Platformizing Higher Education: Computer Science and the Making of MOOC Infrastructures

by

Shreeharsh Kelkar

B.E. Electronics Engineering
University of Mumbai, 2002

M.S. Electrical Engineering
Columbia University, 2004

Submitted to the Program in Science, Technology, and Society in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in History, Anthropology, and Science, Technology, and Society

at the
Massachusetts Institute of Technology

September 2016

© 2016 Shreeharsh Kelkar. All Rights Reserved.

The author hereby grants to MIT permission to reproduce and to distribute publicly paper and electronic copies of this thesis document in whole or in part in any medium now known or hereafter created.

Signature redacted

Signature redacted

Certified by: Graham Jones
Associate Professor of Anthropology
Committee Chair

Signature redacted

Certified by: Susan Silbey
Leon and Anne Goldberg Professor of Humanities, Sociology and Anthropology
Professor of Behavioral and Policy Sciences, Sloan School of Management
Thesis Committee Member

Signature redacted
Certified by: ____________________________________________

Jennifer S. Light
Professor of Science, Technology, and Society
Professor of Urban Studies and Planning
Department Head, Program in Science, Technology, & Society
Thesis Committee Member

Certified by: ____________________________________________

Christopher Kelty
Professor
University of California, Los Angeles
Thesis Committee Member

Certified by: ____________________________________________

Natasha Schüll
Associate Professor of Media, Culture, and Communication
New York University
Thesis Committee Member

Accepted by: ____________________________________________

Christine Walley
Professor of Anthropology
Director of Graduate Studies, History, Anthropology, and STS

Accepted by: ____________________________________________

Jennifer S. Light
Professor of Science, Technology, and Society
Professor of Urban Studies and Planning
Department Head, Program in Science, Technology, & Society
Platformizing Higher Education: Computer Science and the Making of MOOC Infrastructures

by

Shreeharsh Kelkar

Submitted to the Program in Science, Technology, and Society on July 21, 2016 in partial fulfillment of the requirements for the degree of Doctor of Philosophy in History, Anthropology, and Science, Technology, and Society

ABSTRACT

This dissertation investigates the role of software in institutional transformation using the example of Massive Open Online Courses or MOOCs. It ethnographically tracks the development of the software infrastructure being built for MOOCs, focusing on three communities—programmers, instructors, and researchers—who centrally participate in the MOOC start-ups' stated mission of reinventing higher education. It argues that MOOC infrastructures are best viewed as an example of a heterogeneous software assemblage that I call the "software-as-platform," that is today being widely deployed and used in a number of industries and institutions. The software-as-platform consists primarily of software that holds together a variety of normative logics: open-endedness; fast, iterative, production processes; data-driven decision-making; governance for emergent effects; scalability; and personalization. Of these, the most important is that its creators give to it an open-endedness as to its ultimate purpose: thus, the assemblage is often framed using the language of "tools" or "platform." I then argue that the software-as-platform is a vehicle through which the norms and practices of Silicon Valley are making their way into other institutions, a process I call "platformization." Finally, I suggest that the software-as-platform enables the emergence of a new form of expertise: tool-making. Tool-makers see themselves as building software tools, whose ultimate purpose comes from their users. The tools themselves draw on many other kinds of expert knowledge chosen at the discretion of the tool.builders.

The dissertation consists of four chapters bookended by an Introduction and a Conclusion. Chapter 2 is an analysis of the public discourse around MOOCs. Chapter 3 describes MOOC infrastructures, showing how a cluster of institutions, software, and people are organized to produce the plethora of courses as well knowledge about education. Chapter 4 tells the story about how edX, a MOOC start-up, turned itself from an educational organization into a software organization by deploying the software-as-platform, thereby transforming and displacing particular institutional roles. In Chapter 5, I analyze the practices of a rising class of tool-makers, computer scientists, and describe how they are able to draw on other kinds of expertise, and intervene in new domains, while still presenting themselves as neutral system-builders.

Thesis Supervisor: Graham Jones
Title: Associate Professor of Anthropology
ACKNOWLEDGEMENTS

Every dissertation writer dreams about writing the Acknowledgements section. One might almost say that it is the academic's equivalent of the Academy Award acceptance speech (though far longer, and perhaps only read by the academic himself). And at the risk of revealing something about myself—who hasn't fantasized about winning an Academy Award?

My first thank yous must go to the people working on MOOCs—the software engineers, program managers, instructors, course developers, instructional designers, computer scientists, learning scientists, educational researchers—who let me hang around them, generously explained their work to me, and indulged my curiosity with sincerity and good humor, even when some of them weren't exactly sure what I was after. Some of these good people are quoted by name in this dissertation; others are presented with pseudonyms. Thank you, one and all; I couldn't have asked for a more generous set of interlocutors. The fieldwork itself was conducted in Cambridge, MA and in the Bay Area, California and it features individuals mainly from edX, MIT, Harvard, Berkeley and Stanford, working with MOOCs and/or online learning. I want to especially thank the following people for introducing me to the different people at my fieldsites: TC Haldi, Sanjay Sarma, Ike Chuang, and Lori Breslow at MIT; Justin Reich at Harvard; Armando Fox at Berkeley; and finally, Mitchell Stevens at Stanford, who also gave me astute feedback on my paper drafts, canny advice about fieldworking-in-practice, and introduced me to the many complexities of the American higher education system.

I am lucky to have a great dissertation committee. My adviser Graham Jones is the model, exemplary, adviser and sometimes, I find it hard to believe that I am his first advisee; it's safe to say that Graham has coached me intensively on what it means to do ethnographic social science. He pushed me to be more abstract, to frame my findings in theoretical ways, so that they become, as he put it, about more than just the case itself. Without Graham's gentle pushing, it is safe to say that I could not have written the Introduction I wrote. Graham has read every word of this dissertation multiple times (not to mention memos, grant proposals, research and teaching statements), and he has always, without asking, given me feedback at exactly the right level of abstraction that I needed: from high-level thoughts on first-drafts (and some of those first-drafts have been ungrammatical, sloppy, and full of spelling mistakes but he always stoically ignored that) to line-by-line edits on final versions. In addition, he coached me on my job interviews, consoled and counseled me through fieldwork crises, wrote numerous recommendation letters for me, and gave me new ideas to work with (I'm sure he remembers our late-night phone conversation with a baby crying in the background when we came up with the idea of using Foucault in the Wenner-Gren proposal). In the future, if I do mentor students, I hope to be at least one-fifth as good as he is.

Susan Silbey's no-nonsense, tell-it-like-it-is feedback was always useful, and some of her memorable lines will continue to guide me even in other projects: "what is this an example of?" "that is a content-less statement," "ideology is the use of ideas in the pursuit of power" are three I can recite off the top of my head. Susan also wrote countless letters of introduction for me that played an important role in getting me access to my fieldsites; there was a comfort to having her as my advocate. Finally, Graham and Susan supported me in my sixth year (of dissertation writing) with their NSF grant on MOOCs, for which I remain grateful.

Natasha Schull worked with me on my most important generals-exam list that plays a key role in this dissertation, on the history and anthropology of computing. I have fond memories
of our sessions, sometimes at the Andala Cafe in Cambridge, where we worked methodically through various books. In one of those, as I narrated to her an incident from fieldwork, she handed back to me an incisive interpretation of that story. Your engineers, she told me, are fighting with their technology. This idea, that engineers fight with their technology to make it work, made me realize that while technology is not autonomous (which is a starting point for STS), neither are engineers some kind of totalizing force for neoliberalism (which critical studies of technology sometimes seem to imply). Jen Light was the lone historian on my committee and her questions about historical significance were always helpful. Chris Kelty joined my committee only a few months before my defense (though I have always drawn on his inspiring scholarly work) but already, his actionable, pragmatic, affable, and astute feedback is visible in the way this dissertation turned out.

For the last six years, my academic home has been with the History, Anthropology, and Science, Technology, and Society program at MIT, or HASTS, as we like to call it. I want to thank everyone at HASTS who have made my stay here one of the most enjoyable times in my life. Dave Kaiser was the Director of Graduate Studies (DGS) when I joined in 2010, and my mentor for my first and second year papers. Dave had the unenviable task of teaching a neophyte how to do social science and I thank him for that. His memorable comment "show me the people [talking about it]" on an early draft of my second-year paper was perhaps the first time I realized what it meant to do interpretive social science: to describe social actors interpreting and acting on particular situations that lead to concrete consequences. Dave also took the lead in establishing the HASTS program seminar, which has been absolutely wonderful for all of us at HASTS. I took three courses with Stefan Helmreich, and his vast knowledge of social theory and his uncanny way of reframing particular issues were both intimidating and wonderful to behold. Stefan gave me feedback on many drafts and papers. His Socratic (but always gentle) questioning when I workshopped the first draft of my Wenner-Gren grant proposal in the HASTS program seminar made me realize what the project was actually about (I actually have that session on tape though I've never been able to make myself listen to it).

Wanda Orlkwoski (who is not from HASTS, but it felt like she could be) introduced me to the field of organization studies, and one of her first comments after reading a draft of the dissertation proposal was to ask me: but where is the technology? That comment has stayed with me, and runs through this dissertation, which teeters between descriptions of people and descriptions of technology (coherently, I hope). Roz Williams, in her History of Technology seminar, taught me how to think about technological determinism, especially through comparative studies. Jeff Ravel introduced me in the first semester, ever so gently, into what doing history is about. Sheila Jasanoff's class at Harvard was my first introduction to how science, technology, expertise, democracy, and governance could be studied together. Heather Paxson, who was our DGS after Dave, took the time to give us final-years two workshops on writing that were enormously helpful (in one of which she made us generate one-line synopses of our dissertations). Heather and Stefan's end-of-the-semester parties have always been great fun; I, for one, don't think HASTS would be the same without them. Finally, I want to thank David Mindell, my adviser for my first two years, for trusting me enough to advocate for admitting a non-social scientist into a social science program.

Karen Gardner is the most kind and efficient administrator I know. I'll never know how she keeps track of all the various deadlines, juggling together all her multiple responsibilities, while still always leaving her door open to anyone who has a problem or just wants to talk. As my defense neared, I suspect Karen got a few headaches but ultimately everything worked out. From the Anthropology department, Amberly Steward helped me patiently through my NSF research proposal, and Barbara Keller, despite all my inconsistencies, managed my NSF
account, doubtless, doing all kinds of articulatory work to make my life smoother. In addition, I want to thank Carolyn Carson and Irene Hartford for being unfailingly helpful, whatever the occasion.

As I wrote this dissertation and gave talks, multiple people, in different venues, have given me feedback on how to shape the argument. In no particular order, they are: Amah Edoh, Clare Kim, Erik Stayton, Mitali Thakor, Canay Ozden-Schilling, Margo Boenig-Liptsin, Ian Lowrie, Yanni Loukissas, Mike Lynch, Will Derringer, Alex Kindel, Ben Gebre-Medhin, Lori Breslow, David Ribes, Gode Both, Karthik Rao-Cavale, Amy Johnson, Renee Blackburn, Arthur Smith, and Chris Walley.

And then there are my friends. My friend networks sometimes overlapped, often they remained distinct, but they all provided something intangible beyond good times, company, and fun. Becoming buddies with Marie Burks and Canay Ozden-Schilling has been one of the best things to have happened to me after arriving in Cambridge. As an STS student, I can't help but notice how crucial Whatsapp group messaging has been to our friendship, and I hope that even as we continue down different paths, our "Gang" group remains as active as ever. Karthik Rao-Cavale and I debated many different topics while having lunch or dinner at the crummy Indian eatery in the MIT Student Center, sometimes going at it for hours, and the effect was both stimulating and comforting. Karthik also read most of my papers and gave enormously thoughtful comments, always encouraging me to take a more political economic tack. There was less politics and more gossip during my lunches and dinners with Karthik and Anindya Sen, which I will miss. My friend Arthur Smith encouraged me to write to an acquaintance of his at edX to see if there was a project there. That particular email intersected with a parallel set of circumstances being enacted at MIT Anthropology (nothing special; just a research proposal on MOOCs that Graham and Susan were writing a grant on), and that's why this dissertation exists. Arthur also went out of his way to give me feedback on my drafts, and shared his vast knowledge of the education world. Mitali Thakor and I were wide-eyed first-years in HASTS, and then weary (and maybe a little terrified) final-years who sat together in Dewey library, simultaneously writing our dissertation and applying for jobs; doing it together was more fun than doing it alone. At HASTS, I was also lucky to be friends with Tom Ozden-Schilling, Amah Edoh, Lucas Mueller, Ashawari Chaudhury, Amy Johnson, Renee Blackburn, Lan Li, Clare Kim, Beth Semel, Jia Hui Lee, and Luisa Reis Castro. I was blessed with a wonderfully convivial cohort: Marie Burks, Shira Shmuely, Caterina Scaramelli and Mitali Thakor. They never made me realize that I was the only guy amongst the five of us.

In Boston, I was introduced to Wai Ching Leong and Judy Lui by our mutual friend Keiko Nomura, and through them, I came to know Chris O'Connell, Dave Muliawan, Viet Nguyen, Jenny Lui, and Victoria Nguyen. The barbecues, the dim sums, the book clubs that we all attended together in Boston always bring back a warm fuzzy feeling. My New York friends, as I like to call them, provided me a stream of comfort and fun through the periodic visits I made there. Joseph Antony, Ari Daman, Sunay Ravindra, Deepak Gupta, Pankaj Rawat, Vinod PG, and Sugi Hadikusumo, thank you. Mayuresh Kshetramade and Mrinal Menon hosted me when I visited London for fieldwork and were the most gracious hosts, to the point of cooking delicious meals for me. In San Francisco, Srinivas Narayanan generously let me live in his apartment in his absence (rent-free, in the Mission district!), for which I will always be grateful. Amit Patwardhan and Zach Burt became my good friends when I lived in San Francisco, and continue to be so (thanks again, Whatsapp).

My father Arvind Kelkar, my mother Sharayu Kelkar, and my sister and brother-in-law Meenal and Hemant Tikekar, have always been supportive of me, in whatever I choose to do,
in ways that are hard to articulate in words. Yes, even when I dropped out of one PhD program in 2006, took up a job, and then gave that up after four years to pursue another PhD. They are very happy that I finished this one! My mother passed away in 2008, about two years before I started this PhD program. I often wish she was around to see this dissertation; I like to think that she would have gotten a real pleasure from it.
# Table of Contents

1 INTRODUCTION ........................................................................................................... 12

1.1 The Social Studies of Software and Algorithms ................................................. 19

1.2 The Software-as-Platform ...................................................................................... 28

  1.2.1 Open-endedness .............................................................................................. 33

  1.2.2 Fast, Iterative, Production Processes ............................................................... 38

  1.2.3 Data-driven Decision-making ......................................................................... 40

  1.2.4 Governance for Emergent Effects .................................................................... 42

  1.2.5 Scalability ........................................................................................................ 44

  1.2.6 Personalization .............................................................................................. 45

1.3 Platformization ........................................................................................................ 47

1.4 A New Form of Expertise: Tool-Making ................................................................. 51

1.5 Theoretical Contributions .................................................................................... 52

1.6 Methodology ........................................................................................................... 54

1.7 Plan of the Dissertation ......................................................................................... 57

2 MOOCs as an Experimental Moment ....................................................................... 58

2.1 A Brief (Critical) History of MOOCs .................................................................. 60

2.2 xMOOCs versus cMOOCs ................................................................................... 69

2.3 Methods ................................................................................................................ 73

3 Software-as-Platform: Introducing MOOC Infrastructure .................................... 76

3.1 The edX Infrastructure .......................................................................................... 77

  3.1.1 Making Software ............................................................................................ 80

  3.1.2 Making Courses .............................................................................................. 91

  3.1.3 Making Knowledge, Doing Research ................................................................. 97

3.2 Conclusion: The Open-Endedness of the edX Infrastructure ............................ 102
4 PLATFORMIZING PEDAGOGY: MOOC INFRASTRUCTURES AND THE TRANSFORMATION OF ROLES................................................................. 112

4.1 edX BECOMES A PLATFORM.......................................................................................................................... 113

4.1.1 The View from edX: The Early Years (2012-13) ....................................................................................... 114

4.1.2 Scaling the Platform (2013-15) ................................................................................................................. 124

4.1.3 Pedagogy Agnosticism: Reasons and Implications .................................................................................. 132

5 THE TOOL-BUILDERS: COMPUTER SCIENTISTS, MOOC INFRASTRUCTURES, AND THE EMERGENCE OF A NEW EXPERTISE... 141

5.1 THE SOCIAL STUDIES OF EXPERTISE.................................................................................................. 143

5.2 COMPUTER SCIENTISTS AS EXEMPLARY TOOL-BUILDERS .................................................................. 147

5.2.1 Juho Kim and Learnersourcing ............................................................................................................. 149

5.2.2 Chinmay Kulkarni and Peer Assessment ................................................................................................. 159

5.3 COMPARING COMPUTER SCIENTISTS TO LEARNING SCIENTISTS .............................................. 167

5.3.1 The Politics of Expertise ...................................................................................................................... 178

5.4 CONCLUSION: A NEW FORM OF EXPERTISE? ...................................................................................... 184

6 CONCLUSION .................................................................................................................................................. 190

7 REFERENCES .................................................................................................................................................. 211
TABLE OF FIGURES

**Figure 3.1**: How a MOOC looks. The course shown in the figure is called “Designing Mobile Experiences” and teaches learners how to understand user requirements and needs and then build a smart-phone application based on this. 81

**Figure 3.2**: The Peer Assessment Interface. Learners enter their free-form responses in the box provided. Once the time for submission has elapsed, learners are given the responses of three or more of their peers which they assess using an instructor-given rubric. 83

**Figure 3.3**: The Studio application for authoring courses. Studio separates the content of the course from its presentation. Course authors can therefore add content (text, assessments, videos) through a GUI. The software will make sure that the final output looks and runs like a course. 92

**Figure 3.4**: A bar graph from the Analytics dashboard. This shows instructors how learners responded to a particular assessment. 100

**Figure 3.5**: The clickstream data from a course. 101

**Figure 4.1**: A table from a UniversityX “Onboarding” document from edX to the participating university. The table outlines “assets,” “tools,” and “resources” for the UniversityX personnel to use in order to produce MOOCs. 129

**Figure 5.1**: A how-to video on YouTube that shows, through a step-by-step process, how to add blurs to an image in Photoshop. 150
Figure 5.2: A computer scientist’s answer to the question of how the field can contribute to the study of learning and the design of educational technologies. ................................................................. 168

Figure 5.3: The Lytics Lab workspace, Stanford University. Meetings took place around a whiteboard with around 10-15 people in attendance at every meeting. Photo from the Lytics website................................................................. 174

Figure 5.4: The Program Manager of NSF’s CyberLearning Initiative advises computer scientists on the best approaches to securing NSF funding............. 183

Figure 6.1: A slide from George Siemens’ talk at MIT in July 2013 ......................... 199
1 INTRODUCTION

On August 20, 2011, Marc Andreessen, Silicon Valley entrepreneur, creator of the first Netscape Web browser, and venture capitalist extraordinaire (Friend 2015), published an op-ed for the Wall Street Journal titled "Why Software is Eating the World" (Andreessen 2011). "Six decades into the computer revolution," he announced, "four decades since the invention of the microprocessor, and two decades into the rise of the modern Internet, all of the technology required to transform industries through software finally works and can be widely delivered at global scale." The result, he argued, would be a complete reorganization of a variety of industries due to "lower start-up costs and a vastly expanded market for online services." His examples ranged across many sectors: Amazon for books, Netflix for movies
and television, Pandora and Spotify for music, Zynga for games, Skype for communication, LinkedIn for recruiting, and many others. "Health care and education," he predicted, "are next up for fundamental software-based transformation." Just a few weeks before Andreessen's op-ed came out, a video appeared on YouTube on July 17 2011 announcing that Stanford University's Sebastian Thrun and Google's Peter Norvig, both experts on Artificial Intelligence, would open up their Stanford course on the topic to the whole world (knowlabscom 2011). The course, called "Introduction to Artificial Intelligence," ran simultaneously at Stanford and to the outside world from October to December 2011 (i.e. in Stanford's fall quarter); it included lectures and interactive assessments that were graded automatically by software; non-Stanford learners who finished the assessments were even given a certificate. This prompted two other faculty members at Stanford to open up their courses as well: Andrew Ng's Machine Learning and Jennifer Widom's Databases went live in Fall 2011. Greeted with tremendous excitement in the media, these courses came to be called MOOCs or Massive Open Online Courses. By May 2012, three new software start-ups, Udacity, Coursera and edX, all of them led by computer

1 While definitely carrying a kernel of truth, Andreessen's essay also suffers from a surfeit of over-generalizing and over-statement. Thus, both Netflix and Pixar are labelled as "software companies" even though they have very different business models and core strengths. Amazon is praised for its lack of brick-and-mortar stores while Walmart, with its vast network of brick-and-mortar stores, becomes an example of software-driven efficiencies because of its inventory management software. In his rush to label every successful enterprise a software company, Andreessen ends up blunting his own argument with his expansive notion of what software is.
science faculty members from Stanford and MIT, had come into existence. They offered MOOCs by collaborating with instructors and universities; the frenzied activity of 2011-12 prompted the New York Times to declare 2012 as the "Year of the MOOC" (Pappano 2012). The earliest Stanford MOOCs were inspired by a range of Internet websites (of the Web 2.0 variety) dedicated to vague-to-explicit forms of learning, websites that could well have appeared in Andreessen's list of companies whose software was eating the world, in this case, the world of education. These websites included: Khan Academy with its mathematics video tutorials and assessments; YouTube and Lynda.com with their extensive range of user-generated how-to videos on topics ranging from how to apply makeup to how to make a good PowerPoint presentation; and StackOverflow with its crowdsourced format of having users answer each other's questions. Not surprisingly, the MOOCs on Coursera, edX, or Udacity resembled these websites in some respects, and comprised lecture videos, simulations, interactive tests, assessments with automatic grading, discussion forums, and even a certificate for those who finished. "What changed in 2011," the Stanford pioneers wrote in

---

1 Of the three MOOC start-ups, edX (and to some extent, Coursera) will be the primary focus of this dissertation. Both Coursera and edX operate with similar business models: collaborating with universities to offer a variety of courses. Udacity has turned its attention elsewhere, partnering mostly with corporations to offer different kinds of job certificate programs such as its new "nanodegree" in data science (Byrnes 2015).

