With the advent of Web 2.0, the boundaries between consumers and producers of software have become less distinct (Fischer, 2009). Youth are at the forefront of those participating in social networking sites, contributing media content in various forms. More and more youth regularly act as content creators and informal programmers (Lenhart & Madden, 2007). Be it modifying a cell phone ring or creating content for a social Web site like MySpace, youth employ varying computational skills to tailor media to their own needs and tastes. And while modifying a cell phone ring is hardly the same as writing a compression algorithm, youth are nonetheless regularly encountering and navigating programmable media. To fully participate in Web 2.0 communities requires new competencies of youth (Jenkins, Clinton, Purushotm, Robison, & Weigel, 2006) involving more technical and creative dimensions (Kafai & Fields, 2009; Peppler & Kafai, under review).

While the popularity of file-sharing sites suggests that young end-user designers join and participate in large numbers, in the particular case of programming the situation is often different. Much research has documented over and over again that most youth do not know what programming is, do not have access to programming skills, and perceive it as overly technical and thus not for them (Margolis, 2008).
Social and cultural dynamics such as agency, membership, and status are thus instrumental in getting novices into programming (Kelleher & Pausch, 2005; Margolis, 2008; Margolis & Fisher, 2002). The focus of this paper will be how young programmers negotiate their entry into the club—an online community of programmers—its title making reference to the landmark research by Margolis and Fisher (2002).

A recently developed file-sharing site for informal programmers, scratch.mit.edu, allows us to examine in more depth how youth become engaged as end-user designers in Web 2.0 in the particular context of Scratch and its associated file-sharing site. As part of a 4-month ethnographic study, we followed two 12-year-old participants, Lucetta and Matthew, as they learned the programming software Scratch (Resnick et al., in press) and then joined scratch.mit.edu both in an after-school club and in a class. Participation in Scratch shares many similarities and challenges with professional communities and thus we can study this process among the incoming generation of users.

BACKGROUND

Traditionally end-user development has been concerned with professionals and how they can customize tools to accomplish their work. Much of the research has either studied what problems end-user designers encounter in this process or how to design tools that would be supportive of their endeavors (e.g., Lieberman, Paterno, & Wulf, 2006). This research has been largely separate from efforts that have covered the same territory, albeit with young end-users, in school contexts. Here end-user development has been concerned with designing environments and tools that support novices in learning of programming (Guzdial, 2004). Ultimately, end-user development for professionals was seen as facilitating the modification of tools, whereas the focus for youth was on designing tools for ease-of-use, taking into account the differences in motivation, background, and developing expertise between young learners and adult professionals (Soloway, Guzdial, & Hay, 1994).

The research on learning programming and designing novice programming languages and environments has a long tradition (for an extensive overview see Kelleher & Pausch, 2005). Early studies focused mostly on understanding the design and ease-of-use of specific programming concepts such as loops, conditionals, or recursion (e.g., Soloway & Spohrer, 1989). Several efforts concentrated on designing scaffolds for beginning programmers to ease syntax and control problems (Guzdial, 1995; Jackson, Krajcik, & Soloway, 1998) or have developed new genres of interfaces that generated scripts based on users’ interactions (e.g., Cypher, 1993). Perhaps the most longstanding effort has been design of programming languages for students based on an object-oriented programming paradigm which has now has become an industry standard: Logo and Smalltalk were the predecessors of today’s Agentsheets, Alice, and Scratch (Guzdial, 2004).

When Kelleher and Pausch (2005) reviewed the development of programming environments and languages for novice programmers, they found that most efforts aimed at making the mechanics of programming more manageable. But more importantly, Kelleher and Pausch identified “the lack of a social context for programming, and the lack of compelling context in which to learn programming” (p. 132) as key impediments to getting programming novices involved and supported. They argued that these social and cultural barriers are “harder to address than mechanical ones because they are harder to identify and some cannot be addressed through programming systems.” As the most recent designed novice programming language, Scratch facilitates many mechanical aspects of programming (see description below). In particular media-manipulation features provide the compelling context appealing to youth’s interests in digital media, but Scratch also features a networked community and file-sharing site (scratch.mit.edu)—much like many profes-
Related Content

End User Development and Meta-Design: Foundations for Cultures of Participation
Gerhard Fischer (2010). *Journal of Organizational and End User Computing* (pp. 52-82).
[www.igi-global.com/article/end-user-development-meta-design/39120?camid=4v1a](www.igi-global.com/article/end-user-development-meta-design/39120?camid=4v1a)

Exploring the Dimensions and Effects of Computer Software Similarities in Computer Skills Transfer
[www.igi-global.com/chapter/exploring-dimensions-effects-computer-software/69614?camid=4v1a](www.igi-global.com/chapter/exploring-dimensions-effects-computer-software/69614?camid=4v1a)
Mobility and Multimodal User Interfaces
www.igi-global.com/chapter/mobility-multimodal-user-interfaces/18235?camid=4v1a

Accurately Determining Self-Efficacy for Computer Application Domains: An Empirical Comparison of Two Methodologies
www.igi-global.com/article/accurately-determining-self-efficacy-computer/37218?camid=4v1a