Software Engineering as Research Method: Aligning Roles in Econ-ARK

Sebastian Benthall[§‡*], Mridul Seth[‡]

https://youtu.be/nxXr0LNdQUU

Abstract—While general purpose scientific software has enjoyed great success in industry and academia, domain specific scientific software has not yet become well-established in many disciplines where it has potential. Based on a survey of the literature as well as the authors' experiences contributing to Econ-ARK, a structural modeling toolkit for Economics, we argue that this is due to the well-documented skills gap that prevents researchers, publishers, and professors from making the most of the opportunities afforded by scientific software. When researchers professionalize their code, it enables more cumulative progress in research and facilitates technology transfer. When publishers release interactive computational artifacts, it enables constructionist learning of the material. When students are trained in software engineering, they can participate fully in the reproduction of their scientific field. This is especially the case for fields where scientific knowledge is represented in software code, as in the case of Economics. The skills gap will not be closed until software engineering is considered a core skill for the discipline. Software engineering should be reconceived as a research method.

Index Terms—computational method, computional thinking, constructionist learning, research software engineering

Computing in Education and Science

Ever since [Pap82] introduced constructionist learning using computers, educators have been enticed by the possibility that students could learn valuable knowledge by playing with software. While originally used as a tool for teaching mathematics, it was not long before Papert's Logo tool was also used in scientific education, teaching students not just about the abstract mathematical sphere, but about the physical world [ROP90]. The legacy of Logo is alive and well in NetLogo [TW04], which is used by both students and researchers alike in the study of complex and agent-based systems, and in the Python agent-based modeling (ABM) toolkit Mesa [DMJK15].[1]

Since, the ubiquity of computing and its increasingly central role in industry has prompted the spread of ideas that were once specific to computer science into other disciplines. [Win06] coined the term "computational thinking" for the general skills of managing abstraction, modularity, scalability, and robustness of systems. Now it refers to the cross-disciplinary use of these computational concepts [Guz08] [SGB13]. The question raised by computational thinking is how much computer science education is necessary for these cross-disciplinary uses of computation. Logo, after all, not only introduced students to mathematics, but also programming. But did it teach computational thinking?

The industrial demand for students educated in handling "Big Data" systems has since prompted a generalization of statistics beyond its discipline in a way that's analogous to the generalization of computer science. [Jor16] discusses this new industry demand for "inferential thinking". Together, computational thinking and inferential thinking have been reimagined by some as the foundation for a new form of cross-disciplinary data science curriculum [AD17] [EVDSLB19]. A key technological feature of these new curricula are digital notebooks that enable users to compose computational narratives that make computing more cognitively digestible to humans [PG15]. Now, Jupyter notebooks are widely used for collaboration on research and, in some places, as part of pedagogy.

Open source scientific software development has benefited from the influx of capital due to industry interest in data science applications. Software packages such as Numpy [WCV11], Pandas [McK11], and Scikit-learn [PVG+11] have become popular as industrial tools. At the same time, these tools have provided a foundation and aspirational example for more domain specific scientific libraries, such as astropy [RTG+13], Biopython [CAC+09], PsychoPy [Pei07], and SunPy [MPSC+13]. Scientific educators continue to see potential in the use of these tools to support the education of their students not only *about computation*, but *about the world* [Bar16], in a return to Papert's constructionist paradigm.

This vision of scientific research and education supported by open source domain specific scientific libraries faces two significant obstacles. The first is the development and sustainability of the software itself. Open source software projects in general are not guaranteed to succeed; most fail to gain wide adoption or reach sustainability [SE12]. In addition to these general difficulties, scientific software suffers from the fact that researchers who write and modify software often do not have formal training in software development. As a result, scientific software is often hampered by technical debt. These problems are mitigated by national initiatives to train scientists in software engineering skills, such as the UK's Software Sustainability Institute, as well as Software Carpentry [Wil14]. There is further work to be done in institutional design around filling this skills gap [KCN+16]. But it is known that computational thinking skills alone are not sufficient for successful

* *Corresponding author: spb413@nyu.edu*
§ *New York University School of Law*
‡ *Econ-ARK*

1. An example of an ABM is the Wolf Sheep Predation model, which is used to explore the stability of predator-prey ecosystems [Wil97].

scientific software. Software engineering skills are necessary to produce software that is usable beyond the lab or research group that originates it, which is a necessary path towards software sustainability [Ben19].