2 According to Wikipedia, "Web 2.0 describes World Wide Web sites that emphasize user-generated content, usability, and interoperability. [...] Web 2.0 does not refer to an update to any technical specification, but to changes in the way Web pages are made and used. A Web 2.0 site may allow users to interact and collaborate with each other in a social media dialogue as creators of user-generated content in a virtual community, in contrast to Web sites where people are limited to the passive viewing of content. Examples of Web 2.0 include social networking sites, blogs, wikis, folksonomies, video sharing sites, hosted services, Web applications, collaborative consumption platforms, and mashups" ("Web 2.0" 2016).
Chapter 1: Introduction

their retrospective account, "was scale and availability" (Ng and Widom 2014). What they meant was that they were able to do for their own, formal, courses—make them available to large numbers of people—what other Internet website had been able to do for informal learning. And indeed, the scale and the availability of MOOCs were often the topic of conversation in the early days, when many institutional actors saw them—and still do—as promising solutions to the United States' "crisis" in higher education: high costs for students, dramatically reduced government funding for public universities, and a host of underserved "non-traditional" students (i.e. working adults who studied part-time who were not served by traditional non-profit universities and were instead forced to attend exploitative for-profit schools). A number of institutional actors coalesced around the MOOC start-ups: Marc Andreessen’s fellow venture capitalists eager to cause the next "disruption" in the higher education field (Carey 2012; Christensen and Eyring 2011); elite American universities like MIT, Stanford, Harvard, Berkeley, Michigan, Princeton, and Penn, equally determined to stay on top of this disruption and possibly engineer it themselves (Carey 2015); governments, both federal and state, eager to cut costs and promote innovation; and of course, philanthropic foundations thinking about new forms of interactive pedagogy and the computational measurement of learning (The Chronicle of Higher Education 2013).

In 2012, MOOCs looked like a killer software application that would eat the higher education industry. In 2016, with the vantage point of time, they seem more like experiments; their future seems both vaguely promising and completely uncertain. In this dissertation, using MOOCs as my case, and the methods, tools, and theories of Science and Technology Studies (STS) as my guide, I examine seriously claims like Andreessen’s about how software is transforming industries. Focusing on edX, a non-profit online learning enterprise jointly founded by MIT and Harvard, this dissertation ethnographically tracks the development of the software infrastructure being built for MOOCs, focusing on three communities that have
organized around this infrastructure—programmers, instructors, and researchers—who centrally participate in the MOOC start-ups’ stated mission of reinventing higher education. These actors participate in three activities that I deem central to MOOC infrastructures: coding software, making and running online courses, and making knowledge claims about best teaching and learning practices.

Before I summarize my main findings, let me answer the question: why study infrastructure? Taking my cue from STS which views infrastructures as a "seamless web" that is shaped by contingent technical, political, and economic factors (Hughes 1993; Hecht 2009; Mackenzie 1993; Bowker and Star 2000), I define MOOC infrastructures as the assemblage of people, software, and institutions that produce the interactive courses (sometimes called "courseware") that one sees on edX or Coursera, as well as the knowledge claims about good teaching and learning emerging from MOOCs. Larkin (2013) defines infrastructures as "matter that enable the movement of other matter"; "what distinguishes infrastructures from technologies," he argues, "is that they are objects that create the grounds on which other objects operate, and when they do so, they operate as systems" (2013, 329). Leigh Star, who pioneered the studies of infrastructure within STS, argued that infrastructure is what is taken for granted in the accomplishment of tasks; it is therefore perspectival (in the sense that what counts as infrastructure depends on who you ask), fragile, held together by invisible labor, and becomes visible when it breaks down (Jackson 2014; Star 1999). To study infrastructure through a social science lens, Bowker and Star (2000) suggest using the method of "infrastructural inversion": "focus[ing] on all the activities that hold together the functioning of infrastructure rather than those that it invisibly supports" (Ribes and Lee 2010).

Understanding how the infrastructure for MOOCs is made, maintained, and used, this dissertation argues, gives us an insight into how roles and institutions for teaching and learning—the learner, the instructor, the educational researcher, the instructional designer, the
engineer, the college, the university, the course—are being re-imagined and scripted into the technology. If MOOC infrastructures are the future of higher education, and will be incorporated into them in some way, as many think, then investigating how MOOC infrastructures are made, the values written into them, and the forms of expertise they privilege, are important to understanding the impact of MOOCs as institutions that impinge on the American higher education sector.⁴

Based on a two-year ethnographic study of the edX infrastructure and the programmers, instructors, and researchers who build, maintain and use it, this dissertation argues that MOOC infrastructures are best viewed as an example of a heterogeneous software assemblage that I call the "software-as-platform," an assemblage that is today being widely deployed and used in a number of industries and institutions. The software-as-platform consists primarily of software which holds together a variety of technical and normative logics. The most important characteristic of this assemblage is that its creators give to it an open-endedness as to its ultimate purpose. This is driven, in part, by their desire to enroll as

---

⁴ The additional question is: why software infrastructure? Why not hardware, for instance? The answer here is more pragmatic. All the actors I observed and talked to saw themselves as creating software or using software (sometimes both). The software built by edX, called Open edX, is hosted on Amazon Web Services (AWS); this is part of what Andreessen is talking about when he mentions "lower start-up costs." Rather than spending resources to have their own servers, server farms, or data centers, start-ups today (like edX or Coursera) simply host their software on AWS. AWS takes care of these nitty-gritty matters allowing these start-ups to concentrate on writing code. One could doubtless apply an additional layer of infrastructural inversion to ask how AWS becomes an infrastructure. But that is not the point of this dissertation which looks instead at how software infrastructure for learning, teaching, and research is built, used, and maintained.
many diverse actors as possible into this assemblage, and in part, by their self-conception as stewards of innovation. Thus the assemblage is often framed using the language of "tools" or "platform." The purpose of the tool itself (tool for something, platform for something) is left to the different human members of the assemblage ("users"), a property that I characterize as the value-agnosticism of the software-as-platform. In addition, the software-as-platform holds together five key normative logics: a commitment to fast, iterative, production processes (what's referred to in Silicon Valley as "fail fast, fail often" ethos), an emphasis on data as the primary way to make decisions or data-drivenness, the valorizing of a particular open-ended style of governance, and an idealizing of the concepts of "scalability" and "personalization." It is the software that ties together all of these normative and technical logics into a coherent whole.

It should be obvious by now that the normative logics of this assemblage trace their origins to the work practices and institutional logics within the world of Silicon Valley software start-ups. Therefore, the second claim of this dissertation is that the software-as-platform, as an assemblage, is a vehicle through which the norms and practices of Silicon Valley are making their way into other institutions; I call this process through which the software-as-platform is used by various actors to reconfigure/rejigger their institutions as "platformization." By no means should this be seen as a one-sided colonizing process; rather, there are actors within many institutions who are inspired by Silicon Valley's success, and actively seek to draw on its practices and logics as they try to reinvent their own institutions. Using the software-as-platform, these actors re-imagine their institutions in terms of their processes, practices, roles, and sometimes, even their ultimate purpose and/or underlying theories. The key way in which institutions are being transformed, I argue, is that roles within institutions are re-imagined as "innovative" roles.
A final contribution of this dissertation is its description of the emergence of a new form of expertise, constituted and enacted through the software-as-platform. While experts, typically, have tended to constitute themselves as authorities on a particular domain or subject (Abbott 1988; Carr 2010; Eyal 2013; Collins and Evans 2008), these new experts conceive of themselves as tool-builders of software tools, whose ultimate purpose is decided by their users; the tools themselves draw on many other kinds of expert knowledge from other domains/disciplines/subjects (on tool-building, see also Malaby 2009). It is the tool-builders who decide what expert knowledges to encode into their tools, and how to distribute the labor between the tool itself and the user of that tool. I show in this dissertation the asymmetry of this form of expertise: while in theory anyone who has tool-building skills (i.e. can write computer code) can build these tools, the tools with most impact are often built by those with access to institutional resources.

In the next section, I begin by summarizing the recent literature emerging from within STS and neighboring disciplines on the social studies of software and algorithms. I follow that by detailing the three scholarly contributions of this dissertation to the social studies of software. Next, I describe my methods and methodology. I conclude with a plan of the rest of this dissertation.

1.1 The Social Studies of Software and Algorithms
While Andreessen and others talk about software eating the world, there is a parallel debate within the United States over the role of so-called algorithms. A common definition of algorithms is the one used in computer science textbooks: a recipe, or method, to solve a certain problem (like sorting numbers). Algorithms are often seen as the core of certain major systems whose workings become matters of public discussion: e.g. Google and its PageRank search algorithm (Brin and Page 2012) that is used to bring up search results when one searches on Google, or Facebook and its Edgerank algorithm that calculates the order of
items in Facebook's newsfeeds that are served to the Facebook user (Taina Bucher 2012).

Most of the public debate around these algorithms has been about their objectivity, their bias, and their opaqueness/secrecy. Recently, for example, Facebook came under fire from conservative media when it was revealed that its "Trends" were not derived wholly by some mechanical procedure, but curated by some of its employees who looked at the results from a mechanical procedure and then used their judgment to publish what they thought would be most interesting to Facebook users (see Gillespie 2016 for an analysis). Questions in the public debate have ranged from: are these algorithms biased or are their data biased? Are the algorithmic results fair? Are they contestable? What are the terms in which a public debate over algorithms should be conducted? And so on.

When scholars from STS, information studies, sociology, anthropology, and media studies have analyzed these debates however, they have come to a different conclusion. These scholars (Ziewitz 2016; Seaver 2013; Gillespie 2014) argue that these algorithms need to be thought of not just as technical artifacts or computer science recipes but as complex emergent creations that are constructed out of a diverse set of (often unarticulated) assumptions of their designers and their users. These assumptions include the business goals of the corporation that creates the algorithms (Gerlitz and Helmond 2013; c.f. Sandvig 2014 on the evolution of Facebook's "like" feature); the desire not to give away their secrets (Pasquale 2015), cultural assumptions about salient social categories like adult content (Gillespie 2013) or homosexuality (Ananny 2011); technical assumptions about computational complexity like its inherent opaqueness (Burrell 2016); and many others. How these algorithms "work" in all contexts across all categories may not even be clear to their designers. Rather than focusing on algorithms per se, these scholars suggest focusing on the systems of production that these algorithms are part of, that these scholars variously term as "algorithmic systems" (Seaver 2013), "networked information algorithms" (Ananny 2016) etc.
Chapter 1: Introduction

Taking the lead from this literature, this dissertation also focuses on socio-technical systems of production, circulation, and consumption, rather than isolated components or behaviors. But I focus, deliberately, on software, or software infrastructures, as my category of analysis, rather than algorithms. There are three reasons for focusing on software rather than algorithms per se. The first is empirical: most of my interlocutors saw themselves as building software or programs rather than algorithms. The second reason is more methodological and theoretical. While computer science typically makes the distinction between algorithms and code—algorithms are abstract recipes that are realized in code—working software developers rarely code for specific algorithms; rather, they build programs and systems which may use particular algorithms. There is an already existing institutional system for software development with its own set of roles (architects, developers, technical writers, users) and its own set of methods (waterfall model, Agile model). These already-existing categories as well as the robust debate about these categories that happens among software experts make software a great topic for an STS-oriented analysis. Taking software as my unit of analysis also allows me to take an "architectural" viewpoint on socio-technical systems, and thereby integrate both the social and the technical into my analysis; focusing on the architecture means taking how social categories are represented in technical systems, not just within the mathematical formalisms of algorithms, but in the relationship between various working software programs/modules that constitute the software-as-platform (see Dourish 2014 for an architecturally-centered analysis of noSQL databases). Finally, social scientists focus on algorithms because these algorithms are the locus of governance (Ziewitz 2016); thus, it is no wonder that researchers have often focused on algorithms that rank, evaluate, deem relevant, or compute similarities between items. But these are not the only mechanisms of governance. Governance happens through interfaces, documentation, and even through emails, as much as it does through algorithms that rank or sort. As I show later, the potency of the software-as-
platform as an assemblage comes from its agnosticism, its ability to be offered up as a tool to multifarious users to accomplish their own diverse aims and purposes. The power of the assemblage derives from its apparent purposelessness. This, in itself, is a form of governance. My focus on software however in no way invalidates the extensive literature on the politics of algorithms; indeed, it adds to it by showing that the question of governance is key to understanding the power of the new software systems but governance can take place in different ways beyond just nudging.

What is software? The category of software is usually used in opposition to hardware: while hardware refers to the tangible parts of the digital computer, software is intangible. As defined in (MacKenzie 2006, 1), software is a "generic term for those components of a computer system that are intangible rather than physical. It is most commonly used to refer to the programs executed by a computer system as distinct from the physical hardware of that computer system, and to encompass both symbolic and executable forms for such programs."

The historian Nathan Ensmenger writes eloquently about both the challenges of studying software from a social scientific lens as well as why it is so important to do so:

Software is an extraordinarily heterogeneous technology; it straddles the boundaries between science and technology, art and engineering, and the intellectual and the material. Software is clearly a built object, designed and implemented by humans, yet it is also a mathematical formalism, an appropriate object of study for the scientist or theorist. The people who develop software refer to themselves alternatively as programmers, computer scientists, or software engineers— as well as black artists, wizards, hackers, gurus, and cowboys. They do not fit neatly into established academic or professional categories. [...] A computer program is invisible, ethereal, and ephemeral. It exists simultaneously as an idea, as language, as technology, and as practice. Certain forms of software, such as a sorting algorithm, can be generalized and formalized as mathematical abstractions, while others remain inescapably local and specific, subject to the particular constraints imposed by corporate cultures, formal and informal industry standards, and/or government regulations. In this sense, software sits ambiguously at the intersection of science, engineering, and business. As may be imagined, all this heterogeneity renders software extraordinarily difficult to isolate, understand, and write about. [...] (Ensmenger 2012, 763–64)
To situate the category of software, I draw on the work from the history of computing (as well as the interdisciplinary field known as software studies (Fuller 2008). Early computers did not incorporate the software/hardware distinction: to program these computers meant reconfiguring switches and cables, a practice that was cumbersome (but interestingly was often done by women, see Light 1999). Through the development of new computer architectures and programming languages that separated the instructions for manipulating the data from the data itself but treated them as formally equivalent, the activity of "programming" started to shift towards how we currently know it today: a programmer seated at his or her desk typing on the keyboard in a programming language like Java or C++. Writing "code"—which turns into a program when compiled—became the way to issue instructions to a computer.

The historian Martin Campbell-Kelly (2004) identifies three kinds of software: custom-made software for corporations, standardized corporate software products, and finally mass-market software products. He sees these three as proceeding in sequence, although none of them ever really became extinct. Software in the 1950s and 60s was increasingly custom-made by in-house computer programmers for different corporate tasks (payroll, inventory and so on). Generic corporate software products came out in the 1960s; these were standardized products aimed at corporations for important tasks: inventory management, customer relationships, and so on. According to Campbell-Kelly, the market for these products resembled the market for capital goods: the products were high-cost (both in terms of price and the effort it took to make them), took a lot of maintenance and servicing, and were sold in low-volumes. The era of mass-market software products began with the popularity of the minicomputer and then continued with the PC revolution in the 1980s. Mass-market software products often resembled the products of the music industry, and their makers often saw themselves as
producing an information good. They were lower-cost, they required less maintenance and service, and they sold in far higher volumes.

Through all these times, an energetic debate took place amongst software developers and managers about what software was, and how it should be produced and maintained, and what the status of software experts should be within a corporation (T. Haigh 2001; Ensmenger 2010). The main conflict in the pre-PC computer era has been the jurisdictional negotiation (Abbott 1988) between programmers and mid-level managers. Programmers, in their various incarnations as coders, systems analysts, and consultants, have tried to claim—using using as their bargaining chip, their technical skills and the increasing role of data processing in the day-to-day working of corporations—the responsibilities of mid-level managers. Managers, for their part, tended to label programmers as unsocial, unreliable and unlikely to work well in a corporate environment, pointing to how software projects routinely overran budgets, timeframes, and did not produce very good software at the end. Responding to these criticisms and the accompanying discourse of the “software crisis,” programmers worked to professionalize and standardize their work processes. They concentrated on three channels: through the creation of what are called "software engineering" methods, through tools for automatic programming, and by creating associations and mechanisms of credentialing. Yet, they were not successful. The big reason was that they were never able to resolve the central tension in making of software: whether programming is an innate, "black art," or a skill that can taught. Within the corporation, programmers thus remained technicians (Barley 1996), rather than managers. Haigh (2003) argues that while these conflicts have never been completely settled, a somewhat stable social configuration was reached with the rise of personal computing and the decentralization of computing within the corporation. Rather than a single "data processing" department with a single mainframe computer that a priestly class of programmers then interacted with on behalf of the entire firm, corporations now tend
to store their data centrally, but let individual employees work on it from their own workstations using customized software programs. At the same time, the job of customizing software so that it can be used within a corporation is done either by an internal “Information Technology” department (where the programmers or software experts are often situated) or by outside business consultants (whose job is labeled managerial but who are essentially software experts.

An additional wrinkle in the software design process—both in terms of the business models as well as work practices of developers—has come about with the rise of the Internet (Abbate 2000). The place of the software has started to shift from a user's desktop environment to being on the Web (a characteristic example here would be using Microsoft Word which is installed on the desktop versus Google Docs which is accessed from a Web browser). This is often referred to as "cloud computing"—software residing in an amorphous cloud to be drawn on by the user as needed. This has led to a further shift within the software industry: from thinking about software as a mass-produced commodity to thinking about software as a "service" (often referred to as software-as-service). The three separate spheres of software vendors—mass-produced software, enterprise software, and the "computer services" industry—thus have come together, "facilitating the entry of mass-market vendors into enterprise software and of both mass-market and enterprise software vendors into computer services" (Campbell-Kelly and Garcia-Swartz 2007, 735).

The move into the browser allows for other kinds of changes in the software-making production process. It allows software developers to situate both their code and their data at one central location. The code thus becomes endlessly modifiable; most Web-based start-ups post changes to their production code every week, often multiple times a week. They also get to store, often in minute grains of detail, the interactions of their users with their software; these "big data," in turn, can guide future revisions of the software itself. Under the software-
as-service model, the Internet itself starts to function like a desktop operating system: thus one piece of software can draw on another piece of software, conceived as a service, for its own work. This kind of interface that allows one software program to call another is called an Application Program Interface (API). Note that APIs pre-date the Web; their origin can be traced to the IBM 360 systems pioneered by IBM in 1969 (Tania Bucher 2013) which allowed a number of plug-and-play modules to be attached to the core system itself. The concept of software-as-service however has dramatically increased the possibilities of APIs.

The rise of interfaces (APIs, but also Graphical User Interfaces or GUIs) and the concept of software-as-service have changed the nature of the Internet itself. Originally an open network, recent years have seen the rise of so-called “platforms”—the so-called "social media" portals like Facebook, Twitter, and YouTube. These sites have heralded a transformation of the World Wide Web from a relatively open, non-commercial space, to one that is dominated by a few behemoths, obligatory passage points that curate the Internet experiences of many Web users. They have also been accompanied by a rising discourse of democratization and empowerment that sometimes obscures more than it reveals (Kreiss, Finn, and Turner 2011). Three techniques have played an important role in this transformation (Helmond 2015): "separation of content and presentation" (i.e. letting users upload content without worrying about its presentation) which is usually accomplished by creating Graphical User Interfaces or GUIs that lets end-users create content without worrying about its presentation, the so-called “user-generated content”; "modularization of content and features" (i.e. allow features e.g. the Facebook "like" button or the YouTube video to be embedded and used in other websites); and finally, "interfacing with databases" (i.e. letting user clicks across the Web aggregate into the platform's proprietary databases). The last two are accomplished by creating APIs that allow programs to talk to other programs.
and by GUIs that allow users to communicate with each other (and thereby with databases as well).

