End User Programming for Scientists: Modeling Complex Systems

Andrew Begel
Microsoft Research
andrew.begel@microsoft.com

Towards the end of the 20th century, a paradigm shift took place in many scientific labs. Scientists embarked on a new form of scientific inquiry seeking to understand the behavior of complex adaptive systems that increasingly defied traditional reductive analysis. By combining experimental methodology with computer-based simulation tools, scientists gain greater understanding of the behavior of systems such as forest ecologies, global economies, climate modeling, and beach erosion. This improved understanding is already being used to influence policy in critical areas that will affect our nation's future, and the world's.

Some computer tools enabled scientists to create models of phenomena from first principles, rather than from descriptive differential equations. These tools, which directly modeled complex adaptive systems, significantly lowered the mathematical burden required of scientists to understand and create models. Tools such as StarLogo (Klopfer & Begel, 2003), Swarm (Minar, Burkhart, Langton, & Askenazi, 1996), and Repast (North, Collier, & Vos, 2006) enable scientists to program a simulation of a system by describing the behaviors of the individual elements of the system (e.g. each animal eating another, each consumer purchasing a product, each molecule of air and particle of cloud, and each grain of sand and drop of water). These tools reduce the barrier to entry by providing a framework in which to develop models, but they require a degree of programming sophistication to accomplish even relatively simple tasks. Swarm and Repast require the scientist to program in Objective-C and Java, respectively. StarLogo reduces the barrier more than the others through its use of Logo, a more accessible language most often associated with children's programming projects. A more recent version of StarLogo, called TNG (Klopfer & Begel, In Press), improves accessibility to non-programmers further by using a graphical programming language.

Modeling follows the scientific method: hypothesis, experiment creation, observation, evaluation, only instead of studying a system in the real world, a model is created and studied instead. Rather than giving scientists black-box models in which they can only study what they have been given, and only tweak knobs that the author provided, the StarLogo programming environment enables scientists to be model designers and builders, by enabling them to program the behaviors of the entities they want to interact with using the Logo programming language. Programming is a means to an end, yet in order to enable scientists to model what they want to study, it is often the only means.

We have used StarLogo to teach the scientific method and modeling to high school students. Through a series of workshops, called Adventures in Modeling (Colella, Klopfer, & Resnick, 2001), high school students and teachers (and school district technology coordinators) have learned what complex systems are, how to program in StarLogo, how to model a complex system using StarLogo, and how to conduct scientific inquiries using the StarLogo modeling environment. Participants work through a series of participatory activities, games that involve the participant as one of the entities in a complex system. For example, in the majority-minority game, participants must discover what the majority of the group has decided, secretly, about which color chile they like the best, green or red. They can only move around while blindfolded, and whisper anything they like to whomever can hear them. At the end of each round, a vote is taken to determine which chile is the best; and each participant must vote with the choice they think the majority has chosen. The vote tallies initially begin quite divergent, but as the rounds progress, a kind of positive feedback loop forms, with the majority winner being whispered more often, and winning over more votes. Eventually, the majority dominates. The minority game is similar, but participants must pick the chile that the minority of people like. Vote tallies in this game often fluctuate from one extreme to the other; as participants hear more people saying one color chile, they pick the opposite, leading to an unstable dynamic that never converges. After playing the game, participants learn to program it in StarLogo. They develop variants, and run experiments to understand the behavior under different conditions, for example, greater or fewer people, no blindfolds, communication louder than a whisper, or communication only by touch.

The StarLogo workshops were successful at teaching non-experts to program, and we have heard many reports from scientists in many fields of study who have used StarLogo to model systems they were researchers. However, we have found that StarLogo programming can be difficult to pick up, especially when learned on a hobby basis, or without an instructor. Even worse, the longer a novice scientist goes between StarLogo programming sessions, the less they retain, and the more apprehensive they get about creating their own models. It is critical, however, for scientists to be able to design, build and conduct experiments in models that they build themselves. Model creation cannot be turned over to a programmer-for-hire without causing the model to become a black box. In order to ensure the validity of the model and stand behind its experimental results, the scientist must be intimately knowledgeable about its innards as well as its outward behaviors.

Thus, it is important to understand how non-programmers pick up programming languages when the task they want to complete cannot be accomplished in other ways. How does motivation drive learning in the absence of teachers, or a community of learners, which is the usual model of learning to program in school? Unless the fidelity of the finished model is quite high, even demonstrating the model to non-modeler audiences can prove difficult. How are search engines used to provide sample code, explanations, and project ideas, especially when the software modelers use is not widespread, or is new?

Learning a text-based programming language is difficult for novices who want to be programmers. In the first few weeks of learning a language, syntax rules are often the most difficult to comprehend, with semantics interleaved. Non-programmers face these problems, in addition to lacking an engineering mindset to help form mental models of how they want to make the computer do what they want. How does learning graphical programming languages like LabView, ProGraph or StarLogo TNG differ from learning text-based languages in this context? Is the floor lower? Is the ceiling lower? Are the walls more narrow? Graphical languages have not achieved popularity among computer scientists, but remain fashionable in educational settings. Is this making a difference? Does exposure to programming prior to college enable non-programmer scientists to understand and create models more easily?

How does one characterize an expert in a modeling language? When we, as computer scientists, see non-programmers' StarLogo programs, we might cringe at their inelegance. Yet, if the non-programmer is achieving their modeling goals, then their program is effective and just as valid as an elegant one. Is it important to turn expert non-programmers into proper engineers? Can experts teach other non-programmer novices properly? Does an expert's lack of formal instruction hinder their instruction or interfere with novice learning? Is it better or worse than no instructor at all? What can be done to ensure that a model's validity is not affected by poor programming? Can automated tools help a non-computer-scientist see coding flaws and help him to fix them?

Understanding how non-programmer scientists attain and disseminate expertise in programming will help us to design easier to use modeling environments that result in more understandable and maintainable programs. Our goal is to enable all scientists, even the ones who are apprehensive about computer programming, to create and study their own models of complex systems and use them in their research.

References

Colella, V., Klopfer, E., & Resnick, M. (2001). *Adventures in Modeling: Exploring Complex, Dynamic Systems with StarLogo.* Teachers College Press.

Klopfer, E., & Begel, A. (2003). StarLogo in the Classroom and Under the Hood. *Kybernetes , 32* (1/2), 15-37.

Klopfer, E., & Begel, A. (In Press). StarLogo TNG: An Introduction to Game Development. *Journal of E-Learning* .

Minar, N., Burkhart, R., Langton, C., & Askenazi, M. (1996). *The Swarm Simulation System: A Toolkit for Building Multi-Agent Simulations.* Santa Fe: Santa Fe Institute.

North, M. J., Collier, N. T., & Vos, J. R. (2006). Experiences Creating Three Implementation of the Repast Agent Modeling Toolkit. *ACM Transactions on Modeling and Computer Simulation , 16* (1), 1-25.