

# Chapter 2

## Software Solutions for Computational Modelling in the Social Sciences

Piper J. Jackson

**Abstract** Social science is critical to decision making at the policy level. New research in the social sciences focuses on using and innovating new computational methods. However, as a relatively new science, computational research into the social sciences faces significant challenges, both in terms of methodology and acceptance. Important roles software can take in the process of constructing computational models of social science phenomena are discussed. An approach is presented that frames software within this kind of research, aiming at involving software at an early stage in the modelling process. It includes a software framework that seeks to address the issues of computational social science: iterative knowledge development; verification and validation of models; and communication of results to peers and stakeholders.

### 1 Introduction

With the emergence of cheap and powerful computation, social scientists have started to explore the potential of applying computational to their research topics [1]. Hummon and Fararo claim that the traditional two component model of science—theory and empirical research—needs to be expanded to include computation as its third component. Simulation can be thought of as the interaction of theory and computation components. High-level programming code, with the capacity to clearly represent both entities (data structures) and behaviour (algorithms), provides a conceptual

---

P. J. Jackson (✉)

Modelling of Complex Social Systems (MoCSSy) Program, Interdisciplinary Research in the Mathematical and Computational Sciences (IRMACS) Centre, Simon Fraser University, 8888 University Drive, Burnaby, BC V5A 1S6, Canada  
e-mail: [pjj@sfu.ca](mailto:pjj@sfu.ca)

P. J. Jackson

Software Technology Lab, School of Computing Science, Simon Fraser University, 8888 University Drive, Burnaby, BC V5A 1S6, Canada

link between the underlying mathematics of the simulation model and researchers' understanding. Being able to test theories using mathematics promises insight formerly limited to the physical sciences, but it is important to remember that social structures are always at least partly interpretive in nature, since they are constructed and maintained by human activity. Gilbert claims that this is not problematic, and indeed may even be more faithful a representation than simulation of physical phenomena: a translation from a (social) construct to a (computational) construct is likely to be less problematic than translating something in the real world into a constructed computer representation [2].

For the social sciences, applying computational techniques helps in overcoming some of the core limitations of studying social phenomena. Social scientists have always been limited by the inextricability of the subject of their research from its environment. Hence, it is difficult to study different factors influencing a phenomena in isolation. Safety and ethical issues can be an obstacle to innovation, e.g., for criminologists, it is very difficult to get first-hand evidence of crimes while they are being perpetrated—an observer would most likely be legally required to try to prevent the crime rather than letting it take place. Developing response strategies to unpredictable and dangerous situations is difficult to do in the field, since such situations are unpredictable and by their nature very difficult to control. Computational methods allow us to circumvent these problems by generating scenarios inside a virtual environment. In particular, modelling and simulation allow us to dynamically and interactively explore our ideas and existing knowledge.

Computational thinking about social phenomena, however, means thinking in terms of multiple layers of abstraction [3], which facilitates a systematic study of the phenomena by adjusting the level of detail given to the various factors under study. Computer models of social systems simulate dynamic aspects of individual behaviour to study characteristic properties and dominant behavioural patterns of societies as a basis for reasoning about real-world phenomena. This way, one can perform experiments as a mental exercise or by means of computer simulation, analyzing possible outcomes where it is difficult, if not impossible, to observe such outcomes in real life.

However, when we speak of computational methods, it is easy to gloss over the software that is the means to our research goals. We have an intuitive model of what a software program should be, and it is easy to apply this in a superficial sense to the problem of building a social system simulation. Yet the characteristics and functionality of that software are of great importance to our work, as are the roles which we want it to play in our research endeavours. The aspects that make software successful in an interdisciplinary computational social science research project will be outlined here. From these roles, a framework has been developed that is presented here. This framework aims at circumventing problems commonly faced, and increasing the productivity and advancement of groups working on these kind of simulation research projects.

The chapter is organized as follows. Related work is described in Sect. 2. Issues related to the software development process are discussed in Sect. 3. Section 4 describes the main pillars of a social science computational modelling project, as

well as the characteristics such an endeavour should possess. Section 5 is a presentation of the framework, followed by some final words in Sect. 6.

