhere in the global knowledge economy. But can one follow intangible code through society? How can code be made intelligible as a social object? Where is the location of agency, when it comes to software? These are just a few of the questions Adrian Mackenzie deals with in *Cutting Code. Software and Sociality*. He sets out to explore software at the same time "as a material object, as a means of production, as a human-technical hybrid, as medium of communication, as terrain of political-economic contestation – in short as sociality" (p. 2).

Among the fundamental challenges such a project faces, is the very problem of delineating code as a cultural object. Similar to numerical methods and standardisation procedures, software is often perceived as a neutral tool. Epitomising the abstract and the invisible, code presents an object that has long seemed restricted to logic or linguistic analysis. Yet, Mackenzie clearly moves beyond linguistic understandings of code. Unlike many studies of digital culture, the book does not primarily focus on the mediality of the software interface or on virtuality; instead, it turns to the very grounds of code and coding work. While early computing was organised around the 'centers of calculations' of mainframe computers, contemporary code production takes place in highly distributed networks. In order to tackle invisible and easily naturalised code, which remains hidden unless errors occur, Mackenzie draws on social anthropology and STS classics – from Geoffrey Bowker's & Susan Leigh Star's 'boundary infrastructures' in *Sorting Things Out*, Alfred Gell's anthropological analysis of art objects, George Marcus' multi-sited ethnography to Bruno Latour's *Science in Action*.

Building on these tools, Mackenzie treats the formality of code not as natural or fixed, but seeks to understand formalisms through the articulations they contain. The main units of analysis—code, originators, recipients and prototypes—are derived from Gell's theory of art as an index of agency (p. 11). The studied sites range from code-based artwork, software platforms, bioinformatics to telecommunication and extreme programming. Even though the introductory chapter provides helpful guidance through the multiple sites and theories, it demands considerable effort to grasp the book's main arguments and line of thought: Introducing code as cutting "across every aspect of what software is and what software does" (p. 3), Mackenzie gradually unfolds various and complex social layers around code. His aim in tying together these particular studies, some of them previously published as journal articles, is to "map some involutions of agency running across conventional analytical divisions between code as expression and code as action" (p. 19) and to relate these to larger questions of power, identity and democracy.

Starting with code excerpts, the reader is taken through the rich multilayered social environments of code. Exploring "how people's lives cross into code" (p. 41), the subsequent empirical studies give informed insights into the world of code and software development. The life of code as algorithm, as a set of steps expressed in code, is discussed for the case of gene sequence alignment software. Here, Mackenzie explores the 'stickiness' of algorithms: While transferable across domains and inducing movement, they also carry over categories and conventions between different contexts.

To what extent code is mobile and distributed in time and space is illustrated with the open-source concept of the Linux kernel, which has long been considered a platform for hackers collaborating on the re-use of Unix code. Noting that computer code at the first glance appears "an unlikely candidate for performativity analyses" but rather "an exemplar of formal clarity and univocity" (pp. 74-75), Mackenzie explores the "circulatory performativity" in software to make visible agential effects and "collective subjectification" (p. 90). The Java language which promised "write once, run everywhere", has proved much more entrenched in sociality and connected with older media than assumed by early accounts of new media and virtuality (p. 92). As Mackenzie argues, code is embedded in processes of production and consumption, and closely connected to social relations of groups and institutions; moreover, contrary to the collaborative open source programming ethos, software has become increasingly branded and commodified.

Two case studies discuss code within software development contexts and in relation to the individuals and groups which meet these infrastructures. It is shown how, in collaborative development of telecommunication software, coding work becomes attached to "collective imagining" (p. 138), e.g., of mobility and reconfigurability. Another layer of sociality surfaces in extreme programming, which has scaled back to coloured index cards and takes place at a group table. Pair programming in software development has replaced individual coding autonomy; in the very process of coding, intersubjective communication has deliberately been introduced to detect system integration problems at an early stage. Mackenzie convincingly shows how compiling and running code are used as metaphors, as iterations and testing routines are built into the entire process. Thus, software production has come to look more like continuous modulation – as it relies on intersubjective articulation and interactive loopings throughout the development process.

Cutting Code is an insightful and ambitious book towards a theory of software inspired by STS, social anthropology and media theory, which makes it an important contribution to the emerging field of software studies (Manovich, 2001; Fuller, 2003). Mackenzie insists on the multiplicity of code – "as permeated by all the forms of contestation, feeling, identification, intensity, contextualisations and decontextualisations, signification, power relations, imaginings and embodiments that comprise any cultural object" (p. 5). Equipped with science studies' tools, Mackenzie succeeds in tracing how code enacts distributions of agency; via close studies of coding work, the sociality in software becomes intelligible. The book provides in-depth insights into how software and code co-shape communication and personhood. Occasionally, long lists of conceptual terms may make it difficult for readers to grasp the key points. Yet, this conceptual density, if not overload, may mirror the difficulty of the task of analytically pinning down volatile code. Code easily disappears behind well functioning surfaces and ways to tackle questions of agency in software still need to be developed. Mackenzie works towards "an ontology of software in order to handle code as material and practice" (p. 5), conceived here in analogy to Gell's anthropological theory of art. As a provocative engagement with an actor's category— computational ontologies— Mackenzie's ontology is aimed at registering malleability and contingency.

Certainly not an easy read and sometimes difficult to see through its conceptual density, *Cutting Code* provides a valuable overview and rich empirical studies that benefit from the insights of a previous software developer. The book will inspire all those interested in the social life of software and engaged in tackling the agency of code – to develop better understandings of how software systems co-shape everyday life and communication. Of interest to scholars of STS, media theory and digital culture as well as to computer scientists and CSCW scholars, *Cutting Code* invites more science studies research into software and into the performativity of invisible code.

References

Fuller, Matthew (2003) Behind the Blip: Essays on the culture of software (New York: Autonomedia).

Manovich, Lev (2001) The Language of New Media (Cambridge, MA: MIT Press).

Susanne Bauer
Medical Museion, University of Copenhagen, Denmark.
susanne.bauer@mm.ku.dk