Scholars from media and communication studies have pointed to the contradiction between the discourse of democratization that accompanies this new proprietary Internet and the informational asymmetry between the ordinary users and the new media start-ups like Facebook and YouTube. Clearly, YouTube and Facebook have increased ways in which people can participate in economic, social, and political activities, which scholars refer to as the question of participation. Various scholars and lay-people alike have described this with concepts like "networked publics," "peer production," "presumption," "produsers," "prosumers," "convergence culture," "cognitive surplus," "digital serfs," and others to describe how the Internet has transformed participation in (mostly) public spheres, blurring distinctions between producers and consumers, firms and customers, corporations and publics, and, work and pleasure (see Fish et al. 2011 for a summary and critique). But scholars have also pointed out the great asymmetry between the companies that own the infrastructure, say Facebook or YouTube, and their end-users. The companies have the power to shape their users' actions, collect data based on those actions (which is their source of revenue), and nudge users into actions that increase their own profitability, while still positioning themselves as just neutral media that provide value to their users by giving them a channel of production and consumption (Van Dijck 2013; Gillespie 2015). This asymmetry may have other consequences as well. Couldry and van Dijck (2015) point to how hegemonic Facebook practices, promoted by Facebook's algorithms and interfaces, are transforming the category of the "social" into an impoverished one concerned with friending, liking, status-updating, and sharing commercial content. They argue that social theorists need to reclaim the category of the "social"—a category that has fallen into disfavor recently.
because it presumes and reifies "society"—in order to give it back its complexity, which is being reshaped to commercial ends by social media companies.

This dissertation is in conversation with all of these literatures. I pay close attention to the interfaces—technical (GUIs or APIs) and institutional—through which users are both shaped and empowered within the world of edX (i.e. between learners, programmers, instructors, and researchers). I also pay close attention to the asymmetry between the positions of those who design the software architectures (architects at edX) and those who use them in various ways (learners, instructors, researchers, external developers). In the next section, I describe the contributions of this dissertation in more detail, particularly with reference to the literatures discussed. First, I describe the assemblage I call the software-as-platform and its constitutive elements. Then I describe how the software-as-platform is being used to transform institutions. Finally, I describe the emergence of a new form of expertise that is privileged within the software-as-platform: tool-making.

1.2 The Software-as-Platform
This dissertation argues that MOOC infrastructures—the combination of software, institutions, and people that collectively produce courses and knowledge claims emerging from the MOOC ecosystem—are best viewed as a heterogeneous assemblage that I call the software-as-platform, an assemblage held together materially by software, consisting of particular normative logics that apply to both human practices and technological systems. The normative logics include an ethos of open-endedness; fast, iterative production processes; data-driven decision-making; governance for emergent effects; scalability; and personalization. I focus on each of them in turn. But first, I clarify what I mean by "assemblage" and "platform."
First, assemblage. The idea of the assemblage has been used in recent empirical work. Ong and Collier (2008) use it as an analytic to understand the mechanisms of globalization, to account for how both the "local" and the "global" are both emergent properties. Recent studies of algorithms too have used the assemblage as a unit of analysis in describing the production and consumption systems that these algorithms are part of: "sociomaterial assemblage" (Introna 2016) and "assemblages of institutionally situated code, practices, and norms" (Ananny 2016) Assemblages are heterogeneous in that disparate elements are held together, often by human practices or technological artifacts. Elements of assemblages may often be in tension with each other. Some actors may have more use for certain aspects of the assemblage than others; and there may be disagreements between actors as to what the significance of the assemblage is. Yet, assemblages are also singular in the sense that the assemblage itself is more than the sum of its parts. Assemblages can also help bridge the micro-macro divide by linking human practices, organizational structures, and technological artifacts all together (DeLanda 2006; see also Little 2012 for an explanation). Most importantly, actors and institutions can draw on the assemblage, or its particular parts, to debate fundamental questions of ethics, methods, and values (Collier and Lakoff 2008).

Second, platform. The notion of software as a "platform" has emerged in the last two decades, inspired by the success of Microsoft and Intel (Gawer and Cusumano 2002). In a widely read, influential blog-post Marc Andreessen argued that a platform is software that allows other software to be built on top of it (Andreessen 2007). Software analysts, especially those who write about industry strategy, argue that a particular kind of "industry-wide platform" has become possible with the rise of personal computers and the Web (Cusumano 2010). An operating system like Windows, a web browser like Netscape, or an appliance like the iPhone is often not worth anything by itself; it gains value only from the kinds of applications or websites that it can work with. A software apparatus that can give
rise to an entire industry is called a “platform.” Building a platform then—as opposed to just software, or a program—is seen as the key to achieving lasting market dominance.

A platform thus has an organizational and technical "core" (that can be built upon) as well as a periphery of "complementors," (who do this building); all together (the organizations, the technology) collectively make up an "ecosystem." Economists refer to a platform like this as a multi-sided market, a conduit between different classes of consumers and producers, with "network effects," defined as "positive feedback loops that can grow at geometrically increasing rates as adoption of the platform and the complements rise" (Gawer 2014; Cusumano and Gawer 2016; Gawer and Cusumano 2016; Cusumano 2010, 32). What a platform ecosystem like this facilitates is technical innovation and market dominance (if successful and widely adopted). The key facilitator of this innovation is the "interface" between the core and the periphery—an interface which is both technical and institutional. As Gawer defines it, "the interface is therefore a divider (of labour between distinct teams), but also a connector, and a conduit of selected information facilitating interconnection." (Gawer 2014, 1243).

Building and maintaining a platform thus means not just selling more software or gaining more users but becoming a "platform leader" i.e. an organization that makes a well-designed core technical infrastructure while also recruiting an army of complementors, creating distributed innovation, while still staying in charge of this particular ecosystem as a market leader. Cusumano and Gawer (2016) list "four levers of platform leadership": deciding carefully the core functions (which the platform leader does) and the peripheral functions (which can be assigned to complementors); architecting a technology with the right kind of technical interfaces that will allow both the core and periphery to innovate; deciding what kinds of relations—collaborative, competitive, both—should inhere between the platform leader and the complementors; and finally, creating the right internal organization to manage
the relationship with complementors including the conflicts of interest that may arise. Careful decision-making on each of these four lines will allow a company to assume "platform leadership."

Silicon Valley companies are increasingly adopting the "platform" playbook, derived from the strategy literature, and are attempting to be "platform leaders" by building a software infrastructure that caters to multiple constituencies simultaneously, betting that this interconnection of different constituencies will be for them an important source of revenue. YouTube, for instance, started off as a portal for "you," the amateur video enthusiast who could use the site to share videos with a wider audience; it was a platform for "user-generated content" (see Burgess and Green 2013 on YouTube's history; Jenkins 2006 on the possibilities of user-generated content). In its early incarnation, YouTube did not allow advertising but that quickly changed once the question of revenue became crucial to its own survival as a company. Eventually, YouTube ended up making a play for multiple constituencies: as a portal for media companies, established and independent, to market their own products like music videos, movies, and albums; for the average Joe i.e. you, the user, for his or her creative output; for advertisers who wished to find consumers based on their interests in YouTube's videos; as well as any others who wished to use video with their own agenda (e.g. universities hoping to turn lectures into videos). In doing so, YouTube started to deploy the platform semantic in multiple ways. As Tarleton Gillespie (2010) argues, YouTube's use of the word encompasses at least three senses: "technical platforms, sometimes as platforms from which to speak, sometimes as platforms of opportunity." "Whatever tensions exist in serving all of these constituencies," Gillespie argues, "are carefully elided."

The software-as-platform thus consists of software—the material entity that runs on computing infrastructures and is built by writing code—as the material substrate that holds
the assemblage together. The software is materially and ideologically constructed as a "platform": a thing which diverse actors, individuals and organizations, can put to work for their own goals and purposes, "technical platforms, sometimes as platforms from which to speak, sometimes as platforms of opportunity" (Gillespie 2010). The architects of the software-as-platform provide this through interfaces: Graphical User Interfaces (GUIs), that allow humans to connect with other humans, programs, and databases; and Application Program Interfaces (APIs), that allow actors to write code and build programs that can connect to other actors, programs, and databases within the software-as-platform (all of which thus themselves became part of the overall software-as-platform assemblage). The interfaces construct and configure these diverse actors as "users" of the software-as-platform (as opposed to the “architects” who get to design the interfaces). They are users because they use it for their own ends and because they are often configured as self-sufficient through the interfaces themselves; they are also users because they only have access to part (rather than all) of the software, that their interfaces allow them to access (and the access will vary amongst users themselves depending on the types of interfaces). There is thus an asymmetry between the actions available to the users and to the architects; users can modify the periphery of the software infrastructure through which they are constituted, but not the core of it; only the architects of the software-as-platform can modify the core. In a way, those that control the interfaces (i.e. the architects) are the most powerful actors within the software-as-platform.

A note on terminology: sometimes I will refer to software-as-platform as just platform for convenience. I will also refer to those who play a key role in designing the software-as-platform, especially its interfaces, as platform architects. Everyone else is a “user;” being a user is by no means a passive role, yet the possibilities of user actions are constrained by interfaces.
1.2.1 Open-endedness

What inspires the architects of the software-as-platform to build these assemblages, rather than just software or programs? First, of course, there is market share. Building an industry-wide platform is a ticket to creating long-lasting market dominance the way Intel and Microsoft did with the PC revolution. The journalist Kevin Carey (2012) suggests that this preference for building "platforms" has something to do with the spread of the ideology of "disruption" (Christensen 1997) that has taken hold within Silicon Valley. An industry-wide platform will be harder to disrupt, because it caters to a variety of users and purposes; while a routine software program can be easily disrupted as new competitors or technologies arise. Entrepreneurs can choose whether they want to pursue "sustaining innovation" or "winner-takes-all" platforms. As Carey writes, "Investors want to put their money in platforms, and start-ups want to build platforms, because right now, and for the foreseeable future, platforms rule the world."6

---

3 While riding the PC revolution in the 1980s to market dominance, both Intel and Microsoft today are on the losing side of the smartphone revolution.

4 The theory of "disruptive innovation" dates to the publication of Clay Christensen's *The Innovator's Dilemma* (1997). Using the example of the hard-disk industry, Christensen theorizes that it is very difficult for market leaders to pay attention to new techniques that may upend their business model completely; this is because these companies already have much invested in their current way of doing things, which means that the new technology is often picked up and capitalized on by an upstart new company, which may end up putting the incumbent out of business. From this initial beginning, the ideology of disruption has managed to reach almost prescriptive status within the world of Silicon Valley, mostly from Christensen's own efforts to cash in on the success of his first book. From an initial implied prescription that companies remain sensitive to new potentially disruptive technologies, the theory of disruption now includes everything from how companies can manage disruption to how they can intentionally set out to disrupt entire industries.

---

5

6
But there is a second reason why the software-as-platform has become popular within Silicon Valley: Silicon Valley’s engineers and entrepreneurs not only see themselves as looking for market share, but also as stewards of innovation. By building platforms, they are providing a forum to produce and nurture innovation, while staying on top of and managing disruptive innovations. Innovation is another concept, like disruption, that has attained great currency within Silicon Valley as well as policy circles within the last 30-odd years; the origins of this discourse lie in economics and business schools. Innovation came to be seen as the key to improving national productivity in the United States in the 70s when the economy was hit by stagflation and the post-war liberal consensus withered. This argument was mainly made by economists in the academy and made its way out of the academy into consulting, think-tanks, and policy-making. By no means was this argument only made by right-wing economists; rather it also came from left-leaning theorists like Kenneth Arrow (Berman 2011; see also Vinsel 2014b; Vinsel 2014a for the rise of “innovation” as a category; for alternatives to innovation, see Russell and Vinsel 2016; Edgerton 2011). In this view, it is innovation that drives economic growth and market share, and consequently, innovation must be excavated, nurtured, nourished, and allowed to flourish. Platforms, or rather the software-as-platform, are a way to accomplish this.7

7 Innovation and disruption sometimes make strange bedfellows. In his survey of Silicon Valley start-up founders, journalist Greg Ferenstein (2015) finds that Silicon Valley wants a “civil society completely oriented
The third reason is the ideology of Silicon Valley itself: technology, in Silicon Valley, has become suffused with the notion of freedom as "positive liberty" (Berlin 2002): the idea that technology, when put into people's hands, makes them free to achieve their (own) ends. Kelty (2014) argues that rather than seeing Silicon Valley entrepreneurs or researchers as libertarians, it makes more sense to see them as people who conceptualize software as a tool that people can use for self-actualization. As he puts it, what Silicon Valley wants is "not a substantive definition of freedom built into our technologies, but a new capacity for any kind of freedom" (2014, 218). "If there is something to be concerned about in Silicon Valley’s approach to liberty, it is not that it is overly libertarian, but that it is a kind of positive liberty imposed not through government action, but through the creation and dissemination of technologies that coerce us and that interfere with our goals. In this case, it is a set of technologies that has been designed to liberate (or coerce) the individual into being a freer, and more individual, individual" (2014, 209).

The process through which technology (and in particular, software) was imbued with this notion of freedom is a long one. As Fred Turner (2006) recounts it, it was cultural entrepreneurs like Stewart Brand and Howard Rheingold who connected the work being done toward innovation. They don't see conflicts between citizens, the government, big corporations, and other countries—just one big mass of people coming up with mutually beneficial solutions as fast as possible." The Democratic Party should beware, he says, for “tech elites love the Democratic Party in the same way they love the health care, transportation, and education industries—as a hodgepodge of aging leaders ripe for disruption.”
in Cold War laboratories across the US, particularly on computing—and especially the free-wheeling non-hierarchical practices of collaboration within these laboratories—with the emerging countercultural movements and their desire to free themselves from the military-industrial-bureaucratic system. It was these entrepreneurs who imbued the digital computer, otherwise a symbol of the hated system and the government, with the notion of liberation; the digital computer thus came to be seen as the way through which the counterculture could liberate itself from the hateful forms of bureaucracy (corporate or government) that it despised. Thomas Streeter (2010) argues that the Internet has been consistently experienced in the US through the lens of a very American notion of romantic individualism that he traces to Ralph Waldo Emerson; in particular, he notes in particular the experience of the cubicle dwellers in offices who experienced the Internet with a sense of power, immediacy and open-endedness, in a way that their bosses didn't.

An example of this notion of technology as positive liberty is seen in Thomas Malaby's (2009) study of Linden Labs and the making of Second Life. Second Life is a virtual world whose denizens live, in some sense, like they would in the real world: they meet people; make, sell, and buy artifacts, build houses, even dance in clubs (see also Boellstorff 2008). The architects of Second Life, Lindens, saw themselves as designing for contingency: they wanted to design "tools" that Second Life residents could take up and put to their own uses. As a Linden put it in an email: "We are a tool, a platform. We are plastic: users mold us as they feel" (2009, 110). The language of "platform" and "tool" thus appears here too: Lindens referred to Second Life as a platform for whatever purposes its residents put it to: dancing, gambling, buying, selling, and so on. Designers would build tools that they would give to Second Life denizens ("users") who would use it to govern themselves and each other. As Malaby puts it: "Lindens on the whole saw complex processes engaged by individuals pursuing enlightened self-interest as the legitimate path to self-governance" (2009, 104).
Malaby refers to this ethos of Linden Labs—the ethics of their design processes—as "technoliberalism": an “intense suspicion of vertical authority, a commitment to making technology universally accessible and beyond institutional control and a deep faith in the positive aggregate effects that follow from individual use of this technology for the purposes of creative expression” (Malaby 2013, 295). Technoliberalism is small-p politics, a politics pursued through other means, by building tools, and circulating them, without any kind of political labeling. Coleman (2012, 185-207) argues that this "political agnosticism" is crucial to the spread of technoliberal logics like open-source software—what Kelty (2008) calls its "modulation"—into other worlds like corporations and publishing. 8

To reiterate, the software-as-platform is software that is formatted in the language of tools and is built and circulated to cater to multiple kinds of actors who use it to realize their own goals (which may include building tools for still other kinds of actors). The whole assemblage has an emergent quality because its final purpose is left open-ended. At the same time, platform architects see the assemblage as the road to realize all three objectives

---

8 Dominic Boyer (2013, 134-36) has defined "digital liberalism," as the product of a "codeterminate dynamic" between “neoliberal political imaginaries [that] assume entrepreneurial and consumer subjects who are able to circulate effortlessly in zones of transaction" and “the personalized interfaces and lateral and mobile messaging capabilities of digital media enhance their experiential grounding and conceptual intuitiveness.” This is true but it depends on how vague or explicit the concept of "codeterminate dynamic" is. As Gabriella Coleman argues (2012, 185-207), hackers tend to be politically agnostic; this is reflected in how they describe their own work (e.g. building free and open-source software); as tools that can be put to use for open-ended goals. She argues that this is the reason why these "tools" can be fitted into various kinds of institutional frameworks including neoliberal ones.
simultaneously: market dominance, the management of innovation, and the realization of freedom. This open-endedness, I argue, needs to be taken seriously. Platform architects see the software-as-platform as open-ended because they hope it will bring forth new organizational forms and new socio-technical innovations. This is in accordance with the recent turn of late capitalism, the turn towards "the design of commodities that behave, communicate or inform, [...] by making them into processes of variation and difference that can allow for the unforeseen activities in which they may become involved or, used for which, may then act as clues to further incarnations" (Thrift 2006, 294-5).

1.2.2 Fast, Iterative, Production Processes
Because the software-as-platform is a site for innovation, and innovation is a matter of trial-and-error, according to the "hacker ethic" (Levy 2010), the software-as-platform and its various "tools" have to be constructed rapidly, in a process of fast, iterative development. Both platform architects and users build "tools" (i.e. pieces of software) rapidly: constructing a prototype, testing it with a section of users, and then re-iterating. But this process is not just confined to software: users who don't build tools but rather content (e.g. in Second Life or as we will see later, in MOOCs too) are also encouraged within the software-as-platform to use rapid design processes (by developing tools that allow them to do this!). The main advantage of these processes is that they allow failure to be detected early and thereby increase the chance of success. Part of this shift towards rapid design processes is a product of the success of user-driven software design, the so-called rapid action design framework or RAD (Mackay et al. 2000), also sometimes called the Agile framework.

The orientation of computer systems design has shifted in the last 50 years. The early years saw computer manufacturers and system designers concentrating on overcoming "hardware constraints." Through the 1960s on, "software constraints" became the primary focus on systems development, as hardware costs dropped and developers started to focus more and
more on inventing new types of uses for computer systems inside corporations. The constraints they attempted to overcome were the high costs of programming, the question of how programming should be managed as a labor process, the chronic shortage (from their perspective) of good programmers, and other issues (Friedman and Cornford 1989; Ensmenger 2010).

In the 1970s and 80s onwards, software design started to concentrate more and more on the "user" (Hales 1994; Mackay et al. 2000; Agre 1995; Bardini and Horvath 1995; Woolgar 1991). The definition of "users" has tended to be fluid and depended on organizational contingencies and access to users: thus, for software-makers, the users were often the managers of the unit whom they were designing the application for, rather than the so-called "end users" who actually used it for office-work. In Scandinavian software design traditions, the invocation of the user has been motivated by reasons of democratic governance: to give users (although again, here the question is often whether these are end-users or just representatives of end-users) more of a say in the design of applications intended for them.

In the United States, the incorporation of the user into the software design process has been more for market considerations: to make products that work better.

What the focus on the user has accomplished, certainly, is a change in software design processes. Rather than the traditional "waterfall" model of design which tended to emphasize linearity moving from a long gathering of user requirements to realizing them in the software, and then testing it, software developers now favor more of what they call an "Agile" framework which emphasizes designing in short bursts, building prototypes, getting feedback from the users on the prototype, which is then incorporated into the next prototype, and so on. The Agile process, developers argue, minimizes the chances of software failure because it makes the design process more in sync with user requirements.
1.2.3 Data-driven Decision-making

"Data" is often the residue—of tremendous value—of the software-as-platform, and platform architects argue that data, as much as possible, should be the basis of decision-making. In a recent issue of Harvard Business Review, MIT strategists economists McAfee and Brynjolfsson (2012) argue that "throughout the business world today people rely too much on experience and intuition and not enough on data." This sort of data-driven decision-making is seen as the counterpoint to merely intuitive decision-making, or HiPPO, highest-paid person's opinion. HiPPOs, the feeling goes, are too derived from gut feelings and tacit assumptions; decisions need to be more evidence-driven.

Data is related to decision-making through two crucial mechanisms: "analytics" and "A/B testing"; both mechanisms are designed into the software-as-platform. Analytics refers to a software infrastructure which provides its users with metrics and visualizations (sometimes called "dashboards") that allow the user to understand a particular kind of data (e.g. Google Analytics for web traffic data) that will then allow the users to think of future action. Analytics is action-oriented; it could be conceived as action-oriented social science research that is commodified into software programs to enable fast "data-driven" action.

A/B testing might be folded into the analytics framework as well but is worth considering separately given its widespread importance in the world of Silicon Valley. An A/B test on the World Wide Web is a randomized experimental comparison of a control and a treatment group in terms of their response to particular stimuli. Web companies like Google and Amazon use it to determine what features work best to maximize clicks and sales: from the order of search results to the right shade of blue for a particular interface (). In terms of its social organization however, this technique differs radically from the controlled experiment. First, Web companies use it at an almost unimaginable scale: Google has hundreds of A/B tests running continuously, involving millions of unaware users (Christian 2012). Second,
Chapter 1: Introduction

these tests can often be turned on with the click of a button and monitored in real time. Third, results are not published in journals; rather, they are encoded back into the design of the software, accelerating the production processes. Finally, these tests blur the line between research, marketing, and business decision-making.