A second obstacle integrating software tools into scientific practice is that software-based learning requires additional education infrastructure. [SNLT18] document the challenges in providing JupyterHub with automatic grading extensions at universities and colleges; they find that many institutions do not have the resources or deep IT expertise necessary to build and maintain this infrastructure. The growing necessity of cloud-based computational notebooks for assignments and exploration in scientific education therefore raises concerns about social equity.

This paper explores these general themes through an analysis of Econ-ARK [CKK+18] as a case study. Econ-ARK is a domain specific software toolkit currently most widely used in Economics. Launched in 2014, the project has recently experienced a phase transition in development practices because of the onboarding of research software engineers. The collaborations between Economics professors and software engineers have revealed a broad scope of potential in computational research, publication, and pedagogy. It has also exposed how disciplinary training in Economics does not include many concepts necessary to realizing that potential. We conclude that the gaps between disciplinary training and the conditions for realizing this potential can be partially closed by framing software engineering as a research method.

Econ-ARK: Discipline Specifics

The Econ-ARK project [CKK+18] is a toolkit for the structural modeling of optimizing economic choices by heterogenous agents. A primary goal of its flagship software library HARK (Heterogenous Agent Research toolKit) is to support economic research into heterogeneous agent (HA) modeling [Hom06], which became a research priority after the 2008 financial crisis revealed the weaknesses in the then-dominant representative agent (RA) based paradigm.[2] It has been designed so that researchers and students can take a hands-on approach to economic modeling in software [CW18]. Econ-ARK is in some respects a port of Dynare [ABJ+11], an earlier computing library for economic models, into Python.

Econ-ARK lies roughly in the Papertian educational tradition, similar to other agent-based modeling software such as NetLogo [TW04] and Mesa [DMJK15]. However, in Econ-ARK models, agents optimize their behavior strategically with respect to predicted effects over time. In this respect, Econ-ARK has some characteristics of a reinforcement learning or artificial intelligence toolkit.

Example. A paradigmatic, simple example of the kind of problem studied using Econ-ARK is the microeconomic dynamic stochastic optimization problem of calculating the mathematically optimal amount to save [Car11].

This problem can be characterized by the equations:

$$\begin{aligned} U(c_t) &= \frac{c_t^{1-\rho}}{1-\rho} \\ m_{t+1} &= R(m_t - c_t) + p_{t+1} \\ p_{t+1} &= \gamma p_t \end{aligned}$$

where U is a utility function, ρ is a coefficient of risk aversion, c_t is the amount of resources the agent chooses to consume in each period t, m_t is the amount of market resources available to the agent at each time period, p_t is the level of income at each time period, γ is the growth rate of income over time, and R is a rate of return on savings.

These equations define a Markov Decision Problem (MDP), which can be transformed into a Bellman equation given a discount factor β:

$$V_t(m_t, p_t) = max_{c_t} U(c_t) + \beta V_{t+1}(m_{t+1}, p_{t+1})$$

The optimal consumer choice can be solved via dynamic programming.

However, it is possible to reduce the complexity of this problem significantly through mathematical analysis. Because income is growing geometrically, it is possible to remove one of the state variables p from the model, and solve for the MDP with the following transition function:

$$m_{t+1} = \frac{R}{\gamma}(m_t - \hat{c}_t) + 1$$

The consumption function $\hat{c}$ can then be solved in a reduced (1-dimensional) state space. The optimal consumption function for the original problem is then recoverable as $c_t = \hat{c}_t * p_t$. It is the goal of the Econ-ARK software to bundle the analytically reduced solution with the original model as a way of representing and making available the substantive knowledge gained in the mathematical derivation.

Models in HARK are, at a certain level of mathematical abstraction, equivalent to Markov Decision Problems (MDP). However, generic MDP software is not adequate for research in this field, for several reasons.