## 2 Related Work

Discussion and guidelines for how social simulations can be developed using the input of multiple stakeholders is presented in [4]. Importantly, human factors, such as maintaining stakeholder motivation, and technical factors, such as the importance of a formal computing models, are emphasized in their approach.

A particularly complete methodology for prototyping and developing agent-based systems is presented in [5]. The methodology described, ELDAMeth, includes an overall development lifecycle model, automated code generation based on high-level models, and testing through simulation of the target system.

There are numerous multi-agent modelling toolkits available, which include programming libraries and/or languages that facilitate the production of simulation software. A common and important component are methods for visualizing models graphically, reducing the need for toolkit users to have specialist technical knowledge in that kind of programming. Two toolkits are of particular note. Repast has a wide user base and allows for the creation of models using ReLogo (a dialect designed for Repast), Java, and flowcharts [6]. MASON is another toolkit with which a variety of model types can be developed, and is notable in that both the simulation itself and its data visualization can be run concurrently [7].

More generally, Nigel Gilbert and his many colleagues have been central in promoting the use of simulation models in social science research. One example is [8], in which the motivation and general methodology for using such models for social simulation is presented. Other work in this field has suggested common approaches and vocabulary to describe agent-based social simulations [9].

## 3 Software Development

Software development methodologies enable systematic production of software. Agile methodologies have found widespread adoption in industry due to higher rates of success while using fewer resources. Here success is in terms of getting a project done on time and on budget. The category of agile methodologies includes Extreme Programming (XP), the Rational Unified Process (RUP) and Scrum, among others. Agile methods are diverse in practice, but share a number of important characteristics. First of all, they emphasize the role of people in the development process. Thus, how people interact and relate to each other is considered as an important factor in the production of software. This includes clients as well as the people working on the software. They also emphasize flexibility in the face of change and obstacles, which is appropriate since this is a strength of software itself. A particularly

important characteristic of agile methods is their subscription to an iterative mode of production. Instead of following an initial plan from start to finish (the classical “waterfall” style), the problem is broken down into minimal milestones which can each be evaluated when complete. This means that the project is continuously being tested and considered by all stake holders. The result of this is that development is given many opportunities to adapt in the case that problems arise, or if the program does not match user needs, or if those needs change, or any other such issue.

Applying computational techniques and tools in developing scientific software or to support the work of researchers in other disciplines calls for special attention to the unique qualities of a research environment. It is easy to see superficial similarities that the interdisciplinary research environment holds with a production software development environment. For example, in both cases there can be ongoing changes to what is being built, otherwise known as *requirements creep*. One could say that this is a central advantage of working with software: it can be redesigned and rebuilt without repercussions in the physical world, so it makes sense to take advantage of this flexibility. It is not surprising to see this characteristic in any endeavour that adopts software as a tool. Yet there are fundamental differences in who builds research software and what exactly is being constructed. Despite many varieties of development frameworks available today, they all assume a production model at their core: a product is developed for a client. There is a gap between current methods of software development and the needs of the research environment [10].

The communication problem is a recognized challenge for all software development projects [11, 12]. In interdisciplinary research, this is intensified because all team members are specialists. Researchers spend many years building up their domain expertise and cannot be expected to develop deep understanding of another field over a short period of time. By the very nature of research, the topics under investigation are cutting-edge. Finally, in order for the results of a project to be recognized, it is necessary to be able to communicate the workings of any computational elements to reviewers. For these reasons it is vital that there are methods for expressing the essence of an idea that allow for critical inspection, validation and modification.