Two things must be noted about data-driven decision-making. First, that this is again turned back into the language of tools. Platform architects and users are constantly encouraged to build tools for analytics and A/B testing and make them available to others; those others are encouraged to use those tools and put them into the service of their own goals. It is not just platform architects who use analytics to govern their users; users use analytics to govern themselves and each other; indeed, providing tools to make the users themselves data-driven is a key goal within the software-as-platform. Second, just as there is a core and periphery of the software infrastructure with a system of differential access between platform architects and users, there is a core and periphery to the data as well. Platform architects—who control the core of the software—get most access to the data. Users' access to the data depends on their position in the institutional ecosystem; the closer they are to the platform architects, the more data they can access. The kinds of data that are accessible by particular kinds of users thus inform the kinds of decisions they can use the data to make. The question of what data is most useful to a user is a thorny problem of governance within the software-as-platform.

A final point about data-driven decision-making is that it fits into the culture of fast, iterative design processes that I discussed above. The iterative design process revolves around the figure of the user; a feature or a design decision works if users are happy with it. But who speaks for the user is often contested; and when contestations happen, it is often "data" that is used to make the final decision, within the world of the software-as-platform. Fast-iterative design processes and data-driven decision-making then are two features of the same highly empiricist culture of the software-as-platform. Together, they produce an ideology of
innovation and production that is both never-ending and continuous: there is no perfect form the software-as-platform can take; rather, its form must be thought about, improved, and iterated upon as new actors are enrolled, taking into consideration the activities and beliefs of those actors.

1.2.4 Governance for Emergent Effects
Governance emerges as a particularly thorny problem within the assemblage, because the software-as-platform is supposed to be open-ended in terms of its ultimate purpose, but the goals of its architects and its users may often be in tension with each other. As Introna (2016) notes, "governance" is typically a good term to describe this tension, the question of how to regulate others' behavior, with "management" or "control" not being quite adequate for the description.

The two mechanisms (which often go together) used in the act of governing are: interfaces and algorithms. The case of algorithms has been most discussed; e.g. Google's PageRank algorithm (Brin and Page 2012) provides a relevance-based list of links after a user has entered a search term thereby serving as an obligatory passage point for a user's trajectory through the Internet. (Note however that the records of the trajectories of the literally billions of users however are all stored in Google's proprietary databases.) Similarly Facebook's EdgeRank algorithm (see Bucher 2012) computes how relevant Facebook posts are for a given user and only offers those that it deems most relevant. The other way of governing users is through interfaces: YouTube provides channels and playlists for users who access their site (unlike, say, Google whose home page is literally just a textbox), directing them to particular pieces of content. Other ways of governing through interfaces include providing extensive (and strategic) documentation or even visualizations (e.g. through "analytics" modules) that inspire users to act in particular ways. Providing APIs that make it possible to code and build tools into the software-as-platform is another way of governing. In Bucher's
(2013) study of external Twitter developers who make Twitter apps using its API, she found these developers conceiving of their own work as a step in the innovation value-chain; yet, they were also resentful of Twitter for its power to shape their work and livelihoods, which were at the mercy of what changes Twitter decided to make to its API. The Twitter API, Bucher surmises, is not just a tool for governing external developers, it is a tool for governing innovation itself; in other words, actors who control the API get to determine what counts as innovation.

A thorny feature of this governance is its conflict with the open-endedness of the software-as-platform. How to govern while still fostering open-endedness, how to govern while still nurturing innovation, is a problem for both platform architects but also for users. The popularity of “nudging” (Thaler and Sunstein 2009) as a regulative technique in public policy circles allows one way to negotiate this tension. In the “nudge” model of governance, which is inspired by findings in social psychology and behavioral economics, designers are seen as “choice architects” configuring the choices offered to consumers so that the most socially beneficial are most likely to be picked. Software has emerged as the key medium in which nudging techniques can be deployed easily and gracefully, partly because software is easier to modify and customize. Platform architects are intimately familiar with the nudging

9 E.g., designers of retirement plans could set the defaults for their employees plans so that they are automatically enrolled given that employees rarely change their defaults. Designers of school cafeterias might choose to put more nutritious foods in places where students are more likely to find them, and so on.
techniques of social psychology and behavioral economics popularized by policy-advisers like Cass Sunstein and journalists like Charles Duhigg (2014).

Introna (2016) argues, using the plagiarism detection software TurnItIn as an example, that the governance mechanisms animating the software-as-platform are best described using Foucault's notion of governmentality, the regulation of the "conduct of conduct." The process of governance happens in three steps: the determination of the problem to be governed (managing innovation, producing revenue, creating market share, some combination of all), the design of the technology itself (here the interfaces that shape access to the core of the software-as-platform), and finally, the folding back of this technology into the governed subjects themselves (external Twitter developers conceptualizing their work as agents of innovation).

1.2.5 Scalability
"Scale" is an aspiration of the architects of the software-as-platform. Engineers use "scale" in an arguably special way: scale means when something is both large and small at the same time, and elegantly so (Kelty 2000). Thus a small organization whose software caters to a large number of users has a technology that scales. A software-as-platform scales if the technical and institutional machinery that sustains it remains small enough, but the number and variety of users can grow without limit. Venture capitalist Paul Graham argues that a start-up only qualifies as being an innovative start-up if it can grow, i.e. it can expand the number of users continuously (Graham 2012). Even a design process only scales if it can accomplish the required objective (say, building a feature of the software) within a limited, and preferably small, time. Scale is thus a concept that is used for everything: from software and organizations to business models and production processes. Scaling the software-as-platform is a matter of configuring the social relations that constitute it. There are two ways of scaling: transforming the interfaces between different kinds of users, or building new sorts
Chapter 1: Introduction

of "tools" that integrate into the software-as-platform. Both techniques may either bring a new set of users into the ambit of software-as-platform or increase the output of an existing set of users.

Crowdsourcing has emerged as a key technique through which scaling is accomplished (by architects or users). Crowdsourcing is a method, in which designers break down a complicated task into simple "micro-tasks," channel these micro-tasks to various users by building interfaces, computationally aggregate and combine the "inputs" of these users (i.e. the solution of those microtasks), and thus produce the solution to the original complicated task. The process is sometimes called “artificial artificial intelligence” because the “intelligence” of the process is a result of the numerous humans, the microworkers, who are part of it. Crowdsourcing has become a critical component in the achievement of scale as well as the dream of artificial intelligence. Arguably, to be a constituent of a crowd—anonymous, functionally equivalent to everyone else (Felstiner 2011)—is the lowest in the hierarchy of users of the software-as-platform; typically, platform architects have been reluctant to grant any agency to crowds, allowing them no means of organizing or controlling the conditions of their own formation (Irani and Silberman 2013). Irani (2013) shows that the deployment of crowdsourcing makes certain kinds of users, the so-called "innovators" who draw on the interfaces to build workflows for intelligent tasks, more visible and lauded than other kinds of users, the people constituting the crowd themselves, who actually perform the microtasks that aggregate into the intelligent task; thus crowdsourcing ends up doing another kind of cultural work (besides the accomplishment of scale) of highlighting one kind of user over another.

1.2.6 Personalization

Personalization and scalability are two sides of the same coin. If the software-as-platform is to scale to accommodate more and more users, of different kinds, then it must, platform
architects believe, also be able to differentiate, at finer and finer levels, between these users. This is called "personalization." Personalization is also accomplished, like governance (and also scaling), through the simultaneous application of algorithms and interfaces.

The most common method through which personalization is accomplished is called "machine learning." The primary technique of machine learning is building a mathematical function that can discriminate or classify between particular items and embedding it in software. For example, a machine learning algorithm can identify whether the given image contains a face (or an elephant, or even Bill Clinton). To do this the algorithm needs to be "trained"; it is therefore shown many different examples of images with faces from which it then creates a mathematical rule that allows it to discriminate between images with faces and those which do not, and this rule is then deployed in a program (for e.g. Facebook's image processing system). This is not to say that humans have no say about these rules. Indeed, many of the decisions about the kinds of statistical methods to use inside the algorithm, the number of parameters in the classifying function, and of course, the “inputs” of the algorithm, are all decided by humans using contingent criteria. But while humans can "tune" the system, the bulk of the work of "learning" to recognize faces is accomplished by the statistical algorithm that “fits” a particular function to the training data.

Debates about algorithms are often, at their core, debates about the appropriateness of the particular machine learning technique under use: sometimes because corporate secrecy forbids the algorithm from being disclosed (Pasquale 2015), the algorithm has an inherent opacity (Burrell 2016), or about whether the results of the algorithm are unfair or discriminatory. The ability to embed machine learning programs within software infrastructures, and thereby categorize users based on their activities into finer and finer-grained categories, has led some analysts to label this phenomenon as "soft biopolitics": a new form of control where cybernetic algorithms are used to manage populations by
Chapter 1: Introduction

segmenting them into finer and finer levels (Cheney-Lippold 2011). Others have pointed to other effects of personalization: the ability, by categorizing, to link two hitherto unrelated individuals together to create "calculated publics" (Gillespie 2014); the ability to link individuals and resources (songs, commodities) together by computing "similarity" metrics (Ananny 2016), and so on.

In the next section, I describe "platformization," which I define as a process through which the software-as-platform is drawn on to reconfigure institutions.

1.3 Platformization

Taking MOOCs and MOOC infrastructures as its example, this dissertation argues that various actors, either from Silicon Valley or those inspired by the successes of Silicon Valley, draw on the software-as-platform to reconfigure various institutions, often those that they are themselves part of. In other words, the software-as-platform, as both a material entity, as well as a locus of particular normative practices (fast production processes, data-driven decision-making, governance for emergent effects, scalability, and so on), becomes a focal point around which debates about the methods and ultimate goals of particular institutions are conducted. Thus MOOC infrastructures become loci around which the question of "how best to teach and learn" is debated between different actors, with all three terms, "best", "teach" and "learn" as points of debate. Within the news industry, the question might be: "how best
to create and deliver news," where fundamental terms are being debated and reconfigured (Boczkowski 2004; Boyer 2013). These debates are both discursive and practical, in that they are conducted as explicit debates or through proxy, as for example, if a group of actors builds an analytics dashboard and makes it available within the institution. I call this process through which different actors deploy the software-as-platform within a project of institutional change as “platformization.”

In this section, I investigate three questions: what makes the software-as-platform so malleable that it can be ported by actors into new institutional contexts? Who are the actors who do this porting? Finally, what kinds of institutional changes is the software-as-platform most likely to produce?

Malleability: I argue that it is the open-endedness of the software-as-platform that gives it its special malleability that it can be transported into different contexts and institutions: the framing of the assemblage as something that empowers particular actors, as a mere "tool" that

10 In this section, I describe what I call the process of "platformization": the mechanism through which certain actors deploy the software-as-platform to reconfigure institutions. Anne Helmond (2015) has used the term platformization to describe the transformation of social media websites into social media platforms. This transformation has meant that the Web is now controlled by a few dominant Internet companies that serve as obligatory passage points for Internet users and collect data about these users in their own proprietary databases. Helmond finds that this has become possible primarily because of new forms of interfaces (GUIs and APIs) that these companies have successfully built; as she puts it, these interfaces have allowed these companies to decentralize their features onto other websites and recentralize the data that they collect about users from those websites into their own proprietary databases. My own use of platformization is less about the Web and more about how the software-as-platform is being used to transform institutions. What is similar in both my account and Helmond's is the key role that interfaces play; and the asymmetry between those who design those interfaces as opposed to those who merely use them.
can be put to the actors' own uses. Thus within MOOCs, the platform architects see their goal giving tools to their users for "better" teaching; the "better" is made possible by tools that are data-driven or those that allow experimentation or A/B testing, but the use of these features is left to the individual institutions and instructors as they deem fit. The open-endedness is helped by another aspect: these tools are rarely about automation, or an attempt to take away what actors do and give them to machines. Rather these tools are about—or framed as being about—empowering particular actors. By designing software as tools, tool-makers let human beings, i.e. users, fill in the gaps in the software with their own meanings, goals, and methods. Ekbia and Nardi (2014) label this particular configuration of humans and programs working together as "heteromation" wherein humans are “fashioned as computational components” with their outputs connected to programs. In this arrangement, humans take up the slack when programs fail (i.e. when there are no universal rules to decide what is to be done) and use local contexts to make particular working choices. On the other hand, programs are fashioned as tools that allow humans to do what they could not on their own; they are supposed to be empowering.

Actors: Institutional actors who promote the software-as-platform to transform their institutions usually have links to the world of Silicon Valley. Thus the primary agents of MOOCs have been computer scientists at MIT, Stanford and Berkeley, who have a fundamental interest in teaching and learning, a desire to transform higher education, as well as links—through consulting, internships, and students—to the world of Silicon Valley. In his study of how American newsrooms have drawn on the Internet, Pablo Bacikowski (2004) shows something similar: media companies that had fewer ties to print were able to transform themselves far more and adapt to the Web than those that had more history with the print medium. The companies that were able to take advantage of the Web had two main characteristics: their web newsrooms were less tied to their print newsrooms, and
consequently, actors in web newsrooms were able to re-imagine both the editorial functions as well as the audience.

Consequences and Mechanisms of Institutional Change: The use of the software-as-platform to transform institutions impinges on two aspects of institutions: institutional roles and institutional priorities. As we saw, the software-as-platform is infused with notions of empowerment and innovation. Institutional roles thus get framed—transformed, even—as innovative roles: thus the innovative teacher, the innovative reporter, the innovative entrepreneur and so on. Innovation may be framed in various ways. For instance, an innovative role might be imagined as one that involves creative use of data (through the software-as-platform) for organizational decision-making. Or an innovative role might involve making tools that allows other roles to become more innovative.

Platform architects are also able to frame institutional roles as "users" of the software-as-platform. Users are configured, as we have seen, through interfaces that allow them to access what is "under the hood" i.e. various aspects of the software (various levels of code, various levels of data). Depending on what kinds of access are possible, there is an asymmetry in the kinds of users/roles; thus, some are viewed as more innovative than others. One of the starkest examples of this comes from Amazon Mechanical Turk (a software-as-platform built especially for crowdsourcing) where entrepreneurs and innovators (who write some form of code) are privileged over the click-workers whose work the code calls upon but whose labor is invisible, unseen, and cast as profoundly un-innovative (Irani 2013).

A (perhaps unintended) consequence is the primacy of software. Even within the so-called innovative roles (as opposed to routine ones), those that are about making tools (that will then promote further innovation) are considered higher-status (see also Malaby 2009 and Boellstorff 2008 on Second Life). As users govern themselves and make an effort to be innovative, they are more and more likely to make interventions within software that can
propagate further into the software-as-platform. The innovative role becomes closer and closer to a role that emphasizes the making of software.

1.4 A New Form of Expertise: Tool-Making

The third and final contribution of this dissertation is the finding that the spread of the software-as-platform and its attached norms has led to the emergence of a new form of expertise: the expertise to build software tools (the software-as-platform itself, or the tools within the software-as-platform), that is considered both a privileged form of innovation, as well as one that helps create innovative actors by empowering existing roles.

What does this expertise consist of? What does it take to build a tool? First, and foremost, it takes the ability to write code (or pay someone else to write code). Second, and perhaps more important, it requires access to the core of the software-as-platform, which in turn depends on the tool-maker's institutional position within the platform ecosystem.

But writing code is not the only constituent of this expertise. The claim to innovation rests not just on the fact that the tool empowers but that it comes embedded with insights from other, relevant experts and forms of expertise, the so-called "domain experts." Thus tool-makers within the world of MOOCs seek to empower learners and teachers but they also hope to govern the behavior of these actors by guiding them—nudging them really—into practices deemed effective and relevant by educational researchers.

I situate the emergence of this form of expertise within the social studies of expertise (Abbott 1988; Collins and Evans 2008; Suryanarayanan and Kleinman 2013; Jasanoff 2007; Carr 2010). In particular, I draw on Gil Eyal's (2013) suggestion to transform the sociology of professions into the sociology of expertise by looking not just at the jurisdictional contests between professional groups, but at the actual emergence of "tasks and problems," a phrase he draws on from the seminal work of Andrew Abbott (1988). Eyal wants social scientists to
understand the "networks that link together objects, actors, techniques, devices, and institutional and spatial arrangements" (2013, 864) that make possible routine expert utterances and acts. I thus show that the emergence of the software-as-platform as well as the accompanying ideologies of innovation and disruption is crucial to the emergence of this new form of expertise. I also show that this new form of expertise is tied to institutions that give it credibility: of the new experts, those who have close ties to the core of the ecosystem are more likely to have impact than those who do.

1.5 Theoretical Contributions
This dissertation thus makes two core contributions to the critical studies of software, algorithms, and code. It offers an answer to the question: where does the agency of code or software lie? What does it consist of? It also offers an answer to the question: who are users and where does the agency of users lie?

Wendy Chun (2013) has argued that in late modernity, source code is fetishized; it has become "logos," a symbol with the promise of self-present, transparent, meaning. The source code of a program aligns a process that plays out in time (i.e. when the program "executes") into one in space (the lines of the code itself). The reason source code can be so fetishized, the reason source can be equated with action, Chun argues, is because the labor of executing the source code has been systematically deleted and hidden (and typically, within the history of computing, the execution of code required labor that was typically gendered female). Source code has thus come to stand in for itself rather than for the social relations that produce and structure it; source code makes the programmer into the author who gives meaning to the text but this is only possible by obfuscating and hiding the labor required for the execution of the code. Programmability—the ability to write source code—is a key way of producing the sovereign neoliberal subject.
This dissertation adduces evidence that largely agrees with Chun's main point: that source code is not the be-all and end-all of software, and the social studies of software would do well to concentrate on the social and political relations that shape the execution of source code, rather than fetishize source code itself. The privileging of users who make "tools" (i.e. author source code) within the software-as-platform over those who do not is evidence of what Chun calls "sourcery," the fetishization of source code. But this dissertation also shows, that within the software-as-platform, it is access to and the ability to modify and propagate the "core" source code that determines who has the most agency. In other words, it is not source code in general which matters, but particular pieces of source code. Access to particular pieces of source code is determined less by virtue of being the sovereign programmer, the fearless hacker, and more by institutions that control the interfaces to the software-as-platform. In other words, institutions matter; programming has agency only when programmer and institutional objectives align.

This dissertation also contributes to an understanding of the software "user" (or a computer user in general). The agency of the technology user has been a key issue for STS. While some authors have emphasized how technical artifacts have their own "script" which has written into it strong preferences for certain kinds of decoding (Latour 1994; Akrich 1992), others have emphasized the wide latitude of decoding that users end up doing and how it shapes the artifact itself (Oudshoorn and Pinch 2005; Kline 2000). When it comes to the design of computers, Woolgar (1991) showed that the boundary between the designer and the user corresponds to the boundary between the inside and outside of the designed computer which in turn corresponds to the boundary between the organization that makes the computer and the world outside. Users are those that have no access to the inside of the machine while designers do.
Woolgar's definition, while still somewhat right, has become a little more complicated when it comes to software. Unlike hardware, software can be rewritten and customized; in fact, most of the jobs involved with software rarely involve writing code as much as they involve customizing, gathering requirements, and rewriting. The hard line between the context of production and the context of use, which Woolgar observed for hardware design, is considerably softer when it comes to software (Pollock, Williams, and D’Adderio 2007; Mackay et al. 2000). As we have seen, the user came to have a key role in the software design process itself, primarily out of market considerations (Friedman and Cornford 1989). Hales (1994) lists three ways in which users play a role in the design process: users as clients, users as co-designers, and users as actor-constructors.

When it comes to the software-as-platform, except for those who design the "core," whom I call architects, everyone else is configured as users through the interfaces (GUIs and APIs) that give them access to particular parts of the software. The agency of users is thus determined, for the software-as-platform, by three factors: an ideology of innovation that privileges certain users as more innovative than others (typically based on whether they produce more software tools), by the ability of the users to manipulate the source code they have access to, and finally, by their proximity to the core of the "ecosystem" which determines the impact (in terms of circulation) of their code/software. The agency of users is relational; once again, institutions matter.

1.6 Methodology
The process of building and using the software-as-platform has multiple dimensions, taking shape at different times and places and involving multiple actors—the architects of the platform, and the multifarious actors, working individually or within institutions, with varying degrees of access, who are constituted as users. In this dissertation where my primary object of study was the edX infrastructure, this involved the core software engineers.
at edX, instructors and support staff at participating universities, external developers working in other software companies, and the associated network of researchers found mostly in universities. Moreover, the activities of these various constituencies have a dialectical relationship, giving the entire system an inherently emergent quality. I adopted a multi-sited approach (Marcus 1995; Marcus and Fischer 1986) that encompasses each of these spheres of activity, while also examining the way they interact with and influence each other over time.