- **Substantive, policy-oriented structural modeling.** Unlike many recent fields of data science, in which generic model-fitting and machine-learning techniques are applied to a large data set for the purpose of maximizing predictive potential, this branch of Economics operates with relatively scarce data and a drive for model veracity. Besides the academic field of researchers, the intended audience for these models are national central banks and other policy-makers. For example, one policy application of these models is predicting the impact of the CARES stimulus bill on consumption [CCSW20]. These models are scientifically valued for their ability to approximate real social dynamics, and for their ability to build consensus towards policy-making, in addition to their goodness of fit to available data.
- **Analytical results informing solvers.** Like many other sciences, this branch of Economics has a theoretical component consisting in mathematical proofs about the models in question. In addition to providing interpretable insight into the invariant properties of a model, these results also inform the design of model solvers and the user experience. For example, a mathematical result might reveal under what parameter conditions a model has a degenerate solution; the software will warn the user if they attempt to solve the model in such a case. Elsewhere, an analytical result might provide a shortcut such that it is possible to write a solution algorithm with lower computational complexity than a generic one would have.
- **Continuous space decisions.** Most MDP solvers and simulators assume a discrete control and state space. The

2. These weaknesses had been known since the work of [Kir92].

economic problems studied using HARK are most often defined with continuous control and state spaces, and with continuous random variables as exogenous shocks. HARK therefore includes a variety of discretization and interpolation tools that support the transformation between discrete and continuous representations.

The upshot of these conditions is that Econ-ARK software is not only a tool for researchers doing empirical scientific work. Rather, its software is an encoding of substantive research results in mathematical theory. A software implementation, which integrates the results in a larger body of work and is subject to robust software testing, is an additional form of validation of the correctness and salience of a finding. This entails that the success of Econ-ARK will imply a practical change to the research field: students will study models that have been published in Python by researchers in order to learn insights about the economy.

Case Study: Roles in Econ-ARK

Econ-ARK has been broadly conceived as a collection of projects that supports this computational approach to education and research in economic structural modeling. The project has been organized around several different version-controlled software repositories. The software in these repositories is written mostly in Python, though there is also a great deal of expository content and sometimes older code in other languages such as MATLAB and Mathematica.

We have identified several different roles that people take on when interacting with Econ-ARK. The same individual or "natural person" might take on different roles at different times, but nevertheless these categories have been useful as ideal types [Hek83] with which to reason about requirements and skills.

Researcher. The role at the heart of the Econ-ARK system is that of the Researcher. This user is trying to advance the frontier of economic thinking by drawing on deep domain knowledge (Economics) as well as general training in computational and inferential thinking, applied math, and perhaps other fields. Research with Econ-ARK may be nebulously defined because while the question of how to implement a class of economic models efficiently and robustly in Python is a research question in its own right, these implementations are rarely considered first-order research contributions. Researchers work within a complex field of economic capital incentives (such as university salaries and grant funding) and symbolic capital incentives (scholar recognition for published work) [Bou04]. At the time of this article's publication, the institutional mechanisms for training and rewarding Economics researchers to work in the medium of robust software are few. As a consequence there is a skills gap: researchers often have programming ability, but not the software engineering and IT training that is necessary to fully realize the vision of the software's potential [CHH+13].

Publisher. One way to untie the Gordian knot of incentives around Econ-ARK research is to provide a more reliable and efficent path towards recognized scholarly publication that uses it. One proposal has been that economists begin a Journal of Open Source Economics [Isk19], modeled loosely on the Journal of Open Source Software (JOSS), which gives academic publication credit to the creators of scientific software tools. Preliminary efforts towards such a journal have been attempted through the Econ-ARK sub-project REMARK (Replications and Explorations Made using the ARK), which organizes contributed directories of material that meet a minimal 'publishable' standard of reproducibility. This approach has surfaced many challenges, mainly regarding the technical requirements of reliably hosting Python environments for each publishable unit, and managing dependencies across those environments. These technical challenges of *publication* require IT skills that are in general not available to researchers who may be technically capable of programming models that show substantive academic results.

Teacher and Student. In an academic context, the pedagogical use case is as important as the researcher's use case. While the researcher is building new models to communicate new discoveries, the teacher guides students to learn skills and ideas that are already known. Two of the hurdles faced by teachers attempting to use Econ-ARK pedagogically are the creation and grading of assignments and assisting students with the availability of an adequate computing environment that does not distract them from the course materials. Technical solutions have been developed for both hurdles. *nbgrader* enables the creation of assignments with Jupyter notebooks [Ham16] [BBB+19]. JupyterHub has been deployed to allow students to get around the hardware limitations of their laptops and the difficulties of setting up a local coding environment [Kim18]. Notably, both technical solutions, which have been developed only in the past few years, require skills that are not part of normal disciplinary training in economics. Economics professors currently require others to fill the social role that enables these tools to be useful.