Traditionally, a software development process is focused on producing a final product for an end-user. In the context of simulations of social systems, scientific research uses programs as experiments, to test theories and generate ideas that lead to new ones. Testing is useful at each stage of theory development, so a single project can generate a number of programs; they should be thought of as a set of related experiments. In science, the core methodology for discovery of new knowledge can be concisely described as the iteration of the hypothesis-experiment-result-conclusion cycle [13].<sup>1</sup> In the case of the social sciences, this includes exploratory techniques as well as classic inductive and deductive approaches. The programs developed for

---

<sup>1</sup> “A SCIENTIST, whether theorist or experimenter, puts forward statements, or systems of statements, and tests them step by step. In the field of empirical sciences, more particularly, he constructs hypotheses, or systems of theories, and tests them against experience by observation and experiment.” [13]

scientific research are computational experiments, and in order to accept the reality of iteration, it is necessary to envision the software developed for a research project as a set of related experiments. In order to relate to one another, they must share the same theoretical foundation. In other words, it is the conceptual schema that holds the continuity between programs, and not their functionality. However, the specific implementation of any program will be determined by the requirements of the experiment it embodies. Furthermore, since the software being developed is for the use of the team members themselves, the final configuration of that software is not a primary concern. The configuration of a program is only a concern to the extent that team members find it usable. These characteristics firmly shift the focus of the development cycle to the design or prototyping phase instead of implementation [14].

Central to all of these unique characteristics of the research environment is the underlying aim of scientific discovery. A completed program is not the final goal: software is used as a route to testing existing theories and creating new ones. Software is important because of the results experiments can provide. It can act as a sandbox where scientists are able to try out different ideas. It is also important because the transformation of theory into something computable captures the concepts being considered in a mathematical form. This mathematical model provides a blueprint for researchers to explain their ideas, or for peers to analyze its validity. It allows different projects to be compared to one another, and it acts as a foundation for further exploration of the subject matter. This process can be assisted by the adoption of current best coding practices [15]. Agile methods are well suited to research, particularly the importance placed on people and the idea of iterative development [11]. However, new techniques tailored specifically for the research environment can help to encourage software development that is more successful in generating new knowledge.

It is important for all researchers involved in the project to be able to understand and identify how the computational model works, or in other words, what is going on “under the hood”. This does not mean that all members should know programming, however, their understanding of the model will allow them to recognize issues that need to be addressed and either make the alterations directly (through a UI perhaps) or be able to communicate directly to the development members. This is especially important with an experimental project, since vital characteristics may change with each experiment, and may diverge significantly from the design agreed upon in initial phases. Clear representation is necessary for this, and graphical feedback can provide a satisfying way of achieving this. It is important to remember that the goal of any such graphical output is to communicate the behaviour of the program, and not to give an illusion of validity or to otherwise hide inner mechanisms. It also will make it easier to confirm the validity of the computational model. A program with obscure innards is useless to a researcher: they have no confidence in its results (it is just bells and whistles to them), and it cannot be used academically, because the source of any results it gives cannot be explained, so it is not convincing to peer readers.

## 4 Roles of Interdisciplinary Research Software

The roles computational modelling can play in a simulation modelling research initiative are outlined here. It is upon these points that the framework presented in the next section is premised.

**Reinforce Iterative Discovery** The executable nature of software means that it can be tested for various things such as accuracy and bugs whenever a change is introduced. Feedback from such testing can be used to iteratively improve either the program or the model itself. Of course, this is also possible with non-computational research, but software encourages this feedback loop through interactivity. To maintain this process, it is important for the mathematical model and computational models to remain flexible and capable of change.

**Visualization** Visual representation of data allows human users to examine and interpret using natural cognitive and pattern-matching capabilities, in addition to deductive reasoning. Ideally, a variety of visualizations should be available, allowing the user to alter their view of the data in order to consider different perspectives or subsets of the entire data set, which can be overwhelmingly rich in a social simulation.

**Formalization** Computational modelling forces researchers to precisely define the modelled elements of the subject phenomena so that it is possible to execute it on a computer. This process helps to frame existing domain knowledge in a mathematical manner. It also highlights areas which are poorly understood (which can become clear since they are difficult to formalize) and areas which are not important to the current research focus (which can become clear when defining them takes more effort than the value they add to the model).