I give a more detailed description of my sites and my methods in Chapter 2 but I briefly want to reflect on my methods in this Introduction. Briefly, I used four main data-collection methods in this dissertation:

**Long-form interviews:** I conducted almost 70 formal interviews with various actors which included both platform architects, and various kinds of users (developers, instructors, researchers, support staff, in the case of MOOCs). Sometimes an interview would be my only interaction with an actor; at other times, I had observed these actors, sometimes extensively, in other settings (e.g. meetings, or mailing lists). After experimenting with various formats, I settled on the professional biography. I asked my actors what led them to work on the software-as-platform as architects or users, their institutional and technical trajectory across organizations and projects, the central challenges they faced, how they overcame, or tried to overcome, those challenges, and some of the specific socio-technical choices they made in doing so. These biographies gave me a glimpse of two things: the very different motivations that actors brought to their work, and how institutional settings shaped technical choices.

**Meetings:** I observed a variety of in-person meetings, more than a hundred instances. My role in these meetings was often that of an observer, and if called upon, I would give my opinion on a topic. These meetings were either between platform architects; or users (of various kinds); or between architects and users. Meetings provided me with a way to observe
in-situ the challenges that a project encountered and how those challenges were interpreted by the actors. These challenges helped me execute the process of infrastructural inversion (i.e. seeing what actors take for granted) that is key to this dissertation. Most importantly, meetings allowed me to see the disagreements between different actors and to focus on the causes of these disagreements, which in turn, led me to fundamental issues of socio-technical organization, asymmetry, and expertise.

Conferences: Conferences served a similar function as meetings except that they had far larger numbers of people present. They allowed me to determine both "matters of fact" and "matters of concern" for my actors as well as trace the points of contention between them. Conferences also provided me with a fascinating glimpse into front-stage/backstage processes: how a paper or a presentation would be officially received, and how it would be discussed in private.

Circulating documents: These included emails on mailing lists, documents and notes from meetings, research papers, blog-posts, and other forms of communication. In addition, the fact that the edX software is open-source allowed me to access its portal, its published documentation, as well as other kinds of technical discussions.

A note on learners: I have approached MOOCs as an infrastructure that is being used to reimagine an institution—the university. However, I leave out one important group which plays an important role within MOOCs: students (or learners as they are referred to). Students how enter my analysis in the ways they are imagined by actors like teachers, programmers and learning scientists; they are also represented in the form of data as objects of experimentation and knowledge production. One key reason for this is logistical: MOOC students are often distributed all over the world, and often have very different backgrounds and interests. Studying them using qualitative research methods was not feasible for this
dissertation. However as time has gone by, more and more findings about MOOC learners have emerged (e.g. see Ho et al. 2015); this dissertation accordingly draws on these findings.

1.7 Plan of the Dissertation
This rest of this dissertation consists of 4 chapters. Chapter 2 describes the history of MOOCs, arguing that rather than seeing MOOCs as prophetic and signifying the destruction of the public university, they need to be seen as an experimental assemblage being put to a variety of uses. Chapter 3 takes the edX infrastructure in particular, and shows how it is an example of the software-as-platform. In Chapter 4, I focus on the process of platformization, i.e. the debates over teaching and learning as they played out around the MOOC infrastructure. In Chapter 5, I focus on the new form of privileged expertise emerging around the software-as-platform, around the making of tools; specifically I focus on the ideologies and work practices of computer scientists who look at MOOCs both as an opportunity for tool-building as well as making knowledge claims about teaching and learning.
Chapter 2: MOOCs as an Experimental Moment

2 MOOCs as an Experimental Moment

What are MOOCs? MOOCs, or Massive Open Online Courses, emerged in 2011 when a group of computer science faculty members at Stanford—most prominently, Sebastian Thrun and Andrew Ng, researchers in the field of Artificial Intelligence or AI—decided to create a version of their courses for full interaction by anyone with an Internet connection, taking as their inspiration a range of Internet websites dedicated to vague-to-explicit forms of learning: Khan Academy with its mathematics video tutorials, YouTube and Lynda.com with their extensive range of user-generated how-to videos, and StackOverflow with its crowd-sourced format of having users answer each others' questions (Ng and Widom 2014). Not surprisingly, these MOOCs resembled these websites in look and feel, and comprised lecture videos, simulations, interactive tests, assessments with automatic grading, discussion forums, and even a certificate for those who finished. It was an audacious experiment: a full Stanford class being put out in the wild for anyone with an Internet connection.

It isn't clear whether these faculty members expected their experiment to generate the interest it did. But it did, culminating in the New York Times labeling of 2012 as the "Year of the
MOOC" (Pappano 2012). And the computer scientist-entrepreneurs ran with it, abetted by a confluence of interest groups with varying agendas: venture capitalists eager to cause the next "disruption" in the higher education field (Christensen and Eyring 2011; Carey 2012); elite American universities like MIT, Stanford, Harvard, Berkeley, Michigan, Princeton, and Penn, equally determined to stay on top of this disruption and possibly engineer it themselves; governments, both federal and state, as well as charitable foundations eager to solve the "crisis" of American higher education (Selingo 2015); and finally, a host of experts who felt that they had the talents, wherewithal, and knowledge to contribute to the experimental project (whatever the project was). These experts included, among others: computer scientists and software experts eager to build new interactive tools to "scale" education as well as generate knowledge claims about teaching and learning, social psychologists interested in questions of meta-cognition and motivation; learning scientists and other educational researchers eager to advise MOOC-builders with their hard-earned findings about the process of learning; and finally, instructional designers and other participants in the e-learning industry, till then the sector where online learning had primarily existed, who were both thrilled that online learning was receiving such attention and terrified that they would be shut out of the conversation. And of course, one cannot ignore the learners who have flocked to these MOOCs with their own mysterious goals: professional retraining, self-edification, making new contacts, dipping into a new topic of interest, or even learning a new language (Kizilcec and Schneider 2015).

In this chapter, I will step back and present a brief analytical history of MOOCs and developments that preceded them. This chapter serves as a background to the subsequent chapters which investigate MOOC infrastructures in more detail.
2.1 A Brief (Critical) History of MOOCs

In an article on August 2, 2011, the blog "Wired Campus" reported that Stanford University's Sebastian Thrun and Google's Peter Norvig, both experts on Artificial Intelligence, had opened up their Stanford course on the topic, to the whole world (Zou 2011). The course, called "Introduction to Artificial Intelligence," ran simultaneously at Stanford and to the outside world from October to December (i.e. in Stanford's fall quarter). The article reported that the class, which usually enrolled around 200 students each term, had received more than 8000 queries about it (these students had asked to be put on an email list). The course ultimately ended up giving certificates to 35,000 students (and 3 million are said to have browsed the class in some form). This is the class that became the foundation of the start-up Udacity. That same fall, Stanford also put up two other courses for free online: Introduction to Databases (by Jennifer Widom) and Introduction to Machine Learning (by Andrew Ng). It planned to introduce 8 more courses in January 2012, including a course from Berkeley called "Software as Service". Ng's "Introduction to Machine Learning" course eventually became the MOOC start-up Coursera (which he founded with Daphne Koller).

The introduction of these free classes seems to have been the result of a considerable amount of behind-the-scenes activity and competition at Stanford itself. In a retrospective history

---

written in 2014\textsuperscript{12}, Andrew Ng and Jennifer Widom (as the instructors of the first two classes from Stanford) aver that:

\begin{quote}
contrary to popular opinion, MOOCs were not an "overnight success": The idea of highly scalable education had taken years of germination, false starts, and experimentation, culminating finally in the three highly-visible Stanford offerings. (Ng and Widom 2014)
\end{quote}

In particular, they credit the following people:

These MOOCs drew from a wide variety of ideas developed by a community of Stanford and non-Stanford researchers interested in online education. Key members of the Stanford group included Daphne Koller, who had been experimenting with and evangelizing blended learning (the "flipped classroom") at Stanford for several years; John Mitchell, who led a team developing an on-campus learning management system; Bernd Girod, whose students had developed sophisticated lecture-recording technologies; as well as Dan Boneh, Steve Cooper, Tiffany Low, Jane Manning, and Roy Pea, who contributed significantly to early discussions.

The account also hints that this "community of Stanford and non-Stanford researchers interested in online education" was animated by a sense of rivalry as much as the spirit of cooperation. Thus, the first course by Thrun and Norvig ran on a software infrastructure that was built by Thrun and his collaborators; while the first two courses by Ng and Widom ran on software built by Ng and his graduate students. The decision to spin off Coursera into a for-profit, venture capital-backed start-up seems to have been inspired by Sebastian Thrun's decision to turn Know Labs (which was credited as developing the software for his first AI

\textsuperscript{12} The retrospective was written as part of a larger policy report on MOOCs authored by Fiona Hollands and Devyani Tirthali from Teachers College (see Hollands and Tirthalli 2014).
MOOC) into Udacity, also a for-profit venture-funded start-up. The two announcements happened in rapid succession: Udacity was announced on January 23 2012, Coursera on January 31 2012.¹³ Even as Udacity and Coursera spun out of Stanford, Stanford came up with its own online portal, Stanford Online, to offer MOOCs using another homegrown software called Class2Go ("Stanford U. Releases New Open-Source Online-Education Platform" 2012); faculty like Jennifer Widom chose to keep their classes on Stanford Online rather than join Coursera and Udacity even as others, like Scott Klemmer, taught their courses on Coursera.

The events happening at Stanford sent a tremor at MIT, Stanford's traditional rival when it came to the production of technology and technology professionals. On Dec 19 2011, MIT announced that it would create a separate platform called MITx which it would use to deliver MIT courses. Anant Agarwal left his position at CSAIL to lead MITx. He and then MIT Provost Rafael Reif (who had championed the project internally, according to many people) went out of their way to emphasize that in taking this step, MIT was drawing on its previous efforts like Open CourseWare (OCW, a project successfully launched in 2000 to provide course content like syllabi and lectures openly on the Web), rather than just responding to

¹³ The announcement of Coursera as a for-profit start-up surprised the two Berkeley faculty who had been recruited by Ng to teach the next round of MOOCs for Stanford Online. One of them, Armando Fox, chose to go ahead and teach his software engineering class on Coursera, while Peter Aibeel chose to withdraw. Both Fox and Aibeel taught their classes on edX once Berkeley became a formal member of the edX consortium.
Stanford. However, MITx would be different from OCW in that its instructional materials would be made for the Web, rather than just being a recordings and syllabi of in-person classroom courses. And unlike OCW, MITx would have actual problems and assessments for students to solve, which would be graded automatically through programs (Solomon 2011). Some analysts suggested that this meant that MIT (a school that had no distance learning or professional education program) was going for a three-pronged approach to credentialing: MIT degrees for its residential students, MITx certificates for its online learners, and OCW courseware for others. To avoid the conclusion that MIT was reacting to Stanford, Agarwal also pointed to one of his colleagues Rob Miller who had been experimenting with using online techniques in his classroom teaching for many years, just like his Stanford colleagues. Miller, dealing with the increasing number of enrollments in his Introduction to Computer Science class at MIT, had built a system wherein students (but also alumni) could comment and give feedback on other students' code, thereby relieving some burden on the TAs while also producing better learning outcomes for the students themselves (Tang 2011).

The first course on MITx was a Web version of 6.002, an introductory class on Circuits and Electronics. Called 6.002x, it was taught by Agarwal, Gerald Sussman, Chris Terman and
Piotr Mitros\(^{14}\), who led the design the original software that hosted the course. MIT News reported that "in the end, almost 155,000 people registered for 6.002x. Of those, roughly 23,000 tried the first problem set, 9,000 passed the midterm, and 7,157 passed the course as a whole" (Hardesty 2012). On March 15 2012, Anant Agarwal announced that he was stepping down from his post as the head of CSAIL in order to lead MITx fulltime (Solomon 2012). Shortly after this, in May 2012, MITx became edX. On May 2 2012, Harvard and MIT together announced the creation of a new organization called edX (what had previously been called MITx). Each gave 30 million dollars to this new organization which would be non-profit and open-source (unlike Coursera which was for-profit, relied on venture capital and had a proprietary software platform). MITx remained but as an organization, it would concentrate on building high-quality MIT courses that would be hosted on edX's portal, edx.org. It would also focus on bringing new digital techniques to MIT's on-campus courses. MITx was later renamed as the Office of Digital Learning or ODL.

By the summer of 2012, three new start-ups—Udacity, Coursera, and edX—existed that offered free online college-level courses to anyone with an Internet connection. Of these, Coursera and edX partnered with universities, while Udacity partnered with individual

\(^{14}\) Mitros often speaks of himself, e.g. in the edx-code mailing list, as the architect of the original MITx/edX software. He also worked with Sebastian Thrun on the original Know Lab platform.
faculty members (who could be academics or others). In the rest of this dissertation, unless otherwise mentioned, I focus on edX and Coursera.

Two points must be noted about the origin story described in the preceding paragraphs. First, that while other software infrastructures had indeed been used for teaching and learning, both at the K-12 and the college level—for example, the relative success of Intelligent Tutoring Systems through the Carnegie Learning product (Corbett and Koedinger 1998), or the use of Learning Management Systems (LMSes) like Moodle, Sakai, and Blackboard (see Dede and Richards 2012)—the architects of MOOCs were not looking at these at all. Instead their eyes were focused squarely on the new "platforms" of Silicon Valley, most of all on Khan Academy, the website for school students started by young wunderkind Sal Khan with generous funding by the Gates Foundation, that featured tutorial videos (now sometimes called Khan-style videos) and practice exercises. They were also inspired by new forms of interaction on the Web: the way readers responded to each other on the chat forum StackOverflow, or the new kinds of how-to videos popping up on Lynda.com and YouTube. The discussion forums of MOOCs explicitly draw on the design and form of StackOverflow (Ng and Widom 2014; see also Reich 2014).

Second, they were already thinking fluidly about the boundary between formal and informal learning, the learning that took place in institutions of higher education versus learning that
Chapter 2: MOOCs as an Experimental Moment

took place outside them. In this pursuit, there were two strains of thought. First, that learning itself would become ubiquitous and invisible. As a report issued by the Computing Community Consortium argued, the success of MOOCs meant that

\textit{today's highly visible, accessible, and large-scale efforts may help drive education into an oft-touted and idealized “hiding-in-plain-sight” ubiquity, blurring boundaries between formal and informal education.}^{15}

This is expressed in the notion that forms of learning like how-to videos on YouTube and Lynda.com are now considered on a continuum with lectures delivered through video interfaces. And second, this wish was expressed around the vision that that higher education itself might be reconstituted around what is sometimes called twenty-first century learning. Thus Sal Khan proposes that at least one model of the university of the future must let students move fluidly between academic and corporate settings, between school and the world outside. This quote is instructive:

\textit{Imagine a new university in Silicon Valley—it does not have to be there but it will help to make things concrete. I am a big believer that inspiring physical spaces and rich community really does elevate and develop one’s thinking. So we will put in dormitories, nicely manicured outdoor spaces, and as many areas that facilitate interaction and collaboration as possible. Students would be encouraged to start clubs and organize intellectual events. So far, this is not so different from the typical residential college. What is completely different is where and how the students spend their days. Rather than taking notes in lectures halls, these students will be actively learning through real-world, intellectual projects. A student could spend five months at

---

\textsuperscript{15} The report can be read here: \url{http://archive2.cra.org/ccc/files/docs/meetings/OnlineEducation/CCC-MROE-Report.pdf}.\textsuperscript{66}
Google optimizing a search algorithm. She might spend another six months at Microsoft working on human speech recognition. The next four months could be spent apprenticing under a designer at Apple, followed by a year of building her own mobile applications. Six months could be spent doing biomedical research at a start-up or even at another university like Stanford. [...] 

All of this will be tied together with a self-paced academic scaffold through something like EdX (Harvard, MIT, and Berkeley’s “MOOC”) or Khan Academy. Students will also still be expected to have a broad background in the arts and deep proficiency in the sciences; it will just be done in a more natural way. They will be motivated to formally learn about linear algebra when working on a computer graphics apprenticeship at Pixar or Electronic Arts. (Khan 2013; an extract from Khan 2012, his memoir)

Criticisms of MOOCs often fall into two main categories: political economy and models of pedagogy. Many analysts have seen MOOCs as the latest salvo in the perpetually ongoing battle between neoliberal capital and labor (e.g. Bady 2013; Bady 2012). As the events of the last few decades have unfolded, U.S. higher education is now widely seen as being in crisis: from matters as diverse as the high price of college (Archibald and Feldman 2010), cutbacks in state funding for public universities and community colleges despite high demand (Newfield 2011; Bady and Konczal 2012) and the inability of universities to provide for working adults who attend school part-time, now the majority of higher education students (Deil-Amen 2015). Partly because of their flexibility suits non-traditional students, for-profit schools have seen an increase in enrollment; yet their high cost and poor job placement rates are largely responsible for increasing student debt (Mettler 2014). Finally, there is the increasingly-publicized finding that higher education students don’t learn (Arum and Roksa 2011)—a sign perhaps that the accountability revolution may have reached higher education. In parallel, there is a stream of discourse lamenting that the United States does not produce enough trained technical professionals to sustain its knowledge and innovation-driven economy and may fall behind its international competitors. In this pessimistic landscape, the emergence of MOOCs in the heart of Silicon Valley has signaled to many commentators a further erosion of public higher education and a commitment to reconfiguring higher education to the needs and purposes of Silicon Valley. In other words, higher education was
to be about job training rather than, say, a liberal education (see Veysey 1970 for different conceptions of the American university). As Aaron Bady put it in his widely read piece in 2013:

> [T]he educational-industrial complex is the foundation on which [Silicon Valley] rests, where it’s pretty normal for a Stanford professor to also be an executive at Google, and for a university president to see his duty as split between working for education and working for industry. But things get weird if that model starts to be the basis from which to transform a public system of higher education. Which is what’s now happening.

Mixed with this critique of political economy was a critique of MOOC pedagogy, although sometimes it was hard to see where one began and the other ended. For e.g., Bady, in his piece that I quoted above, critiqued the MOOC’s reliance on videos, arguing that the replacement of the one-hour lecture by a bite-sized YouTube video meant that an extended piece of argument has been replaced by a 600-word New York Times op-ed. The “logic of the MOOC,” he lamented, “is a function of shallow thinking, of arguments that go no deeper than a David Brooks or Thomas Friedman column. But they also valorize and reward that level of depth, even make it compulsory.” Other critics have also harped on the videos and the multiple-choice questions, although more from their stance as experts on pedagogy. Videos, they argued, are a passive mode of consumption where the logic of the lecture is replicated and the learner is fed information by the instructor. Rather than passively consume videos, the learner should be doing something, most importantly, practicing new examples (see Barba 2015; Thille et al. 2014). Another critique came from the fact that video watching in MOOCs was a solitary activity. Learners, critics argued, were better off talking to other learners, something that wasn’t possible while just watching videos (McConachie and Schmidt 2015).

Many analysts have seen the MOOC launch as a kind of prophetic act. Certainly the self-aggrandizing statements of the MOOC founders have not helped here. These claims have ranged from statements about democratization (that MOOCs will expand access to higher
Chapter 2: MOOCs as an Experimental Moment

education drastically not just in the US but also the Third World where they will help produce the "next Steve Jobs," see Koller 2012) to the claim that MOOCs are the "particle accelerators" of learning and will generate data that will be the basis of a new and revolutionary computational science of learning (Stokes 2013). It is tempting to see in these statements a master plan to disenfranchise public higher education, build a two-tiered higher education system of elite universities that produce content and everyone else who uses them, create new forms of surveillance around educational data, and connect the form and purpose of higher education even more firmly to the interests of Silicon Valley. And certainly, it's not clear if those might not be the eventual outcomes. This dissertation argues instead that it's best to see MOOCs as a kind of giant experiment that is being enacted out in the wild, with considerable "interpretive flexibility" (Bijker, Hughes, and Pinch 2012). The act of releasing the Stanford courses into the wild needs to be seen as an expression of the lets-try-to-see-if-it-works spirit that this dissertation argues is a key value of the software-as-platform, that MOOC infrastructures are an embodiment of.

In the next subsection, I try to show the interpretive flexibility of MOOCs by taking up an often-made comparison between the MOOCs started by Stanford (the so-called xMOOCs), and the blog-based MOOCs from 2008, the so-called cMOOCs.

2.2 xMOOCs versus cMOOCs
One reason to see MOOCs as an experimental moment is to reflect on how the name MOOC itself came to be. In 2008, a group of Canadian researchers and teachers working at distance-learning universities in Canada, who had been working to leverage new forms of Internet interactivity in their teaching, opened up one of their courses both to a set of officially enrolled students as well as to anyone else who would participate; they ended up calling these courses as MOOCs. "Course" would be a loose way to describe what happened; "non-course" might be a better description. Essentially, every week, the instructor would begin
with a question that would be sent out to the students (over email, published as a blog-post, on Twitter etc.). Students would try to answer these questions, again, using a medium of their choice. Finally, at the end of the week, the instructor would compile all the responses and aggregate them and encourage students to comment on and discuss each other's work. There were no learning goals as such, or even anything that could be called "content." Rather content was produced by the students and instructor together; the instructor's attitude towards students was: "You choose the manner in which you participate in the course. If you just want to follow along and read the newsletter, that's fine. If you want to set up a blog or a feed of some sort, and contribute, that's fine." (From Hollands and Tirthalli 2014, see 31-33 for a narrative).