Software engineer. The elephant in the room in all discussions of scientific software and computational education is that building and deploying robust software is its own complex field that often shares few disciplinary roots with the domain sciences. These skills are often specific to technologies that originated in industry or open source technology production, not in academia. For example, the version control system Git was not originally an academic project, but it nevertheless is now ubiquitously used for computational academic research through its popularization via GitHub. The workflow patterns of collaboratively developing software using GitHub and managing release cycles are not part of any conventional Economics curriculum, and yet researchers increasingly need to learn and use these in order to participate in computational research. Software engineering skills are not only useful for these infrastructural requirements of publication and pedagogy. Integrating new features, expressing substantive disciplinary material, and making these features available for new users requires these skills. In other words, software engineering skills are required to make a software project robust and reusable across many different labs and groups of researchers [Ben19]. This has led to calls in some places for a better supported and formalized role for Research Software Engineers [PHH16] [BHG+12].

This division of roles and skills raises some quandaries for computational economics. Publication, pedagogy, and the sustainability of the domain specific software library Econ-ARK all require software engineering skills. But there is no point at which new entrants into this discipline are trained in these skills. They must be learned informally by researchers who are not incentivized to do so, or they must be hired from an external talent pool trained in other disciplines or at another workplace.

This interrupts the cycle, from student to researcher to professor who teaches more students, which is necessary for the autonomy of Economics as a field of knowledge. If at every point in the process -- even at the point where new discoveries are

integrated into the core software library -- there is a dependence on an externally sourced skillset, then the discipline will fail to reproduce scholars with the competence to participate in its own field.

Case Study: Econ-ARK infrastructure

The Econ-ARK infrastructure is built around creating a sustainable community with respect to various use cases and the challenges of creating sustainable scientific software in Economics. We have discussed some of the challenges of bridging work across user roles of Researchers, Publishers, Professors and Software Engineers. Here. we illustrate these general points with examples from our software and infrastructure practices.

Decoupling scientific content from code. A lot of scientific code is written as part of academic research projects where the incentives aren't closely aligned with those of creating scientific software. The recent case of UK COVID microsimulation code [MRC] brings out a stronger need of creating scientific software with the correct incentives. The decision to draw the line between a research artifact and a software is a hard decision which varies a lot between different scientific domains and requires a high level overlap of the researcher, publisher and software engineer roles.

When scientific code written by researchers is geared towards the publishable end result like a paper, it can lead to short-sighted design choices that in a broader software context are known as "technical debt" [KNO12]. An illustration is this example of a difference between a script and a modular function [Sci].

```
# a research project to calculate the moving
# averages of two stocks

import pandas as pd

data = pd.read_csv('stocks_data.csv')

x = data['APPL'].rolling(window=5).mean()
y = data['GOOG'].rolling(window=5).mean()

print(x, y)
```

Running this script prints out the moving average time series of the two stocks. We can also create a software package which achieves the similar thing in a more modular way.

```
# move_avg.py

import pandas as pd

def calculate_MA(data, stock, days):
  # Calculates the moving average for a stock
  return data[stock].rolling(window=days).mean()
```

We can achieve similar results using our new package *move_avg*, but this isn't restricted to our specific hard coded variables (number of days, stock, input data).

```
import pandas as pd
from move_avg import calculate_MA

data = pd.read_csv('stocks_data.csv')
print(calculate_MA(data, 'APPL', 5))
print(calculate_MA(data, 'GOOG', 5))
```

Initial decisions like hard coding variables in the code while creating the research artifact (which happens in a lot of academic research projects) lead away from creating a well defined reusable scientific software library. This seems trivial for people with a software engineering background but not necessarily for others. We know this is a hard problem to solve in domain specific scientific code where the boundaries between a research paper and code could be blurry. To tackle this is Econ-ARK, we extracted generalized code from research artifacts to create our software package HARK [CKK+18] and maintained the research artifacts which heavily rely on HARK as REMARKs (Replications and Explorations Made using the ARK).