**Testing Ideas** Computational models are ideal for testing hypotheses in a sandbox-like environment.

**Raising Questions** The modelling process can help raise questions that can be answered by more traditional research in the domain field. This can happen when information is missing to complete the design of the mathematical model, or if the computational model produces contradictory or conflicting results. In these cases, further literature review or discussion among experts may help to elucidate the missing information. It can also guide field research by suggesting questions to ask in future surveys.

**Demonstration** A computational model is valuable if it can illustrate domain concepts to non-experts. The dynamic qualities of a simulation combined with visualization capabilities combine to provide a powerful means of explaining complex phenomena in a straightforward manner.

**Communication of Ideas** As a formal model of a concept, the computational model is a proposed theory that can be used to communicate new ideas in an interactive manner among peers. Likewise, the model should be transparent enough that peers can examine it critically to point out problems, ask for clarification, or use the ideas in other research.

**Complex Calculation** A computational model is often a combination of many simple rules and concepts, but in aggregate the behaviour of such a model can easily

be beyond the comprehension or predictive capabilities of a human expert. In this way, a simulation program can act in the place of a human expert, correctly combining rules and concepts to produce aggregate (and sometimes emergent) results.

**Reproducibility** A hallmark of the scientific process, since it allows results to be validated independently of the original claimant. Computational models are reproducible in two senses: one, in a very literal sense, the simulation program can be distributed and executed by other people, demonstrating the results directly to them (or not, depending on their independent analysis); and two, in that aspects and elements of the proposed model can be taken out and used in other projects, or modelling ideas from other projects can be combined with or substituted into the proposed model [16]. The result of this latter point is that the model concepts can be tested and validated more widely. From a Popperian perspective, if the model concepts are transferrable and useful, they are more general and qualify for greater esteem and respect (at least until shown to be problematic)[13]. Reproducibility of specific runs can even be achieved for stochastic systems by recording the seed number of the random number generator.

## 5 Social System Computational Modelling Framework

Here, a high level description of the characteristics and goals of a framework for pursuing computational social science is presented. The framework includes both a process through which the model is developed, and a structure for the simulation software that closely aligns with this approach. Note that this does not refer to a specific piece of software, but rather the structure and features a simulation software should possess in order to fully support the model development process.

The agile software development paradigm is in general the most appropriate for working on computational models. The emphasis on flexibility to change is crucial here, since the level of understanding and current focus of interest is subject to ongoing change. Scientific fallibility means that we must be prepared for the discovery of errors in our model, and be able to try out promising alternatives when they appear. It is important to recognize the role of insight as both a driver and goal of working on this kind of project. When it comes to the model of the system, the attitude of “if it isn’t broken, don’t fix it” is not appropriate; instead it is a constant drive to improve something that is necessarily broken (to some extent). Another reason for an emphasis on agile methods is their focus on the role of human beings in the development process. Team members bring a variety of skills, knowledge, and needs to any project, and failing to take advantage of the good and/or deal with the difficult can be catastrophic in terms of success of the project. With these ideas in mind, some important characteristics of the software package used to implement the simulation system should be:

1. *Easy to change*, so that suggestions and requests for changes, as well as new ideas, can be implemented in a rapid manner. Obscure programming languages



are problematic here, or complicated libraries the code is dependent upon (e.g., for graphics).

2. *Easy to distribute*, so that each team member can easily remain up-to-date with regards to the latest build and test it out on any machine they have access to. Software that is difficult to compile or that requires specific operating systems or settings may be intimidating to some team members, and prevent adoption.
3. *Easy to use*, so that all team members feel comfortable using it to test out their ideas. This can be accomplished in part by simplifying the user interface and layout of the program. It can also be accomplished by identifying and maintaining an effective user interface, so that users are not forced to relearn the program frequently. Changes to the user interface are acceptable depending on the level of their contribution to the program.
4. *Easy to discuss*, so that the experiences and insight of users can be efficiently converted into improvements to the software. This requires a user-friendly system for reporting errors and discussing ideas. Time spent on establishing a shared vocabulary to describe important features and concepts is also valuable.