This MOOC experiment from Canada was written up in the Chronicle (Parry 2010), attained some fame within distance learning circles, and allowed some of those involved to attract new sources of funding. But it remained unknown to most of the academy. Yet, when Stanford announced its open online classes in the summer of 2011, George Siemens, the most savvy amongst the Canadian researchers, told Tamara Lewin of the New York Times that what the Stanford professors were doing was a continuation of their own experiments with open teaching, though with lecture-videos and assessments, rather than blog-posts and tweets (Hollands and Tirthalli 2014, 31-33). From then on, the Stanford classes and the subsequent MITx classes came to be called MOOCs. To keep the two forms distinct, the Canadian MOOCs came to be called cMOOCs (for "connectivist", or many-to-many network) and the ones from Coursera, edX or Udacity came to be called "xMOOCs" (for "exponential" or one-to-many network).

MOOC critics for e.g. often paint this cMOOCs versus xMOOCs as a David versus Goliath scenario where the cMOOCs led by little people were partly trampled upon, partly co-opted, partly hijacked, by the sinister Silicon Valley-affiliated xMOOCs horde; cMOOCs are
sometimes talked about as the path not taken where online learning could have gone instead.

Here is Aaron Bady (2012) again:

[MOOCs have] gone from a rather singular experiment in connectivist and distributed learning to a behemoth force that we are told and retold is reshaping the face of higher education. [...] Rather than transferring course content from expert to student, the original MOOCs stemmed from a connectivist desire to decentralize and de-institutionalize education, creating fundamentally open and open-ended networks of circulation and collaboration. But the MOOCs which are being developed by Silicon Valley start-ups Udacity and Coursera, as well as by non-profit initiatives like edX, aim to do exactly the same thing that traditional courses have done—transfer course content from expert to student—only to do so massively more cheaply and on a much larger scale.

From a social science perspective, however, this does not ring true. xMOOCs were not inspired by cMOOCs, they were inspired by Internet platforms like Khan Academy and StackOverflow. Moreover, the originators of xMOOCs and the originators of cMOOCs came from widely different worlds: the former from elite computer science departments with close ties to Silicon Valley, the latter from the distance and e-learning universities in Canada. What the two movements do have in common is a disappointment with and aversion to commercial Learning Management Systems (LMSes) like Blackboard, but even here the disappointment is for vastly different reasons. Just like the open-source movement in the 1980s was born out of a desire to avoid "vendor lock-in" (Kelty 2008)—the desire to avoid being trapped into using products by commercial vendors over which customers themselves had very little control—the progenitors of cMOOCs were frustrated with the expressive power of conventional LMSes. Whereas users all over the world were using social media in innovative ways, LMSes restricted students to very particular forms of expression (the assessment, the forum posting); cMOOC creators saw themselves as empowering their learners by letting them access multiple ways of expression, the full power of the Web. The creators of xMOOCs too were frustrated with conventional LMSes, but they saw them as restricting the creative expression of instructors and teachers (rather than learners). These conventional LMSes did not "scale," they were burdened with too many administrative tasks.
and tried to do too many things and did none of them well, and moreover, they allowed instructors no good way to be data-driven and modify their teaching in response to student interactions.

What's interesting then in the cMOOCs versus xMOOCs debate is not that cMOOCs were trampled on by the xMOOC hordes, but that the cMOOC researchers were able to lump the two widely different enterprises together, to label the xMOOCs as a continuation of what they had always been doing, but through different means. That they were able to do so is testament to their messaging skills, but also to the open-endedness of the MOOC moment. The results have brought the cMOOC researchers more fame and more resources than back in 2008, when only a few higher education outlets like the Chronicle covered their efforts. George Siemens, the savviest of them, moved from Manitoba University in Canada to the University of Texas, Arlington, and is leading a 1.6 million research grant which includes Stanford and CMU.16 Siemens also was the coordinator of the Gates Foundation's MOOC Research Initiative that disbursed slightly less than 1 million dollars to researchers who were studying MOOCs; naturally, these included both cMOOC and xMOOC researchers (Straumsheim 2014). In short, the lumping together of the cMOOCs and the xMOOcs has benefited the cMOOCs researchers far more; rather than a David versus Goliath story, this is a story of enrollment and participation. xMOOC architects, for their part, have only been too

16 See https://www.uta.edu/news/releases/2014/11/LINKLab-dLRN.php for details on the grant.
glad to see their enterprise linked with a broader trend in the higher education world itself, rather than something with its roots only in Silicon Valley.

The use of the moniker MOOCs to describe the events following the Stanford experiment might be interpreted then as a symptom of the experimental nature of MOOCS and this moment in time. In the rest of this dissertation, when I use the term MOOCs, I mean xMOOCs; cMOOCs will be referred to specifically as cMOOCs.

2.3 Methods
The process of making, maintaining, and using MOOC infrastructure, that I focus on in this dissertation, has multiple dimensions, taking shape at different times and places and involving multiple actors: the engineers at edX, instructors at participating universities, and the associated network of learning researchers. I adopted a multi-sited ethnographic approach (Marcus 1995; Marcus and Fischer 1986) that encompasses participant observation fieldwork for each of these spheres of activity, while also examining the way they interact with and influence each other over time. I selected three kinds of workplaces to do my fieldwork: the offices where programmers build the software infrastructure for MOOCs, university arenas where professors and TAs construct course content that sits on top of this software, often using the tools created by programmers, and finally, the research laboratories where the data representing the students' learning is parsed and analyzed, and findings about student cognition are produced and circulated, often serving as the basis for the so-called best teaching and learning practices. I used participant observation and interviewing in offices, laboratories, conferences, and workshops to track how the infrastructure is constructed and used, paying particular attention to the question of expertise (e.g. questions like what experts are involved? What sorts of expert contests happen?)
I structured this fieldwork in two phases: 8 months from September 2013 to August 2014 in Cambridge, MA and 8 months from September 2014 to May 2015 in the San Francisco Bay Area. In Cambridge, I conducted participant observation at the edX offices, interacting with engineers and program managers (I was given a desk in the office but no access to meetings). I also observed four MIT course teams as they went about teaching a MOOC on the edX platform and spent a month embedded in the MITx office. In addition, I attended the research meetings of two research groups on campus interested in studying online learning. I attended the HarvardX research colloquium regularly every week at Harvard where I heard about the difficulties, challenges, and successes of Harvard’s course and research teams. I also conducted formal interviews with many of these actors.

From September 2014 to May 2015, I did similar research at Berkeley and Stanford. I interviewed MOOC instructors at Berkeley and Stanford, as well as engineers, instructional designers, and researchers who worked with MOOCs. At Stanford, I observed two research groups, attending weekly meetings: the Lytics Lab and Scott Klemmer’s Peer Learning group. The Lytics Lab was one of the highlights because it housed both computer scientists and more traditional educational researchers. I also attended other semi-public meetings such as the weekly technology demo at Stanford’s offices of the Vice Provost of Online Learning or VPOL. Finally, I also read continuously: research papers, mailing lists, Github comments, public discourse, etc.

Each institution was chosen for particular reasons. Harvard and MIT are the founding partners of edX and their input, based, partly on the experiences of their instructors and researchers, is important to the shaping of the software. Berkeley was an early contributory member of the edX consortium and is the only public university among the four. Stanford uses the Open edX software to host its own MOOCs and is the origin of the MOOC revolution. Finally, MIT, Stanford and Berkeley have world-class computer science
departments, while Harvard, Stanford and Berkeley have prestigious education schools. All four schools are heavily involved in conducting research with MOOCs.

Throughout this period, I also attended several conferences where MOOC researchers presented and discussed their findings about best teaching and learning practices. These included the annual Artificial Intelligence in Education (AIED) conference, the Education Data Mining (EDM) conference, the ACM conference on Learning Analytics and Knowledge (LAK), the Learning at Scale (L@S) conference and the International Conference on the Learning Sciences (ICLS). At these conferences, I attended presentations, paying particular attention to the question-and-answer sessions where scientists often express their underlying epistemic and ideological commitments (Mulkay 1976), and enact their expertise (Carr 2010). Finally, I attended one week-long summer school at Carnegie Mellon University called LearnLab where I was trained in understanding Intelligent Tutoring Systems.

As is the case with ethnographic methods of inquiry, I recorded observations (say, during a meeting or while observing an around-the-whiteboard discussion at edX) in my notebook or directly typed them into my laptop (Emerson, Fretz, and Shaw 2011). At the end of the day, I expanded these observations into extensive typed field notes, adding to them the visual evidence like photographs of whiteboards with charts and equations. I read articles on the Web, or articles circulated by my actors, and added them to my database of public discourse. I read through my fieldnotes many many times, searching for emerging themes, and writing memos when I found something interesting. As themes started to emerge, I coded some of my interviews and fieldnotes using grounded theory methods. Overall, my methods stressed interpretation and the actor’s perspective.
Chapter 3: Software-as-Platform: Introducing MOOC Infrastructure

3 SOFTWARE-AS-PLATFORM: INTRODUCING MOOC INFRASTRUCTURE

In this chapter, I describe MOOC infrastructure as it is made and maintained. By MOOC infrastructure, I mean the lumpy mixture of software, institutions and people that produce the continuous stream of courses (or "courseware"), the certified learner (who finishes those courses and is awarded a certificate), and finally, a continuous stream of knowledge claims about teaching and learning. I argue that MOOC infrastructures are an example of a software assemblage that I call the software-as-platform, an assemblage that wraps the values and practices of Silicon Valley into software, and is now being increasingly ported into multiple institutions (by particular actors), where it becomes the locus of fundamental debates about the goals and methods of those institutions. In the Introduction, I described the characteristics of the software-as-platform; here I describe MOOC infrastructures: the different actors, techniques, values, and practices that populate them; and how all of them together are an example of the software-as-platform. The next two chapters will focus on how the software-as-platform is the center of debates about the meaning of teaching and
learning (Chapter 4), and the new hegemonic form of expertise—tool-building—that emerges with it (Chapter 5).

I first describe the edX infrastructure (people, organizations, technology) that produces software, courses, knowledge, and tools. I conclude by describing how different parties within the MOOC ecosystem—learners, instructors, researchers, designers, programmers, etc.—have differing interpretations of what they are doing, a significant characteristic enabled by the software-as-platform nature of the enterprise they are involved in.

3.1 The edX Infrastructure
edX, the word, refers to two things: the edx.org website which hosts courses from its many different partner universities: 850 courses from 91 partner institutions as of January 2016 (starting from 3 partners and 7 courses in the summer of 2012), and the Open edX software that powers the edx.org website, which is open-source. Both the software and the website are primarily built and maintained at the edX headquarters in Cambridge, MA. Some features of the software may be built by programmers working within the wider open-source community (often consultants who help to create custom Open EdX installations for their customers) and sometimes by programmers working at the edX partner institutions.

The business model of the edX eco-system thus works this way: the edX organization produces software, its partner organizations, mostly universities, but also others like the Linux Foundation or the W3C consortium, produce courses that run on edX’s portal edx.org. Becoming a partner organization, aka a member of the "xConsortium," involves signing some sort of agreement with edX and paying maintenance and membership fees to edX. The agreement usually includes clauses about revenue-sharing (how the revenue from learners who pay for their ID-verified certificates will be distributed) and the kinds and amount of support edX will offer the partner organization members, especially instructors and
researchers. There are two kinds of partner organizations: "charter members" receive more support, while "contributing members" receive less support; "self-service" members may receive little to no support at all. MIT and Harvard, its founders, remain edX's most powerful partners; they have maximum representation on edX's main governing body, its board of directors, and a lot of say in determining the future direction of the edX system. The main source of revenue for the entire system (besides the initial investment and some funding from foundations) is the fees that learners pay to get ID-verified certificates.

The source code of the Open edX software is available on Github, a version-control portal used by most cutting-edge software companies. The code was formally released in June 2013. The code on Github can be downloaded and installed to create a separate instantiation of the Open edX platform. A few organizations do this; a sizable number of small consultant-type companies have come up to help institutions (universities, corporations, non-profits and others) customize the Open edX platform for their own educational ends. A new version of the code is released every week; the "release" branch of the code updated every week is the version that runs edx.org. The weekly release includes many bug fixes and smaller features that sometimes only those working at that level of software can understand. Because the features of these weekly releases may not matter much to other developers

---

17 Note that while the announcement did say that edX would be an open-source, it did not say when. The code was open-sourced only in June 2013, under pressure from Stanford. And the original code was never architected —for practical reasons having to do with time and resources—to enable open-sourcing.
outside the core edX eco-system (i.e. the edX organization and its partner institutions), and because these releases may turn out to be unstable, edX instituted a policy of having half-yearly "named" releases; these would be code releases with names (so far: Aspen, Birch, Cyprus, and the just released Dogwood) which are designed to be stable. Because the scale of edx.org (hundreds of thousands of learners, hundreds of courses) is far greater than that of other Open edX installations, and because edx.org has its own commercial needs, the version of the code that runs edx.org is sometimes different from the named releases: thus, edx.org uses noSQL databases rather than mySQL databases for some of its event logging, there is more emphasis on smooth e-commerce transactions, and so on.

Looked at as an assemblage composed of various parts, we see various things emerging from the edX ecosystem. First, of course, the plethora of courses, 878 as of February 2016; learners often sample or audit these courses; a small minority of learners takes these courses to completion and receive certificates; to do so, they have to pay a small fee. Then there are the knowledge claims about learners, learning, and teaching, often published as reports (e.g. Ho et al. 2015) or peer-reviewed papers in journals like Science (Hansen and Reich 2015) or conferences like Learning at Scale (L@S '15: Proceedings of the Second (2015) ACM Conference on Learning @ Scale 2015).18 And finally, there is the intangible notion of

---

18 The Learning at Scale conference will be discussed in more detail in Chapter 5.
"innovation": all of these together are supposed to lead to better practices, better courses, better learning and on and on, in a virtuous cycle.

Applying Bowker and Star's (2000) notion of "infrastructural inversion," I ask: what socio-technical arrangements, i.e. infrastructure, are in place (or being put in place) to make these outputs possible? What kinds of social relations exist between people, institutions, and technology? In this section, I describe this infrastructure by describing the process through which three key activities happen: making software, making courses (or learners), making knowledge. All of these three activities are often held together through the notion that these are all "tools"; tool-making functions as a discursive pivot through which all the other three activities are put into the service of "innovation," an interpretive logic that Thomas Malaby (2009) has called "technoliberalism." I discuss each of these in turn: starting from software the most esoteric but perhaps the most fundamental, moving up one step to courses, then knowledge or research, moving up to knowledge.

3.1.1 Making Software
Figure 3.1: How a MOOC looks. The course shown in the figure is called “Designing Mobile Experiences” and teaches learners how to understand user requirements and needs and then build a smart-phone application based on this.

Let’s say you decide to take a MOOC. You register on edx.org and sample a course titled "Building Mobile Experiences." The course is taught by two instructors from MIT: Frank Bentley and Ed Barrett. The goal of the course is to help you understand what it takes to build a smart-phone application (or app). The course instructors take you through the entire design process: from thinking about the audience you want to design for, how to understand
what their needs are, and then how to go about building these needs into the application itself. This is not a programming course as much as it is a course about design *methods*.

When you sign up for the class, and then log into it, you see a row of links at the very top:

**Home Course Discussion Progress Building Mobile Experiences Syllabus FAQ**

The "Home" tab takes you to a page that catalogs the course announcements; you also receive these announcements by email at the address you used to register. The Course tab takes you to the actual course itself: consisting of lectures, assessments, and other kinds of interactive modules. The Discussion tab takes you into the discussion forum where you can leave comments for the instructor as well as for your fellow learners; it is meant to resemble a community discussion forum. The Progress tab lets you monitor your own progress in the course; it tells you how many points you have scored and how many you may need in order to get the certificate. The "Building Mobile Experiences" tab takes you to an online textbook that you can peruse. Syllabus brings up the syllabus for the course, and FAQ is a list of most frequently asked questions about the course.

You click on the Course tab and it brings up the course material itself, the so-called "courseware." On the left-most column, you see how the material is divided into five weeks labeled "Week 1" to "Week 5" (see Figure 3.1). If you click on "Week 3," you see a list of further subsections. One of those subsections is called "Special Topics in Mobile" and it includes a lecture on "Urban Computing" and a multiple choice assessment question below it that is meant to test you on what you learnt from the video.
Chapter 3: Software-as-Platform: Introducing MOOC Infrastructure

Figure 3.2: The Peer Assessment Interface. Learners enter their free-form responses in the box provided. Once the time for submission has elapsed, learners are given the responses of three or more of their peers which they assess using an instructor-given rubric.

Multiple-choice questions are not the only kinds of assessments: another subunit of the class titled "Mobile location and Networking assignment" lets learners submit long-form answers and then lets them assess each other's answers (see Figure 3.2). You submit your open-response assignment as usual in the text box provided. Once the time for submission has elapsed, the peer review opens. You are first trained as an evaluator as the software asks you
to assess some pre-prepared answers using the instructor-provided rubric; these answers have already been assessed by the instructor. Once the software has determined that your assessments and the instructor's match, you are shown three answers from your fellow-learners and asked to assess them, with points as well as qualitative feedback. Finally, once the deadline for the grading and peer feedback is over, the system will return to you a grade as well as feedback from your peers (provided of course that you did your share of three assessments). This is an algorithmically computed grade that is usually the average of the grades your peers gave you. Peer assessment was pioneered in MOOCs by Coursera in early 2012 and it has since remained the mainstay of how long-form open-ended answers within MOOCs are graded (this story is explored in more detail in Chapter 5).

It should be clear by now that the core of a MOOC is really a software program. Or rather, it is a whole lot of software programs bundled into each other, nestling and jostling. The entire edx.org website with its 800+ courses is one program, of course. But each course is itself a program, as is each course component: video, assessment, simulations, and so on. These programs allow learners to interact with the course material as well as keep tabs on every move made by the learner which gets logged as data.

Where does this entire edifice sit and who produces this software? The entire software edifice sits somewhere on the servers of Amazon, hosted by Amazon Web Services (AWS).
AWS is a suite of services whereby software companies can host their software on the Web without worrying about server farms and actual hardware infrastructures; AWS takes care of this for them. Since 2005, AWS is increasingly used by many companies and start-ups around the world (including Netflix and NASA) as an infrastructure that allows them to concentrate on writing their core software without worrying about the process through which it will be hosted.19

The software itself is written—coded—in the offices of edX, a MOOC start-up company, jointly set up by MIT and Harvard in May 2012, who each put in 30 million dollars to establish it. Unlike Coursera and Udacity (which emerged from the early 2011 Stanford MOOCs), edX would be based in Cambridge, MA (rather than Silicon Valley) and more importantly, it would be a non-profit company whose software would be open-source. (Coursera and Udacity are venture-funded with proprietary software.)

I observed the edX organization in two stints: the first, when I began this project, for about three months beginning in September 2012 and then another three-month stint beginning from January to April 2014. At no point during these two stints was I actually allowed to attend team meetings but I was allowed to hang around the office, talk to employees when they were free, or during lunch. As the fieldwork continued, and the edX ecosystem became

---

19 As it was once explained to me, though not in the course of this fieldwork, this is the equivalent of letting restaurant workers concentrate on cooking and their customers without worrying about rent, inventory and other aspects of running a business.
more open, I was also able to access circulating documents about new features of the
software, best organizational practices for teaching and learning, and on many other topics
relevant to the edX eco-system.

In the summer of 2012, edX—which in its earlier incarnation as MITx had been housed at a
giant table at MIT's Stata Building where the Computer Science department is located—
moved to its new digs at 1 Broadway. This was an open L-shaped office: there were rooms
along the edges but the inside the L was an open space with tables, chairs, and computers;
and at the very inside of the square made by the L was a giant conference room. The rooms
were usually occupied by more than one person (and then even more as edX rapidly hired
more people); the open-space on the inside was mainly developers: they were grouped by
teams; those who worked together tended to sit next to each other. Along the edges of the L
were giant whiteboards, which were also prominent within the conference rooms, where
small-group meetings were often held. Whiteboards were often the site around which
disagreements were hashed out and design decisions made, which is normal practice in
technical offices and laboratories (Suchman and Trigg 1996).

In February 2014, after almost a year and a half in this office, edX moved to a new office on
Portland Street in Cambridge, to accommodate the swelling number of its employees. The
new office had two floors, and had formerly been occupied by Twitter. The lower floor was
for the rank-and-file who sat in cubicles with those who worked together sitting close to each
other; the upper floor was mostly the few executives who got glass-walled small offices; it
also housed a kitchen, a large eating space, and a very large meeting room. Large
confidential meetings often took place here; while more low-key meetings involving all
employees as well as meetup group meetings for edX developers took place in the kitchen
and eating area. Meeting rooms were all mostly on the lower-level and they consisted of a
conference table with whiteboards on the walls.
Within the edX organization, there are two main sub-units: the Engineering team and the Services team. The Engineering organization concentrates on building the Open edX software; they are also concerned with maintaining the edx.org website to make sure it works well for all of its users: learners, yes, but also partnering organizations. The Services organization within edX supports the partnering organizations in their goals: some of them offer customer and technology support, but most of them operate as "relationship managers": offering training to instructional teams and others, addressing their difficulties, and offering them advice about what courses to put up on edx.org. As the number of edX consortium members have expanded, the Services organization has also grown in size (as has edX generally); the usual mode of operation is to have a group of "relationship managers" for particular institutions or set of institutions, based on location and importance, among other factors. Relationship managers do not just maintain the existing relationship between edX and an institution, smoothening out the processes, giving advice, and so forth, they also add new dimensions to it. Relationship managers are responsible for pitching new projects to institutions: projects that might be of mutual interest to both edX and the institution (such as edX's XSeries program, or a new set of courses for AP students in US high schools, etc.). They are also responsible for making sure that edX has the right mix of activities: new courses, courses that are balanced across disciplines, the right mix between lucrative in-demand courses and more esoteric ones, and so forth. In Chapter 4, I show how the conception of Services transformed from a more education-oriented role to a more managerial and strategy-oriented role.

