Book Reviews

Coding Places: Software Practice in a South American City. By Yuri Takhteyev. Cambridge, Mass.: MIT Press, 2012. Pp. ix+257. $34.00.

Steve G. Hoffman
University at Buffalo, SUNY

The world of computer software is not much like international cuisine. Firms do not seek out "Brazilian software" like foodies do a good *feijoada*. Software is supposed to be universal and placeless. Of course, like all of social life, it isn't.

The software industry is both globally dispersed and highly centralized, most notably within professional hubs like Silicon Valley. In addition, software development tends to be team-oriented work, despite the popular stereotype of asocial computer geeks clacking away at lonely terminals. These characteristics form a contradictory combination of consumer demand for product universality and high regional clustering of product development. This contradiction creates its most acute dilemmas for developers located at the industry's periphery. How does one harness the cultural, human, and material resources of a particular place in order to create products that are placeless (or, at least, not clearly *brasileiro*)? Or, as one developer puts it, how to create products that will "succeed abroad to gain acceptance at home" (p. 165)?

In *Coding Places*, Yuri Takhteyev mobilizes participant observation and interview data to document how software engineers in Rio de Janeiro cobble together ad hoc solutions to this fascinating insider-outsider dilemma. Although the effort is not theoretically dynamic, the book does provide a very cleverly chosen case study of what Yakhteyev refers to as a "wrong place" for software development. It helps fill an important gap in research on the computer software industry since the vast majority of software developers live in "wrong places" like Rio. Scholars generally interested in the pressures of globalization on peripheral actors, or, those specifically interested in the production of South American computer software, should find this study a welcome addition to their library.

In addition to chapters on the history of computing in Brazil and description of the local scene, the book documents three software development projects centered in Rio. First there is Alta, a company that specializes in tailored software solutions for local firms. Alta is as an example of

a typical business strategy in Rio. The company achieves modest success by using software made mostly in California to solve local business needs. Next there is Lua, a very atypical international success. Lua is a programming language that has risen to some prominence within the mobile computing and video game industries. It is used in the hit games *World of Warcraft* and *Angry Birds*, among others. Its anomalous development involves complex linkages between local and global human capital; a national university; public, private, and semiprivate industry; international corporations; research policy; and open-source software development. Savvy, though never prescient, decision making on the part of its authors characterizes the technology's consistent separation from the local context. Despite this international success, however, Lua's inventors are largely unable to monetize their work or raise the profile of the Rio and Brazilian software industry. Last, Takhteyev documents the development of Keplar, an ambitious attempt to use Lua in a global Web site application package. Keplar's model for success, a typical one in America, proves the hardest to execute in Rio. The team joins a local company and taps into government resources to build a local user base before scaling up globally. Tellingly, the project never gains much of a foothold in either a local or global market.

This book provides many good examples of the disadvantages of practicing supposedly placeless "knowledge work" in peripheral places. For example, if an engineer in Rio wants to learn Lua, he or she would have to study a manual written in English, German, or Korean. There is no Portuguese translation. Never mind that the same Brazilian team that invented the language also maintains it, and that one of them, a well-known Rio-based academic computer scientist, wrote the manual. In fact, the easiest way to obtain a copy is to order it from Amazon.com and have it shipped from America!

The lingua franca of software programming is English and its cultural standards are those of Silicon Valley. In addition to obvious material, cultural, and network disadvantages, this fact also means that Rio-based developers work hard to avoid the whiff of parochialism. Programs that seek international success must be documented exclusively in English. Discussion boards too. Most college students assume that local software is backward, useless, or irrelevant to cultivating the skills necessary for global labor pools. Paradoxically, efforts to avoid parochialism often contribute to further marginalization. On the one hand, it can be highly problematic if city or state officials promote a local software success story with regional or nationalistic pride. On the other hand, by not doing so, there is little reason to question the orthodox wisdom that nothing much exciting is happening.

A lack of theoretical rigor and verve sometimes underserves this book's rich empirical substrate. For example, Takhteyev's account of the various

challenges faced by the developers of Lua is fascinating stuff, especially the discussion of open-source software. However, this point is also where a descriptive "social worlds approach" provides a too-vague explanation for the fundamental power asymmetries that characterize the geopolitics of computer software design. Similarly, the author gravitates toward noncommittal explanatory concepts, most notably Anthony Giddens's notions of "disembedding," "reembedding," being "co-constitutive," "symbolic tokens," and the like. Elsewhere we get pretentious jargon like "subvocal imagination" where "vague aspiration" would serve fine. In a similar vein, much of the inferential logic reduces to "just-so" expressions—a contingent description with little framework for ordering consequences or essential parts.

Coding Places goes through the standard motions on this account: situate the empirical case, borrow a bit of conceptual terminology, and then speculate on why social action transpired as it did based primarily on the observation that it did, in fact, transpire as it did. These criticisms aside, that a great deal of software development occurs where Takhteyev found it should be enough to widen any readers' appreciation for the unique challenges faced by professionals at the margins of a global industry.

Is American Science in Decline? By Yu Xie and Alexandra A. Killewald. Cambridge, Mass.: Harvard University Press, 2012. Pp. x+230. $45.00.

Harriet Zuckerman
Columbia University

Efforts to assess the condition of a nation's science go back to the 19th century and Charles Babbage's *Reflections on the Decline of Science in England* (B. Fellowes and J. Booth, 1830). Babbage, best known as the inventor of the "calculating engine" or computer, was an advocate for returning British science to its previous glories. His aims were political and his biases were evident. Even so, historians of science count his work as the first in what is now a long line of quantitative assessments of the scientific enterprise (e.g., Arnold Thackray, "Measurement in the Historiography of Science," in *Toward a Metric of Science*, ed. Elkana et al. [Wiley and Sons, 1978]). Consider just two examples of this genre: the U.S. Science and Technology Indicators project beginning in 1972 (National Science Board, Government Printing Office) and the OECD's multiple data collections on science in its member states (e.g., OECD, "Science, Technology, and R&D Statistics" [multiple versions since 1961]). Yu Xie and Alexandra A. Killewald's study of American science, *Is American Science in Decline*, is a latter-day descendent of the Babbage book; the resemblance in title and intent is clear. The similarities however stop there. Xie and Killewald have