The structure of this framework is based on the widely accepted hypothetico-deductive model of the scientific method [17]. Generally, this is broken down into a series of stages applied iteratively. For example: observation, hypothesis, methodology, testing, and analysis. In principle, all stages of this process should be represented in a simulation software package, since this allows researchers to tighten the iterative cycle of experimentation. They should not have to waste time switching back and forth between software packages and data formats. Of course, they do not need to visit every step of the cycle each time: they may be uninterested in results analysis at an early stage of sensitivity analysis, where they are interested in parameter values and are still becoming familiar with the qualitative features of model behaviour.

Expanding upon the iterative computational model development process first presented in [18], the stages of the experimental process are:

**Conceptual Design** A general description of the features and expectations of the model. Relevant domain theory and ideas.

**Mathematical Model** Equations, formulas, and mathematical statements used to precisely define the structure of the model.

**Computational Model (Program Code)** The parts of the program related to the simulation model (not technical aspects like the GUI, visualization, etc.) should always be visible to users. Having it visible at all times would encourage the writing of clear code, and allow users familiar with the programming language to verify that the computational description of the simulation entities matches their expectations. It also encourages peers to check over the relevant code while testing out the software, without forcing them to sift through the entirety of the code base. A straightforward specification language (e.g., Abstract State Machines [19]) is ideal for this purpose.

**Experimental Design** This is where the initial state of a system and the duration of an experiment are specified. Combined with the user's expectations, this composes the scientific claim or hypothesis of this experiment.



**Playback** Here, the user can view the behaviour of the system as it passes through time during the simulation. They are capable of shuttling back and forwards through time to compare relevant states, such as the beginning, midpoint, and end. Options for viewing all simulated variables and entities should be accessible here.

**Visualization/Analysis** The data generated by the experiment is presented here for analysis. Currently, the changes in predetermined variables over time are plotted here, but this stage could benefit greatly from including a visualization suite (pre-existing or custom) that would give users the freedom to modify their view of the data as they consider it. In particular, it would be most useful if the visualization tool was capable of identifying all of the variables of the entities included in the simulation, so that data options at this stage would update automatically whenever the model is reprogrammed.

Each of these stages can be clearly included in the design of the software framework for working on social system simulations, for example, as a tab in software user interface. It may be a good idea to limit some users from changing some elements,