Most of the Engineering team works on building the Open edX software, building new features and improving existing ones; these are the ones I concentrate on. There are other roles, for instance, the dev-ops (development + operations) team works to maintain the Open edX installation on edx.org; when the installation goes down or there are problems for
learners or instructors, it is their job to track them down. Others work on dealing with urgent bugs in the production system; their job, as one developer wryly remarked to me, was about "extinguishing fires." Yet another group serves as the group that connects the open-source community of external developers with the inside of the edX organization.

The bulk of the Engineering team builds new features as well as maintains updates and improves existing ones. They are usually divided into units based on the features that they are in charge of: examples include Analytics, or forums, or particular teaching and learning features (TNL), or user experience (UX). Each small unit has four to five developers and several such units together (i.e. those dealing with features that are part of some larger conceptual unit) have a product manager. The Product managers together form a Product team which coordinates with the Services team to come up with a "roadmap" of the software: a list of features, priorities in choosing between features, and a schedule for when they will be developed and integrated into the software. The roadmap of features is open to all members of the consortium (and even to the wider public since this is open-source software), the idea being that consortium members can look at these list of features to think about their own activities.

The edX organization follows the Agile methodology of designing software, in particular the Scrum variant. Within the Agile methodology, software is designed in rapid bursts or sprints. Rather than designing fully functional features, a minimal version (MVP or minimum viable prototype) is designed and written in each sprint; it is tested to see how it works and then refined. Developer teams working on a feature meet every day in what is referred to as a stand-up meeting: standup meetings are quick affairs (hence the term standup) where developers share what they've been doing the previous day, what problems they encountered and what they plan to do today. Rather than features per se, Agile developers prefer to think of the software in terms of user "stories," certain kinds of stories that need to be broken down
into smaller stories are called "epics." After every sprint, there is a retrospective meeting where developers discussed what went right or wrong; they try to incorporate this insight into their next sprint.

The Agile methodology is not just a means for achieving certain practical ends such as designing workable software within limited budgets and time constraints. Rather, developers often see it as an ideal, an embodiment of what a good organization, a start-up, should be like. In this vision, employees are not just carrying out a process through which they design good software, but actively reflecting on the process itself, its pros, its cons, what works, what doesn't. Moreover, there is no end-point in sight: the process of reflecting on the software development process is supposed to continue indefinitely. The Agile process, while no doubt also a result of the very specific problems in software development (see Friedman and Cornford 1989), is also a conduit to the vision of the always-innovating, non-hierarchical, post-Fordist (Harvey 1991) organization.

There are three components to the Open edX software itself that its architects often emphasize: the Learning Management System (LMS), the Content Management System (CMS) or Studio, and Analytics, or the researcher's view. The LMS is the module that runs the actual course: it shows learners the videos and the assessments, keeps track of their progress, and lets instructors see how the learners are doing. It is learner-facing as well as instructor-facing. The CMS, or Studio, is mostly used by course authors (i.e. instructors and their teams) when courses are created, i.e. during the authoring phase. CMS is the internal name for this module; but its instructor-facing name is Studio. I discuss Studio in more detail in Section 3.1.2.

Software developers at the edX organization work at many different levels of the code. Some are actually involved in building units that correspond to specific user (i.e. learners, instructors, researchers) tasks. Others are involved at a very different scale: looking at
abstruse technical points like variable-passing, memory requirements, and so on, that will
make very little sense except to those who are immersed in the technical details of the
software.

New features —i.e. versions of the software—are released every week. They can be anything
from new features to fixing a trivial bug (which may of course not be so trivial to fix). Here
for e.g. is something from the latest Open edX release on June 20 2016, as of writing this
chapter:

In course discussions, learners can no longer vote for or report their own posts.
(TNL-4703)

Here, learners can no longer vote or report their own posts; TNL-4703 refers to the name of
the ticket in the JIRA management system used for tracking bugs and features. This sounds
like a trivial fix and one wonders why it took edX four years since its inception to fix this.

As an ethnographer, I am not able to answer a question like this without having to do deep
down into the innards of the Open edX software. It is possible, for instance, that checking to
make sure that the user who "votes" for a post is not the same user who authored the post,
would take a non-trivial amount of time because of the nature of the database. Thus, it was
only possible for edX to make this change once the database structure itself became suitably
different. Changing the innards of a database is quite likely to have taken four years.

On the other hand, the weekly release may include a full feature. Thus, here is an item in the
weekly release from June 6, 2016:

Options for sending bulk email messages have changed. When course teams send
bulk email messages from the LMS, they can now select one or more of these
recipient groups: Myself, Staff and Administrators, and All Learners. When you
send an email message to more than one of these groups, duplicate recipients are
filtered out, so that recipients who are in more than one group do not receive
multiple copies of the same email message. (TNL-4356)

This is a substantial feature that allows instructors and course teams to send out emails,
flexibly, to a subset of the people involved in a course.
In sum, the development of the Open edX software is an unending process of implementing big features, maintaining, modifying and debugging small ones, all the while paying attention to what users (learners, teachers, course developers, researchers) are doing and what they may need in the future.

3.1.2 Making Courses
The task of making courses is done by course teams—content creators, instructors, instructional designers, beta-testers, and others—who typically work in participating institutions, members of the xConsortium. The key person on this team is usually not the official instructor for the class (e.g. the person who appears in the videos) but a post-doctoral student or content creator who works with the official instructor to author the course. This person, whom I'll call the course author, is assisted by others who help with specific parts of the course: the video, the assessments, the forums, and so on. The course authors and their team may be supported by the participating university itself; this support and the form it takes varies from university to university, depending on the resources available. MIT, for instance, offers funding to instructors, who then use it to recruit course authors and their team. Most of the pedagogy is left to the instructional team itself but MIT offers some technical, video and managerial support. Harvard offers not just funding but also human resources; instructors get a course author who works within a centralized workplace built especially for course authors, video editors and researchers. Berkeley instructors get some funding but hardly any organizational support; they seem to rely on the edX organization itself for support. (Stanford’s model is similar to MIT’s but since Stanford is not a member of the edX consortium—it only uses the Open edX software to host its own MOOCs—I will not be discussing Stanford in this chapter.)

Participating institutions typically are interested in more than just creating courses. They are interested in understanding the kinds of learners that take MOOCs and their motivations; they
are also responsible for fostering and encouraging research using MOOC data. One of their tasks that they take a great amount of interest in was the creation of pre-course and post-course surveys to understand MOOC learners; the Qualtrix web software has become key to this. Surveys are an important part of understanding MOOC learners, given that this is a diffuse population distributed across the world. Most participating universities (especially those like MIT and Harvard which have a large number of courses) have standardized their surveys across courses; and universities have been talking to each other to make surveys more in sync with each other's. Surveys are not just ways of gaining information about learners; they have also emerged as a way to make interventions—especially social psychological interventions—to nudge learners into better learning practices (e.g. see Kizilcec and Halawa 2015).

Figure 3.3: The Studio application for authoring courses. Studio separates the content of the course from its presentation. Course authors can therefore add content (text, assessments, videos) through a GUI. The software will make sure that the final output looks and runs like a course.
Courses are authored using an application called Studio. Studio resembles the authoring interfaces of blogging platforms like Wordpress, Blogger and Drupal. By that I mean that the style of working is similar: courses and course components are "authored," just like a blog-post is, and then "published" by pressing a button. (Courses, of course, are far more complicated than blog-posts and involve not just content, but also grading policies, publishing dates, homework deadlines, and more.) Like blogging content management systems, Studio allows authors to concentrate on the content of a course (lectures, assessments, problems, simulations, grading policies, course deadlines) and not worry about its presentation. edX maintains two versions of Studio: one at studio.edx.org and the other at studio.edge.edx.org. The edge server is a sort of preparatory stage; course teams create and tinker with their course on edge before moving it over to the actual edX platform. This is necessary because having actual course development happen on the main server means risking a course being accidentally published and learners being able to view it.

Typically the course-making process begins 6 months to 2 years before a course goes "live."

The first step is usually the commissioning of the course itself. Different institutions have different processes through which they decide what courses are to be made into MOOCs, some processes being more transparent than others. Some institutions issue open calls for instructors to apply to teach MOOCs on edX; others use a more top-down process where administrators at institutions find courses that might work best for the institution as a MOOC. Once a decision is made, the instructor is allocated resources for course-making; these resources are typically different for different institutions. At MIT, instructors are offered funds to be able to employ their course teams, and are allowed access to resources (an educational technologist to consult with when the team runs into problems, a video consultant to help them out with managing the video files, and so on, all of them at MIT's Office of Digital Learning). At Harvard, instructors not only get funding, they also get a dedicated
course developer who works within the centralized space with other course developers working on other HarvardX courses. Instructional teams at Harvard or MIT rarely, if ever, come into contact with anyone from edX. Instructors at Berkeley get some funding but not many other resources: they typically work on their own and connect with their edX program manager from time to time to report on their progress or ask questions if they have difficulties.

The next step is to start authoring the course itself. Typically instructors draw on their existing course materials: notes, assessments, syllabi, problems. They start authoring them using the Studio application. The process involves all kinds of decisions: what material to keep for the MOOC (level of difficulty, duration of the course), what sorts of assessments to design the course around (long-form essay-type questions, multiple choice questions, projects, forum postings), whether or not to use videos and deciding the kind of videos to make (low or high production values, duration, etc.). Some instructors choose to go beyond the standard course component types offered by the Open edX platform; in that case, they need to come up with new types of course components and then write them into their courses. To do this, they require some programming skills; sometimes course team members already possess this skill, at other times, they may need to hire someone to do this. Using a non-standard course component entails certain risks; course teams cannot look to the edX
organization to support them on this; when there are any errors or problems, they need to fix them themselves.\textsuperscript{20}

An added wrinkle in the course authoring process is the making of course videos. Instructional teams take many different approaches to video-making: some use pre-recorded lecture videos taken in the classroom, others shoot specially for MOOCs but the production values can vary depending on whether it’s shot with literally a webcam or in a studio with professional filmmakers in attendance. Editing and post-processing typically also depends on how elaborate the video is: many course teams choose to edit it themselves; others hire professional editors to enhance the videos and make them more cinematic. Course teams also have to make sure that the videos are transcribed and that the transcript is integrated into the video. Typically this is done using a professional transcriber service. Videos on edx.org are typically hosted on YouTube and then embedded within the course.

Course authoring typically takes up much of the course team’s time; once the course is online and live, course teams usually have one main task: to monitor the discussion forum and to make sure that nothing goes wrong. Typically when a course component does not work, learners will complain about it on the discussion forum; course teams will then either fix the

\textsuperscript{20} The standard way of making non-standard course components within the edX world is through an architecture that is called xblocks, which is native to the Open edX software, but which needs a considerable amount of familiarity with it. However course teams may build their course components in other ways and then embed them into an edX course using what is called the LTI specification. LTI stands for Learning Tools Interoperability.
problem, or contact the edX organization to fix it (if it's a software issue). Most issues on the discussion forum are about assessments: an assessment did not give the learner credit even though the learner did it right, multiple options in a multiple-choice assessment are right but credit is only being awarded for choosing a particular one, the assessment is malfunctioning in the learner's browser, the learner ran out of time in the exam because the software malfunctioned, and so on. Different course teams take different approaches to the forum: certain teams will have dedicated community monitors (usually undergraduates paid by the hour) who will address learner questions or escalate them to the next level; in other teams, the course developers take very specific interest in what the learners are saying and participate extensively in the course forum.

Once the course run is finished, course teams get to decide whether to take the course off edx.org or to keep the material online (though no one will be monitoring it). While edX encourages teams to leave the course online even after it’s over, course teams make various decisions on this, for reasons ranging from their comfort to the question of copyright. Often course teams will opt for multiple runs of the course, usually once or twice a year.\footnote{It’s worth nothing here that both edX and Coursera seem to be increasingly pivoting towards what they call “self-paced” courses. Self-paced courses are always on; students finish them at their own pace, and the deadlines adapt accordingly.}
3.1.3 Making Knowledge, Doing Research
The edX organization was announced with three goals: it would increase access to high-
quality courses by underprivileged learners around the world, it would help universities
improve on-campus education with its cutting-edge technology, and it would spur research on
education and learning, not only through the data that its MOOCs would generate but also by
building technology that made research easier. My interlocutors told me that these three
objectives had been crafted so that the project would appeal to as many faculty as possible,
especially for Harvard initially when the establishment of edX was being debated there, but
also faculty of other potential affiliated universities.\textsuperscript{22} For some other faculty, especially at
MIT, it was the research that was the appeal.

Software, data, and the software to parse data to make decisions are the three ways through
which knowledge is generated within MOOCs. This knowledge generation need not
explicitly form a part of research proper. Rather, the features of the Open edX software are
such that instructors can choose to be researchers, if they want to; i.e. they may choose to be
"data-driven" and draw on the data to modify their teaching. Researchers may choose to do

\textsuperscript{22} The fourth unsaid objective, some joked, was revenue, although no one knew yet how the revenue would
actually materialize. The story goes that at a faculty dinner to promote edX, Michael Sandel, a superstar
professor of philosophy, stood up to declare that of the three goals, only access was interesting; the contribution
to on-campus teaching didn't matter to Harvard because the students were good enough, the research was a joke,
and the revenue was non-existent. One of the earliest Harvard courses on edX was Sandel's justly famous course
on Justice. This was also the course that led to the famous open letter from the philosophy faculty at San Jose
State University (see \textit{The Chronicle of Higher Education} 2013).
research that is more applied: enough to have actionable consequences for teaching and learning but not enough to be published in journals or conferences. Through its four years, the Open edX software has successively added features that allow data access, at multiple levels, by instructors and researchers. These include the analytics dashboard as well as the analytics suit, the A/B testing (or conditionals) feature, the possibilities for data download by instructors, and the data pipeline so that large volumes of click stream data can be analyzed.

Data has been one of the foundational justifications for MOOCs. In the words of edX’s inimitable president Anant Agarwal, MOOCs would turn out to be the “particle accelerators” of learning research. In an interview given to the Chronicle in 2013, he said: “All of the schools in our consortium have access to all the data in the platform in an anonymized format. This is what I call ‘the particle accelerator of learning’—big data in learning in real-time” (Stokes 2013). This, of course, is rhetoric, and statements like this were routine when any interview with a MOOC founder was published. But it does bring to mind one of the foundational texts of the anthropology of science: Sharon Traweek’s study of particle physicists “Beamtimes and Lifetimes” (Traweek 1992). As Traweek shows in her book, getting access to the particle accelerator, or what particle physicists call “beamtime,” is a crucial part of a particle physicist’s lifeworld and a considerable source of anxiety in the early stages of a physicist’s career. Similarly, while in principle, data is anywhere and everywhere, in practice, access to data—and therefore to the possibilities of data—is mediated by software and institutions.

Ever since edX began, instructors, at Harvard and MIT, have had two requests: that they be able to access data. Not the big data, the entire clickstream of the course, that edX the organization concentrated on getting to them (which also took a year or two), but the small data, for e.g. being able to see how learners answered a particular assessment which would
allow them to determine whether the assessment worked (i.e. answer questions like: was it wrong? too difficult? confusing? etc.)

The edX organization slowly rolled out a number of features for instructors: including downloadable data on particular assessments as well as "dashboards" that allowed instructors to understand student enrollment and "engagement" (edX calls this analytics portion of the software as Insights). These metrics and dashboards allow instructors to understand the demographics of their learners (their age, geographical locations, their prior education, and gender) as well as the extent of their involvement with the course (which is called engagement; it includes information such as the frequency with which they log into the course, the number of videos they watched, or the assessments they attempted). It is unclear how much instructors actually use these metrics but their purpose is clear, at least from the edX side. edX wants instructors to keep their learners more engaged as well as modify their teaching on-the-fly, if possible, to take into account the demographics and inequalities between learners. While it is instructors who are supposed to modify their behavior in response to these metrics and visualizations, the metrics themselves are decided by the edX organization.
Figure 3.4: A bar graph from the Analytics dashboard. This shows instructors how learners responded to a particular assessment.

What instructors have typically wanted from edX is more granular data about how learners performed on the assessments which they could download and examine on their own. Starting in the middle of 2014, these started to become available for instructors through the Insights dashboard. In these reports, which are only available for particular kinds of standard assessments, instructors can download CSV files where every row represents a learner attempt at a particular assessment. The row tells the instructor the ID of the assessment, the learner's response, and the credit awarded, among other information. Typically, I found instructors more interested in these downloadable CSV files than in the Engagement dashboards.

In the middle of 2015, edX unrolled another new feature for instructors (and also by implication researchers): an A/B testing tool that allows instructors to conduct experiments. Instructors can set up their course so that a certain portion of their learners sees a particular piece of course material, while another group sees an alternative part of the course material. Theoretically, this allows course teams to find out what kinds of instructional material produces better outcomes for students. Instructor-researchers can see the some of the results
of the A/B tests in their small data downloads, but usually have to wait until they get access to the clickstream data, which I turn to next.

The assessment downloads file for instructors is not the data Aggarwal was talking about when he mentions "particle accelerators." That excitement is reserved for the clickstream data—the big data—the copious recording of every click made by the learner. This data however is not so easy to obtain either for instructors or researchers.

Within the edX system, there are two salient layers through which this data travels. First, edX, the organization, has an “Analytics” team. Every week, this team takes all the student data they’ve collected on an Amazon server for all the courses for a particular university and sends it off to one relevant person at the client institution, the so-called Data Czar. The Data Czar is the custodian of the data at the university. Researchers who wish to use the data for research contact the Data Czar. After making sure they have the relevant permissions (institutional review, adequate credentials etc.), the Data Czar sends the data off to the researchers.

The data is depicted in Figure 3.5. As you can see, it is not human-readable and is often jagged and irregular. It is very difficult—even for those who are well-versed in programming and data analysis—to read and make sense of this data. Depending on how many resources the Data Czars have, the data they give researchers is rarely in this raw form. To do this
however, data czars need to be either skilled at parsing data themselves or have teams of
coders who can do this.

3.2 Conclusion: The Open-Endedness of the edX Infrastructure

MOOCs are constituted by two main kinds of actors: the learners and the instructional staff
(for short-hand, I will refer to these as instructors); in addition, there are two other actors to
take into account: researchers and external software developers (particularly in the case of the
edX ecosystem). All of these actors are connected together via software (built by software
designers working at edX or Coursera). Let us take each of these actors in turn.

MOOCs are "open" so any learner can take a course irrespective of their background as long
as they have an Internet connection. The open-ness here means that learners may be from a
wide range of backgrounds and demographics: they could be of any age, ethnicities, national
origin, educational credentials, and skills; they also may have a multiplicity of goals: from
professional certification to self-edification to a desire to meet new people. In other words, a
learner is an individual, a tabula rasa free of (or at least, dealt with as if he or she is free of)
any institutional background. Sometimes, the assumption is relaxed, or it is made more
specific. In the early years of edX, the learner was assumed to be able; in response to a
Department of Justice investigation, edX has since made the Open edX software more
accessible to disabled users (Thomason 2015). In practice, it was found that MOOC learners
tended to be disproportionately older with graduate degrees or even PhDs; other researchers
who build tools for the edX software have tried to come up with ways to increase
participation by more disadvantaged learners. But at bottom, the edX software, in principle,
is supposed to cater to all learners who interact with it on their own time, in their own space,
with whatever resources they have at their disposal, and for whatever goals they choose. To
help them do this, the software is designed to be as easy as possible to navigate, extensive
help documentation is provided to them, and they are given a course forum where they can interact with other learners on their own time.

Self-sufficient instructors are a different type of creature. Unlike learners, instructors are supposed to deliver a fully operational course for which they need certain kinds of help. The Open edX software assumes that instructors will get this assistance from their institutions (which is why, remember, edX partners with institutions not with instructors). For these institutions to provide the instructors this help, i.e. to render the institutions themselves self-sufficient with respect to course-making, the edX organization offers them advice on best practices for making videos, authoring assessments, managing learners in discussion forums, and so on. This is done by the part of the edX organization called the Services team, often outside the Open edX software; it is rendered through phone meetings, shared project management tools, and extensive documentation that are provided to the institution and to the instructors.