This decoupling exercise also helps with the reproducibility of research projects as it gives other researchers the necessary tools to examine the research artifacts. The decoupling can also enable the use of empirical data and model fitting techniques, expanding the functional scope of the original script.

Reproducible builds of scientific content. The reproducibility crisis has been plaguing academic research for some time and the current ecosystem of software packaging and distribution certainly does not help it. To tackle this in Econ-ARK we have used containerization technologies like Docker. Tools like Repo2Docker [Jup] further help us with creating reproducible builds of scientific content. Creating and working with these tools still requires a basic background with software engineering, and end users like students and researchers in economics may not have the required background. We made tools to lower the barrier by using pre-built containers and one-click (or one-command) reproducible research artifacts [EA]. This effort has required a strong overlap between Researchers and Software Engineers in a project. Pushing for reproducibility in the community benefits students by lowering the barriers to access research and publishers/researchers by creating tools required to address the reproducibility crisis.

Interactive scientific publication. The publication of the Econ-ARK-based analysis of the consumption response to the CARES Act [CCSW20] was accompanied by an online Dashboard[3] that allows users to change parameters of the model and visualize their impact on policy outcomes. This Dashboard was deployed using Binder and developed by an Econ-ARK Research Software Engineer. This dashboard supports the constructionist learning of the substance of the model. Here, that paradigm is applied to convey knowledge not to students, but to public policy makers and other economists.

This new way of presenting economic models may be more digestible to a wider audience than a traditional research publication. However, researchers are not trained to create these Dashboards as they are trained to write research papers. This limits the scholarly impact of domain specific research software, as many computational models are not being presented in this rich interactive way.

Teaching resources. To keep the wheels turning in a research discipline we require effective pedagogical resources, especially in domains which are increasingly using scientific software to further research. After creating pedagogical content we are faced with the next hard challenge of creating an effective teaching infrastructure. The crème de la crème of the SciPy community has faced installation problems with software packages and it is not hard to create a monster out of your local environment. But luckily tools like MyBinder and JupyterHub have drastically reduced the work required to set up a stable environment required for teaching courses that depend heavily on scientific software. At Econ-ARK we have used MyBinder (publicly and privately

3. https://mybinder.org/v2/gh/econ-ark/Pandemic/master?urlpath=voila%2Frender%2FCode%2FPython%2Fdashboard.ipynb

hosted) extensively for teaching graduate economics courses and it has significantly reduced the overhead required for local setup, especially for students who are the primary users of a domain specific scientific software like HARK. We have also effectively used containerization for standardizing student assignments which streamlines the work for both students and teachers.

Discussion

Is research software engineering becoming a core skill for research that involves writing code? The skills for navigating many practical elements of software engineering are necessary for equipping a digital classroom, effectively publishing results, and contributing new features to scientific libraries. Yet they are currently considered a peripheral part of disciplinary education in Economics. Researchers and professors are not taught these skills as part of their training as students. This contributes to a systemic skills gap between the discipline and technology.

One potential solution to this problem would be to introduce more software engineering training into the core curriculum for graduate students. Some Economics departments already offer a course on Computational Methods, analogous to earlier courses on Mathematical Methods, Econometrics, or other methods. As the pragmatic needs of computational methods increasingly require such activities as setting up local development environments, preparing cloud computing infrastructure, and utilizing autodocumentation, version control and package management tools, these techniques could be included as part of a computational methods curriculum.

This is a departure from both the computational thinking [Win06] approach, which emphasizes abstract, conceptual skills explicitly in contrast to the mechanical skills of programming, let alone software engineering. It is also a departure from constructionist learning [Pap82], in that the method of learning is not childlike play but what is instead most often considered a form of laborious work. Rather, it is perhaps best conceived and taught in the paradigm of situated learning [LW91], or an apprenticeship based model. In this model, students engage in "legitimate peripheral participation" by working with tools under the mentorship of experts, gradually becoming more central in the community of practice. This model has been applied to both software engineering education and open source community participation [YK03].

Preparing scientists with more general software engineering skills would pave the way for more general acceptance of computational narrative [PG15] as a core method in scientific practice. In the social sciences especially, this would open research fields to wider ranges of discoveries through computational methods. [Eps06] has argued that computational modeling in social science is the natural successor to game theoretic and rational choice modeling, which has a long social scientific history, allowing a wider range of models with greater realism and theoretical insight. While [Hom06] and [Tes06] have shown the applicability of these methods to economics in particular, progress has been limited by the lack of research software engineering skills available in the field. To unlock the potential of computational science, research software engineering must become recognized as a research method.