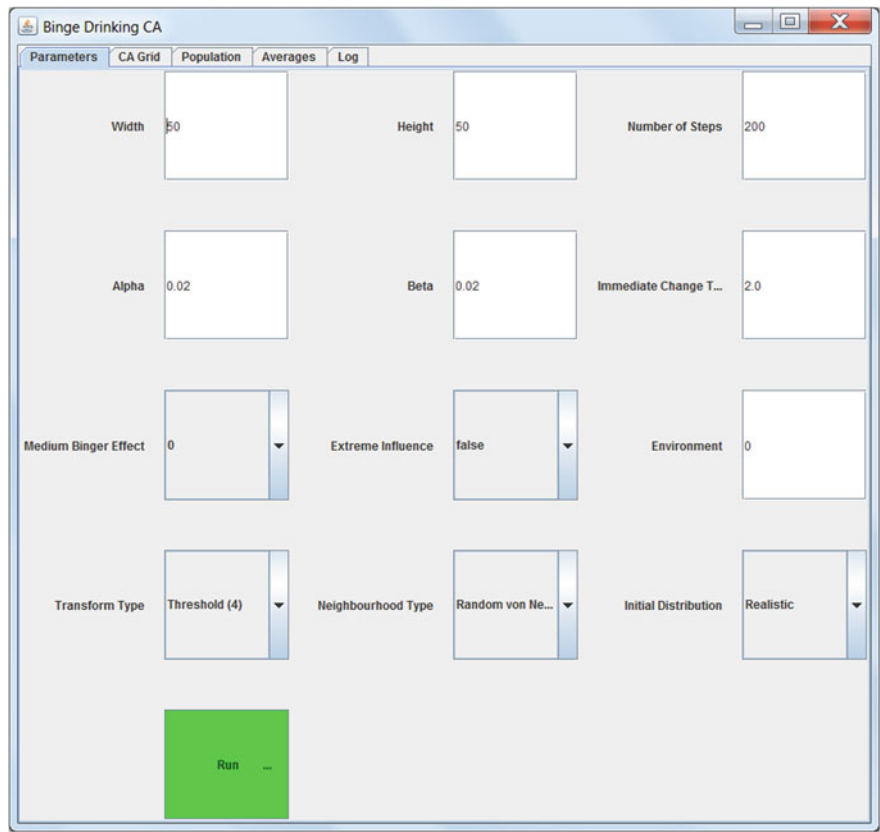


Fig. 1 Binge drinking cellular automata program: parameter set-up

particularly the model code, but it should still be available for them to check over. In the case of something like the conceptual model, a box of text is likely sufficient: it would contain expectations, ideas, and references to relevant literature (perhaps in the form of hyperlinks). Having this as a reference is useful for some or all of the people using the program, even if the project has moved beyond frequent updates of this aspect.

Figures 1, 2, and 3 show the Binge Drinking Cellular Automata program, a model of peer influence on the drinking behaviour of undergraduate students [20]. This model was one of several projects whose development led to the framework presented here [18, 21, 22]. As shown in the screenshot, the program interface has the following tabs (which correspond to the stage in parentheses): Parameters (experimental design), Simulation (playback), Population (visualization), and Averages (visualization). In this case, there is also a Log tab which contains system output (including random number generator seed values). It is important that the software reinforce good scientific practices, and remain a unifying theme across different

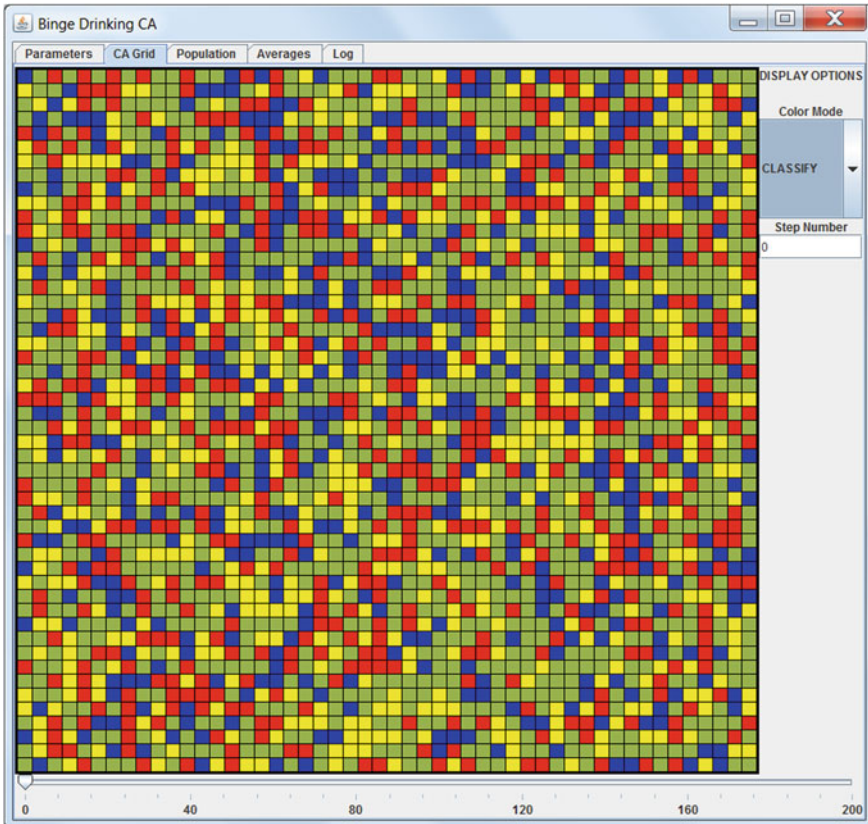
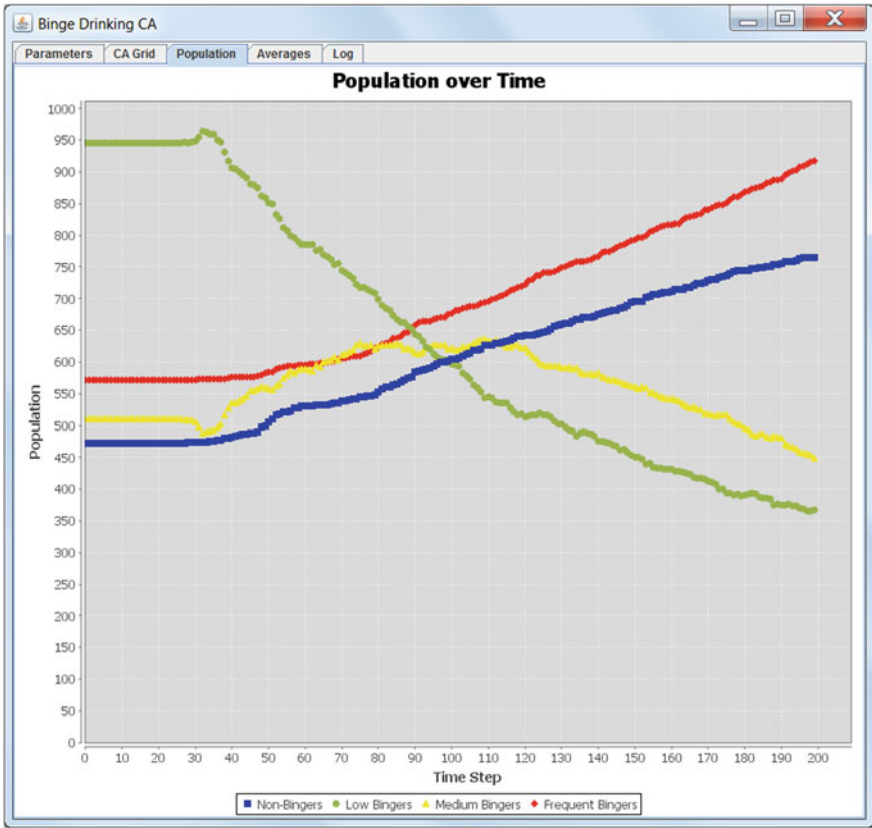


Fig. 2 Binge drinking cellular automata program: simulation playback



**Fig. 3** Binge drinking cellular automata program: population visualization

projects, ideally across different work groups, so that they may more easily understand each other’s work. Indeed, developing and maintaining a clear and accessible user interface will allow users to get to understand it better so that they are less intimidated by the software and are able to focus more of their attention on the computational model it is simulating.

## 6 Final Words

The framework presented here for the construction and development of computational models of social systems is the result of collaborative work on several computational modelling projects in the social sciences. It successfully brings the varied elements required for producing a computational model together in a way that emphasizes the integrity of the model and facilitates feedback-driven improvement. Perhaps

more importantly, general acceptance of a development framework by stakeholders in research and policy making institutions could be one of the factors that leads to computational modelling being more widely used as a tool for inquiry. It provides structure for the careful production and refinement of knowledge, and mechanisms for independently establishing the value of ideas and reusing them. There is still a great deal of work to be done before simulation is as widely used as, for example, statistics in research and decision making, but considering the potential for a method that exploits the algorithmic and mathematical nature of computing for the purpose of advancing science, it is a worthwhile goal to aim for.