Another incentive for making software engineering more central as a research method for scientific practice is that mature software products are a vector for technology transfer from academic labs to the market [DR04]. As national funding agencies anticipate a pivot towards bringing scientific results to market a top priority [Amb20] it raises questions about what research methods are most commercially relevant.

We are definitely not the first push for more training to scientific researchers about general software design and best practices (software versioning, continuous integration, testing). Organizations like Software Carpentry [Wil14] have been successful in this domain. Creating sustainable domain specific scientific software requires a systematic decoupling of reusable library code from research artifacts so users from different backgrounds can successfully work with the software. Researchers writing code with knowledge about software design will have more success in creating a sustainable community. Our contribution in this paper is to discuss how software design can be reconceived as a scientific method, as opposed to a peripheral skill.

References

[ABJ+11] Stéphane Adjemian, Houtan Bastani, Michel Juillard, Ferhat Mihoubi, George Perendia, Marco Ratto, and Sébastien Villemot. Dynare: Reference manual, version 4. 2011.

[AD17] Ani Adhikari and John DeNero. Computational and Inferential Thinking: The Foundations of Data Science, 2017. URL: https://www.inferentialthinking.com/.

[Amb20] Mitch Ambrose. Lawmakers propose dramatic expansion of nsf to boost us technology, May 2020. URL: https://www.aip.org/fyi/2020/lawmakers-propose-dramatic-expansion-nsf-boost-us-technology.

[Bar16] Lorena A Barba. Computational thinking: I do not think it means what you think it means, 2016. URL: https://lorenabarba.com/blog/computational-thinking-i-do-not-think-it-means-what-you-think-it-means/.

[BBB+19] Douglas S Blank, David Bourgin, Alexander Brown, Matthias Bussonnier, Jonathan Frederic, Brian Granger, Thomas L Griffiths, Jessica Hamrick, Kyle Kelley, M Pacer, et al. nbgrader: A tool for creating and grading assignments in the jupyter notebook. *The Journal of Open Source Education*, 2(11), 2019. doi:10.21105/jose.00032.

[Ben19] Sebastian Benthall. Software incubator workshop: A synthesis, Feb 2019. URL: http://urssi.us/blog/2019/02/25/software-incubator-workshop-a-synthesis/.

[BHG+12] Rob Baxter, N Chue Hong, Dirk Gorissen, James Hetherington, and Ilian Todorov. The research software engineer. In *Digital Research Conference, Oxford*, pages 1–3, 2012.

[Bou04] Pierre Bourdieu. *Science of science and reflexivity*. Polity, 2004.

[CAC+09] Peter JA Cock, Tiago Antao, Jeffrey T Chang, Brad A Chapman, Cymon J Cox, Andrew Dalke, Iddo Friedberg, Thomas Hamelryck, Frank Kauff, Bartek Wilczynski, et al. Biopython: freely available python tools for computational molecular biology and bioinformatics. *Bioinformatics*, 25(11):1422–1423, 2009. doi:10.1093/bioinformatics/btp163.

[Car11] Christopher D Carroll. Solution methods for microeconomic dynamic stochastic optimization problems, 2011. URL: http://www.econ.jhu.edu/people/ccarroll/solvingmicrodsops.pdf.

[CCSW20] Christopher D Carroll, Edmund Crawley, Jiri Slacalek, and Matthew N White. Modeling the consumption response to the cares act. *COVID Economics*, 2020.

[CHH+13] Stephen Crouch, Neil Chue Hong, Simon Hettrick, Mike Jackson, Aleksandra Pawlik, Shoaib Sufi, Les Carr, David De Roure, Carole Goble, and Mark Parsons. The software sustainability institute: changing research software attitudes and practices. *Computing in Science & Engineering*, 15(6):74–80, 2013. doi:10.1109/MCSE.2013.133.

[CKK+18] Christopher D. Carroll, Alexander M. Kaufman, Jacqueline L. Kazil, Nathan M. Palmer, and Matthew N. White. The Econ-ARK and HARK: Open Source Tools for Computational Economics. In Fatih Akici, David Lippa, Dillon Niederhut, and M Pacer, editors, *Proceedings of the 17th Python in Science Conference*, pages 25 – 30, 2018. doi:10.25080/Majora-4af1f417-004.

[CW18] Christopher D Carroll and Matthew N White. Hands-on heterogeneous agent macroeconomics, 2018. URL: https://safe-frankfurt.de/fileadmin/user_upload/editor_common/Events/Chris_Carrol_Syllabus.pdf.

[DMJK15] David Masad and Jacqueline Kazil. Mesa: An Agent-Based Modeling Framework. In Kathryn Huff and James Bergstra, editors, *Proceedings of the 14th Python in Science Conference*, pages 51 – 58, 2015. doi:10.25080/Majora-7b98e3ed-009.

[DR04] Jean-Michel Dalle and Guillaume Rousseau. Toward collaborative open-source technology transfer. *Collaboration, Conflict and Control*, pages 34–42, 2004. doi:10.1049/ic:20040262.

[EA] Econ-ARK. BufferStockTheory reproduce. URL: https://github.com/llorracc/BufferStockTheory/blob/master/reproduce.sh.

[Eps06] Joshua M Epstein. *Generative social science: Studies in agent-based computational modeling*. Princeton University Press, 2006. doi:10.23943/princeton/9780691158884.001.0001.

[EVDSLB19] Eric Van Dusen, Anthony Suen, Alan Liang, and Amal Bhatnagar. Accelerating the Advancement of Data Science Education. In Chris Calloway, David Lippa, Dillon Niederhut, and David Shupe, editors, *Proceedings of the 18th Python in Science Conference*, pages 1 – 4, 2019. doi:10.25080/Majora-7ddc1dd1-000.

[Guz08] Mark Guzdial. Education paving the way for computational thinking. *Communications of the ACM*, 51(8):25–27, 2008. doi:10.1145/1378704.1378731.

[Ham16] Jessica B Hamrick. Creating and grading ipython/jupyter notebook assignments with nbgrader. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, pages 242–242, 2016. doi:10.1145/2839509.2850507.

[Hek83] Susan J Hekman. Weber's ideal type: A contemporary reassessment. *Polity*, 16(1):119–137, 1983. doi:10.2307/3234525.

[Hom06] Cars H Hommes. Heterogeneous agent models in economics and finance. *Handbook of Computational Economics*, 2:1109–1186, 2006. doi:10.1016/s1574-0021(05)02023-x.

[Isk19] Fedor Iskhakov. The journal of open source economics journal charter, 2019. URL: https://github.com/joseconomics/JOSEcon-Project-Charter/blob/master/josecon_charter.pdf.

[Jor16] Michael I Jordan. On computational thinking, inferential thinking and data science. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 47–47, 2016. doi:10.1145/2935764.2935826.

[Jup] Jupyter. repo2docker. URL: https://github.com/jupyter/repo2docker.

[KCN+16] Daniel Katz, Sou-Cheng Choi, Kyle Niemeyer, James Hetherington, Frank Löffler, Dan Gunter, Ray Idaszak, Steven Brandt, Mark Miller, Sandra Gessing, et al. Report on the third workshop on sustainable software for science: Practice and experiences (wssspe3). *Journal of Open Research Software*, 4(1), 2016. doi:10.5334/jors.118.

[Kim18] Alicia Kim. The jupyterhub journey: Starting small and scaling up, May 2018. URL: https://data.berkeley.edu/news/jupyterhub-journey-starting-small-and-scaling.

[Kir92] Alan P Kirman. Whom or what does the representative individual represent? *Journal of economic perspectives*, 6(2):117–136, 1992.

[KNO12] Philippe Kruchten, Robert L Nord, and Ipek Ozkaya. Technical debt: From metaphor to theory and practice. *Ieee software*, 29(6):18–21, 2012. doi:10.1109/ms.2012.167.

[LW91] Jean Lave and Etienne Wenger. *Situated learning: Legitimate peripheral participation*. Cambridge University Press, 1991. doi:10.1017/cbo9780511815355.

[McK11] Wes McKinney. pandas: a foundational python library for data analysis and statistics. *Python for High Performance and Scientific Computing*, 14(9), 2011.