## References

1. Hummon, N.P., Fararo, T.J.: The emergence of computational sociology. *J. Math. Sociol.* **20**(2–3), 79–87 (1995)
2. Gilbert, N.: Modeling sociality: the view from Europe. In: Kohler, T.A., Gummerman, G.J. (eds.) *Dynamics in Human and Primate Societies*, pp. 355–371. Oxford University Press, Oxford (2000)
3. Wing, J.M.: Computational thinking. *Commun. ACM* **4**(3), 33–35 (2006)
4. Ramanath, A.M., Gilbert, N.: The design of participatory agent-based social simulations. *J. Artif. Soc. Soc. Simul.* **7**(4), (2004)
5. Fortino, G., Russo, W.: Eldameth: an agent-oriented methodology for simulation-based prototyping of distributed agent systems. *Inf. Softw. Technol.* **54**(6), 608–624 (2012)
6. North, M.J., Collier, N.T., Vos, J.R.: Experiences creating three implementations of the repast agent modeling toolkit. *ACM Trans. Model. Comput. Simul.* **16**(1), 1–25 (2006)
7. Luke, S., Cioffi-Revilla, C., Panait, L., Sullivan, K., Balan, G.: Mason: a multiagent simulation environment. *Simulation* **81**(7), 517–527 (2005)
8. Gilbert, N., Terna, P.: How to build and use agent-based models in social science. *Mind Soc.* **1**, 57–72 (2000)
9. Leombruni, R., Richiardi, M., Revelli, L., Studies, C.E., Real, V.: A common protocol for agent-based social simulation. *J. Artif. Soc. Soc. Simul.* **9**(1), (2006)
10. Kelly, D.F.: A software chasm: software engineering and scientific computing. *IEEE Softw.* **24**, 119–120 (2007)
11. Fowler, M.: The new methodology. <http://martinfowler.com/articles/newMethodology.html> (2003)
12. Berry, D.M.: Formal methods: the very idea—some thoughts about why they work when they work. *Sci. Comput. Program.* **42**(1), 11–27 (2002)
13. Popper, K.: *The Logic of Scientific Discovery*. Hutchinson Education, London (1959)
14. Rickett, C.D., Choi, S., Rasmussen, C.E., Sottile, M.J.: Rapid prototyping frameworks for developing scientific applications: a case study. *J. Supercomputing* **36**, 123–134 (May 2006)
15. Baxter, S.M., Day, S.W., Fetrow, J.S., Reisinger, S.J.: Scientific software development is not an oxymoron. *PLoS Comput. Biol.* **2**, e87 (2006)
16. Buckheit, J., Buckheit, J.B., Donoho, D.L., Donoho, D.L.: *Wavelab and Reproducible Research*, pp. 55–81. Springer-Verlag (1995)
17. Godfrey-Smith, P.: *Theory and Reality: An Introduction to the Philosophy of Science*. University of Chicago Press, Chicago (2003)
18. Brantingham, P., Glaser, U., Jackson, P., Vajihollahi, M.: Modeling criminal activity in urban landscapes. In: Memon, N., David Farley, J., Hicks, D.L., Rosenorn, T. (eds.) *Mathematical Methods in Counterterrorism*, pp. 9–31. Springer, Vienna (2009)
19. Börger, E., Stärk, R.F.: *Abstract State Machines. A Method for High-Level System Design and Analysis*. Springer, Berlin (2003)

20. Jackson, P., Reid, A., Huitson, N., Wuschke, K., Dabbaghian, V.: Drinking with friends: a cellular automata approach to modeling peer influence of on binge drinking behaviour. In: International Symposium on Cellular Automata Modeling for Urban and Spatial Systems (CAMUSS), pp. 147–157 (2012)
21. Dabbaghian, V., Jackson, P., Spicer, V., Wuschke, K.: A cellular automata model on residential migration in response to neighborhood social dynamics. *Math. Comput. Model.* **52**(9–10), 1752–1762 (2010)
22. Pratt, S., Giabbanelli, P., Jackson, P., Mago, V.: Rebel with many causes: a computational model of insurgency. In: IEEE International Conference on Intelligence and Security Informatics, pp. 90–95 (2012)

Theories and Simulations of Complex Social Systems

Dabbaghian, V.; Mago, V.K. (Eds.)

2014, X, 204 p. 72 illus., Hardcover

ISBN: 978-3-642-39148-4