[MPSC+13] Stuart Mumford, David Pérez-Suárez, Steven Christe, Florian Mayer, and Russell J. Hewett. SunPy: Python for Solar Physicists. In Stéfan van der Walt, Jarrod Millman, and Katy Huff, editors, *Proceedings of the 12th Python in Science Conference*, pages 70 – 73, 2013. doi:10.25080/Majora-8b375195-00c.

[MRC] UK MRC. Covidsim microsimulation model. URL: https://github.com/mrc-ide/covid-sim.

[Pap82] Seymour Papert. Mindstorms. *NY: Basic Books*, 1982. doi:10.1007/978-3-0348-5357-6.

[Pei07] Jonathan W Peirce. Psychopy—psychophysics software in python. *Journal of Neuroscience Methods*, 162(1-2):8–13, 2007. doi:10.1016/j.jneumeth.2006.11.017.

[PG15] Fernando Perez and Brian E Granger. Project jupyter: Computational narratives as the engine of collaborative data science, 2015. URL: http://archive.ipython.org/JupyterGrantNarrative-2015.pdf.

[PHH16] Olivier Philippe, Neil Chue Hong, and Simon Hettrick. Preliminary analysis of a survey of uk research software engineers. In *4th Workshop on Sustainable Software for Science: Practice and Experience*, 2016.

[PVG+11] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research*, 12:2825–2830, 2011.

[ROP90] Mitchel Resnick, Stephen Ocko, and Seymour Papert. *LEGO/logo–learning through and about design*. Epistemology and Learning Group, MIT Media Laboratory Cambridge, MA, 1990.

[RTG+13] Thomas P Robitaille, Erik J Tollerud, Perry Greenfield, Michael Droettboom, Erik Bray, Tom Aldcroft, Matt Davis, Adam Ginsburg, Adrian M Price-Whelan, Wolfgang E Kerzendorf, et al. Astropy: A community python package for astronomy. *Astronomy & Astrophysics*, 558:A33, 2013.

[Sci] SciPy. Scipy lecture notes. URL: https://scipy-lectures.org/intro/language/reusing_code.html.

[SE12] Charles M Schweik and Robert C English. *Internet success: a study of open-source software commons*. MIT Press, 2012. doi:10.7551/mitpress/9780262017251.001.0001.

[SGB13] Amber Settle, Debra S Goldberg, and Valerie Barr. Beyond computer science: computational thinking across disciplines. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education*, pages 311–312, 2013. doi:10.1145/2462476.2462511.

[SNLT18] Anthony Suen, Laura Norén, Alan Liang, and Andrea Tu. Equity, Scalability, and Sustainability of Data Science Infrastructure. In Fatih Akici, David Lippa, Dillon Niederhut, and M Pacer, editors, *Proceedings of the 17th Python in Science Conference*, pages 15 – 17, 2018. doi:10.25080/Majora-4af1f417-002.

[Tes06] Leigh Tesfatsion. Agent-based computational economics: A constructive approach to economic theory. *Handbook of Computational Economics*, 2:831–880, 2006. doi:10.1016/s1574-0021(05)02016-2.

[TW04] Seth Tisue and Uri Wilensky. Netlogo: Design and implementation of a multi-agent modeling environment. In *Proceedings of the Agent 2004 Conference on Social Dynamics: Interaction, Reflexivity and Emergence*, volume 2004, pages 7–9, 2004.

[WCV11] Stéfan van der Walt, S Chris Colbert, and Gael Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011. doi:10.1109/mcse.2011.37.

[Wil97] Uri Wilensky. Netlogo wolf sheep predation model, 1997. URL: http://ccl.northwestern.edu/netlogo/models/WolfSheepPredation.

[Wil14] Greg Wilson. Software carpentry: lessons learned. *F1000Research*, 3, 2014. doi:10.12688/f1000research.3-62.v2.

[Win06] Jeannette M Wing. Computational thinking. *Communications of the ACM*, 49(3):33–35, 2006. doi:10.1145/1118178.1118215.

[YK03] Yunwen Ye and Kouichi Kishida. Toward an understanding of the motivation of open source software developers. In *Proceedings of the 25th International Conference on Software Engineering, 2003*, pages 419–429. IEEE, 2003. doi:10.1109/icse.2003.1201220.