But the most important software tool—a key part of the Open edX software—through which instructors are made self-sufficient actors is Studio. Studio is a graphical user interface or GUI that separates the content of the course from its presentation; in a way that is very similar to how web publishing software (e.g. Wordpress or Drupal, those that constitute Web 2.0) separate the text from its presentation. Other than videos, which need to be filmed, composed, and edited outside the Open edX software, before being uploaded into it, all the other course components—text and assessments, primarily—can be authored inside Studio, for those with little to no technical skills. In other words, instructors do not need to know any programming or coding to author courses in Studio (although they can author a greater variety of course components if they do). Since any edX course, or even any course component, is a software program, the Studio interface allows course teams to author a program without knowing how to code. It makes them technically self-sufficient.
But the process of instructor self-sufficiency has another, arguably more important dimension. Instructors—and their supporting institutions—are left free to determine the goals and structure of their courses. In their survey of why institutions invest in MOOCs, Hollands and Tirthali (2014) find six major clusters of objectives:

- Extending Reach and Access
- Building and Maintaining Brand
- Improving Economics
- Improving Educational Outcomes
- Innovation
- Research on Teaching and Learning

Yet, even these clusters hide a range of strikingly different meanings. The instructors for MIT's Introduction to Physics courses want to teach a very rigorous level of introductory physics to their learners. An instructor of a course on Virology however sees it as a course that expands public health awareness, rather than rigorous knowledge of viruses. An instructor teaching a course on Microsoft Excel or Linux or Data Science has a completely different goal: to train or retrain existing workers for newer kinds of jobs and job-skills. Moreover different institutions have very different processes for deciding which goals to favor. Some institutions issue a formal call for MOOCs and use a selection process to decide which MOOCs to fund. Others opt for a more informal process with administrators choosing hand-picked faculty whose courses align with their institutional goals.

Researchers are another constituency whom software designers configure as self-sufficient users. For political reasons, over questions of who owns the data generated by the courses, publishable research about the edX ecosystem is carried out at the edX partner institutions, and therefore it is partner institutions—and the disciplines that dominate these institutions—who get to decide the goals and purposes of this research. Every week, a team at edX takes the clickstream data produced by courses and sends it to the particular institutions that are then free to grant this data to researchers. In 2013, edX started producing an up-to-date
research and data documentation for these research teams. While analyzing this clickstream data is one way of producing knowledge claims about teaching and learning, a more favored method amongst researchers is the experiment, which yields causal knowledge claims. Initially, in the early days of MOOCs, researchers resorted to hacky workarounds to conduct experiments through MOOCs. In 2015, a feature for A/B testing (which took a long development period) was developed and built into the Studio content authoring tool so that instructors (and researchers) could easily create content experiments in their courses. The idea is that instructors could offer different course content to different learners and thereby measure which course material produces best learning outcomes. Researchers (and instructor-researchers) are therefore domesticated into self-sufficient users of the software, who work with their own institutions, to decide the aims, goals, and methods of their research.

The last important constituency is external developers i.e. those who develop software tools but are not housed within the edX organization. These developers may work for institutions who are customizing an instance of the Open edX software to use for their own purposes, but are not members of the consortium; they may be researchers who are working to build new functionalities into the Open edX software; they may be instructors who wish to develop new kinds of course content. Developers—understood in these varying senses—have been made self-sufficient by the edX organization by crafting a software architecture; there is a certain point of extension of the Open edX software that is made available for these external developers (with attendant tools for development); at the same time, the "core" of the software is only modifiable by the members of the edX organization. For the Open edX software, this unit of extensibility is called an "xblock"; an xblock is the core component of a course; a course is made of xblocks, which in turn can be made of other xblocks.
What we see here then is a set of "tools" being made available to different constituencies—learners, instructors, researchers, and external developers—but letting them use those tools for their own goals and purposes and with their own methods. In other words, the socio-technical infrastructure of edX is designed to enable its own contingency with respect to its ultimate form and purpose by configuring the various actors who participate in it as self-sufficient with respect to their goals, their objectives, and their methods to achieve those goals. Thomas Malaby (2009) has labeled this notion of designing for contingency using technology with the moniker of "technoliberalism," a form of governance that he argues is a successor to bureaucracies which are designed to minimize contingency. The Linden designers that he studied seek to empower their Second Life users by giving them "tools"; these tools help these users craft not only Second Life but also themselves, they are tools of empowerment and self-actualization as much as they are tools to build things.

In the world of edX (and MOOCs in general), those designing the core software (the software developers, the management, at edX) seek to empower instructors and learners (and also researchers, other developers and so on) by building various tools (like Studio) into Open edX; they refer to this process as becoming a "platform." As Beth Porter, then edX's Vice President of Product explained to me, the difference between merely building software versus building a platform was that "you expect to support multiple kinds of ... ecosystems, you have
authors, you have learners, you have instructors, you have administrators, you have course
developers, you have developers, like, it's an ecosystem is a platform"; "what the platform
has [is] all these parties that can participate in it to create a full-fledged system of people
interoperating with one another."23

The edX software-as-platform not only has to cater to multiple kinds of self-sufficient parties,
it also has to cater to multiple kinds of content, courses of a variety of hues, shapes, and sizes.
As an educational researcher put it, "Like a platform is very, like general, [pause], thing. And
how do you design something that is general enough to like capture metrics that can assess
the same kind of competency when the content and the goals of the courses that are using it
are quite different? And it requires like very strong agreement about what kind of
competencies are universally important."24 What she leaves unsaid is that there are no
universal agreements over what competencies are important.

How, then, can the software-as-platform cater to multiple constituencies as well as multiple
kinds of content? It does so, this dissertation argues, because software designers deploy the
principle of heteromation. By designing software as tools, they let human beings, i.e. users,
fill in the gaps in the software with their own meanings, their own goals, and their own
methods. Ekbia and Nardi (2014) label this particular configuration of humans and programs

23 Interview with Beth Porter, 7/20/2015.
24 Interview with Emily Schneider, 5/13/2015.
working together as "heteromation" wherein humans are “fashioned as computational components” with their outputs connected to programs. In this arrangement, humans take up the slack when programs fail (i.e. when there are no universal rules to decide what is important) and use local contexts to make particular working choices. On the other hand, programs are fashioned as tools that allow humans to do what they could not on their own. The edX software-as-platform thus constitutes and connects multiple, diverse actors, configuring them as self-sufficient; allowing something like universality to be achieved by letting humans take local decisions that then feed into the software itself. In doing so, it constitutes the actors who work through it (such as instructors, administrators, learners, external developers) as "users" while those that build the core software become designers and architects.

A complicated set of designer-user relationships exists with respect to the edX platform. The key designers sit at the edX organization in Cambridge; these include members of the Engineering team, the Product team, and the Services team. The Services team works with clients: institutions in the edX consortium offering them advice on best practices, easing their way through the software, and collecting from them their experience with the software and what features they would wish for. These are carefully collated and passed on to the Product team, which collects this and assembles a "roadmap" of product features that will be developed over the next quarter, year and so on. The roadmap lists both which features have highest priority and a rough timeline for when those features will be built. The Engineering team works on the roadmap, unless some unforeseen contingency arises. In other words, the Services team shields the Engineering team from direct contact with clients.

In this arrangement, the partner institution—and by implication, the instructors, researchers, and developers who work for it—is configured as a client of the edX organization. Some client institutions have a voice as a member of edX’s Technical Board and they can advocate
for features that they want there. In other cases, the edX Product team takes into consideration both clients’ needs, the priorities of the edX organization, as well as other contingencies, to build a roadmap of features. Institutions have other options as well. They can simply take the Open edX software, modify it with whatever features they want, and then have their own instance of it to run their particular course. This requires a high degree of technical competence (or paying someone who has this technical competence); but it also restricts their impact: their course is not on edx.org which is a magnet for attracting learners simply by virtue of other courses that exist there; also, a feature that is implemented in their own version of Open edX will not propagate "upstream"; it has less chance of spreading to other courses by virtue of being integrated into the main branch of the software that runs edX.org.

It is in this sense that instructors, researchers, and developers at client institutions are users of the Open edX software; that their agency has been purposefully, productively, constrained through particular channels. These users have recourse to two channels: they can "hack" the software if they have technical skills, although their hacks may remain isolated to their own local instances (their courses or their instance of the Open edX software). Their other option is to pursue intuitional channels: to exert institutional pressure so that their features get added to the "core" of the software. As we will see, institutions often have to use some combination of the two channels; develop code that implements what they want and then negotiate with the edX organization to integrate it into the core of their software. In other words, then the boundary between the "core" edX organization and its users maps to the boundary between the "core" software that is controlled by edX and the activities of its clients (institutions and people). This boundary can be crossed in socio-technical ways: by hacking code and then negotiating so that this code moves from the periphery to the center.
What about learners? If the core developers configure instructors and researchers as clients who are also empowered in terms of setting their aims, goals, and methods, then the relationship of all parties to the learners is more complex. It is often instructors and researchers who channel the "voice" of the learner: the instructors because they interact with them through their courses and MOOC learners are often very vocal about their wants and needs in the discussion forums; the researchers because they survey and question these learners and mine their interaction data; many of the influential research findings from MOOCs have come through surveys supplemented by clickstream data. Yet, sometimes, the core developers also feel that they may be ceding too much ground to instructors. As an edX engineer told me, "[It's possible that] professors have so loud a voice that it's really easy for teaching and learning to really be about teaching and maybe we're not hearing enough from the students [about learning]."25

Speaking for the learner, then, as the end-user, is then something of a prized thing in the world of MOOCs. It is in this way that the concept of pedagogy enters the picture. If the learners are learners, then those who are best positioned to understand them are experts on learning. But as we have seen, the idea of a "course" on the edX platform is itself flexible, course topics range from disciplines to skills. The question of what pedagogy is, and who is

---

25 Interview with Ned Batchelder, 7/2/2015.
best positioned to understand and shape it, therefore becomes a point of contestation between the different constituents of the software-as-platform. I turn to this in the next chapter.
4 PLATFORMIZING PEDAGOGY: MOOC INFRASTRUCTURES AND THE TRANSFORMATION OF ROLES

"We do software so that you can do education. [...] We’ve built the bulk of the platform (including the boring parts!), and they’ve built the interesting pedagogical tools on top."


On 16 September 2014, the website opensource.com carried an interview with edX software developer Ned Batchelder (Huger 2014) in which Batchelder explains edX’s mission as follows: "we do software so that you can do education." He continues: "We’ve built the bulk of the platform (including the boring parts!), and they’ve built the interesting pedagogical tools on top." These quotes juxtapose multiple oppositions in particular ways (see Table 4.1): the "we" versus "you"/"they", the "software" versus "education", the "platform" versus "pedagogy," and finally the "boring" versus "interesting." By "we," Batchelder means the members of his organization edX; the terms "you" and "they" refer to "academic teams" at partnering institutions but it could just as well be a reference to anyone: you the educator, you
Chapter 5: The Tool-Builders: Computer Scientists, MOOC Infrastructures, and the Emergence of a New Expertise

the teacher, you the educational researcher, you the educational hobbyist, and even you the learner. The particular trope employed in the quote, of being a mere "platform," whose eventual form is shaped only by the activities of its creative users, and is therefore a mere tool for them, is a standard trope of Silicon Valley and an ideology closely associated with the software-as-platform, as discussed before. This chapter takes a much closer look at the process through which the particular trope—of platform, tool, open-endedness—became naturalized within the world of edX; it also explores the consequences of this, in particular how the establishment of the edX infrastructure in the software-as-platform format led to changes in the institutional roles within the edX ecosystem, a process that I label as "platformization."

<table>
<thead>
<tr>
<th>We (edX)</th>
<th>You (instructor, learner, educational hobbyist, partner)</th>
</tr>
</thead>
<tbody>
<tr>
<td>software</td>
<td>education</td>
</tr>
<tr>
<td>platform</td>
<td>pedagogy</td>
</tr>
<tr>
<td>boring</td>
<td>interesting</td>
</tr>
</tbody>
</table>

Table 4.1: The list of oppositions between "platform" and "pedagogy." Platforms are mostly boring software infrastructures made by edX, that allow edX partners to build interesting pedagogical tools on top.

4.1 edX Becomes a Platform

In Chapter 3, I described the infrastructure—the software, institutions, and people—that constituted the edX eco-system, responsible for the courses on edx.org as well as the knowledge claims about teaching and learning. edX began with a funding of 30 million dollars (each) from MIT and Harvard. Its envisioned business model was similar to Coursera’s: edX would partner with educational institutions to create courses which would be offered for free, or for a small price, to learners around the world. The edX organization consists of three main units: the Engineering team, the Product team, and the Services team. However this was a comparatively recent achievement: the Product team was only instituted
in the summer of 2014, and the crafting of the Services team had only begun a few months earlier. In this chapter, I tell the story of how this happened. Briefly, I argue, that edX began with an organizational conception of being an educational start-up; as it got more established and its management started to think about the long-term, it shifted to being more of a software organization: one which produced a “platform” that other institutions—its clients, or others who used its software—could use in their quest to teach, learn, or to improve pedagogy. I label this particular stance towards pedagogy as the “pedagogy agnosticism” of the edX eco-system, enacted through its software-as-platform, and explore its consequences. Along the way, I compare the edX eco-system to Carnegie Mellon University’s Open Learning Initiative (OLI), which allows me to explore “pedagogy agnosticism” comparatively.

4.1.1 The View from edX: The Early Years (2012-13)
Before edX was edX, it was MITx. In response to the runaway success of the Stanford MOOCs, MIT announced its own initiative, MITx, a platform through which MIT would open up certain MIT classes to the world. The first class from MITx, Circuits and Electronics, ran for 14 weeks from March to June 2012 and was taught by Anant Agarwal, who resigned his position at MIT to head the initiative. Eventually, more than 90,000
learners signed up for the course and around 7000 finished it to earn a certificate. MITx ran out of a simple office at MIT’s Stata Center.\textsuperscript{26}

MITx eventually became edX. The edX organization was formally announced in a press conference on May 2, 2012, by the presidents of Harvard and MIT through an investment of 30 million dollars each. Unlike Coursera and Udacity, edX would be a non-profit organization with non-proprietary open-source software.\textsuperscript{27} The envisioned edX business model would be similar to Coursera’s: edX would partner with educational institutions (not just limited to MIT or Harvard) to create courses which would be offered for free, or for a small price, to learners around the world. The move from MITx to edX was clearly inspired also by what was happening with Coursera; by this time, Coursera had partnered with 12 other universities and proposed to have 100 courses by fall 2012. An organization to compete with Coursera could not be run out of MIT; it needed its own autonomy. Also, MIT administrators saw themselves as serving as serving a population—MIT residential students—that MITx did not necessarily serve.

From the beginning, key administrators at MIT and Harvard sought to distinguish what made edX different from MITx, its predecessor (which would continue to exist, but instead be the

\textsuperscript{26} See https://6002x.mitx.mit.edu/wiki/view/StaffListing for a list of the original MITx employees.

\textsuperscript{27} The actual source code was not made public until June 2013.
organization that helped produce MIT’s MOOCs). One MIT administrator analogized it to the difference between books and a printing press. edX, he suggested, was like a printing press, while individual institutions it was affiliated with, mainly MIT and Harvard, would be responsible for the books that were published; MITx therefore would concentrate on producing courses for edX, and supporting digital learning initiatives within residential MIT education. In this formulation, edX was a value-neutral, content-agnostic enterprise, whose job was merely to publish what its authors came up with. On the other hand, the "printing press" analogy was also supposed to evoke images of the printing press’ revolutionary impact underlying the transformation of early modern Europe (see Eisenstein 2005).

It is worth unpacking the "printing press" analogy. It is true that publishers do not actually write books; nevertheless, they are responsible for a number of editorial and curatorial functions. Publishing houses select—through their editors—what counts as a book worth publishing, and they help authors through rounds of editing. Sometimes, they even pay authors advances, and share revenues with them once the book hits the market. How much of the curatorial and editorial responsibility of a publisher would the edX organization take with

---

28 See Section 2.3 for the story of MITx. But briefly, MITx, relabeled ODL (Office of Digital Learning), became the organization that would help produce MIT MOOCs.

29 Then Chancellor Eric Grimson in a meeting with the representatives of MIT’s Graduate Student Council’s subcommittee on digital learning, 9/28/2012.
Chapter 5: The Tool-Builders: Computer Scientists, MOOC Infrastructures, and the Emergence of a New Expertise

respect to its authors i.e. its affiliated universities? How much input would it have into the courses it hosted and the actual course content and its design?

The answer, at the beginning at least, seemed to be: a lot, primarily through the envisioned class of employees called "edX Fellows". When edX moved into its new quarters in the summer of 2012 (from a giant conference room at MIT's Stata Center), it had less than 30 employees, 3 institutional partners (Berkeley, Harvard, and MIT), and about 7 courses running. While roles tended to be more fluid in the early days, and everyone helped during a crisis, there were two main divisions: the Engineering team worked on the underlying software that powered courses, while "edX Fellows" worked with partnering instructors to design courses, and were the key link between pedagogy and software.

How was the edX Fellow job conceptualized in edX's early days? Here is the job description from September 2012:

*EdX Fellows focus on the development of innovative solutions for online teaching, learning, and research. They create and manage partnerships with faculty and staff from edX universities in the development, implementation, and evaluation of online courses and related learning products. EdX is seeking candidates with doctoral degrees in the social sciences, humanities, natural sciences, engineering, or education, who are committed to the development of innovative pedagogies to improve online teaching and learning. [my emphasis]*

Three characteristics of the edX Fellow emerge from this description: they have to "manage partnerships" with instructors at participating universities, they have a disciplinary PhD, and they have an interest in "innovative pedagogies." The edX management envisioned a whole host of Fellows who spanned the entire disciplinary spectrum; they would build on their disciplinary expertise, their interest in pedagogy, and their knowledge of software and technology, to work with instructors in participating institutions on one side, and the Engineering team on the other. Early edX Fellows were often PhDs from MIT or Harvard (usually in physics, chemistry, mathematics or engineering) with a deep interest in teaching and learning, who joined because they wanted to be part of this idealistic new enterprise.
During my stint at edX from September to December 2012, I was given to understand that this—the presence of the edX Fellows—was also what made edX different from Coursera, which by that time had scores of partners and courses in its portfolio. Coursera mass-produced courses generic run-of-the-mill courses because they were concentrated on quantity; edX, on the other hand, was about quality because the courses were customized and adapted both in terms of pedagogy and in terms of content.

One example of this—the crafted rather than mass-produced nature of edX courses—that was often brought up was the popular periodic table that was designed and put into MIT professor Michael Cima's class "3.091x: Introduction to Solid State Chemistry," a famous MIT introductory class taken by both Chemistry and Material Science majors. Victoria, the edX Fellow who worked on this class, had a PhD in Chemistry from MIT; she coordinated the making of this tool working with the professor and the course team (which had requested it) on the one hand, and the edX engineers on the other. Victoria finished her PhD in Chemistry from MIT in 2007. She then moved to New York where she worked (sometimes volunteered) for various education initiatives; she was particularly interested in those that took science topics to a wider audience without dumbing it down. Around that time, she took the first MITx class on Circuits and Electronics (i.e. 6.002x) which she said was a class that anyone with a general interest in Circuits should take. She had really liked the class and had
emailed one of the early MITx employees (whom she knew from her days at MIT) to congratulate them. So when she heard that the Solid-State Chemistry class (which was not quite Chemistry, she added, but more material sciences, though well within her area of expertise) would be coming to edX, she had called edX up to ask if she could help in any way. That led to the job, which as of that conversation, was a contract job, but later on became permanent. 30

Victoria described her job to me in 2012 as an interface: to tie together the work of the content creators (the TAs, the professors), the programmers, and the learners. With the instructional team, it was to make sure that the course material, the tests, the homework were all correct and on the site; with the programmers, it was to make sure that the online tools for the class were built correctly and to the right specifications. She had also worked on framing questions for learners, based on the lecture videos, as well as scrubbed the lecture videos when they became too MIT-specific (the lecture videos were recorded from MIT classroom lectures and therefore tended to include things like class deadlines, other MIT classes, or names of MIT buildings that were not necessarily relevant or useful to the online learners).

30 Fieldnotes from conversation with Victoria on 11/12/2012. This conversation took place at the Andala Café in Cambridge, MA where edX had organized a meetup (through the website meetup.com) for its learners. Victoria was coordinating it. Sadly no learners showed up, though this meant that Victoria and I got a lot of time to talk.
Chapter 5: The Tool-Builders: Computer Scientists, MOOC Infrastructures, and the Emergence of a New Expertise

She said some of her duties—especially the job of coordination—could be construed as managerial though she hesitated to use that word.

I asked her that how she would conceive her work as an input-output matter, if she wanted to describe it in terms of the inputs she received that she wanted to transform into outputs; it was my lame, clumsy attempt at doing infrastructural inversion. She said the inputs were clear enough: in a sense, the inputs also involved people, the professors and the TAs and so forth, as well as the chemistry. The output right now, she said, was the course itself but she would actually like her output to be the learning produced in the learners. She said that right now she had no knowledge of how much people are learning. Sure, there were indicators, e.g. the persistence level of the learners can tell you which ones might be getting something from the class; but better measures were needed. In the future, Victoria told me, she would like to do learning-related experiments, as edX matured as an organization. So, for instance, is it more useful to show the video before the questions, or show the questions before the video? Rather than being evaluated on the course itself (her output, one might say), she would like being evaluated based on the learning that she was able to produce in her students/learners, rather than the metrics of persistence and engagement that they were currently using.31

31 Fieldnotes from conversation with Victoria on 11/12/2012.
It should be clear from this that Victoria saw herself and her role within the edX eco-system as a steward of learning; her expertise was education-related rather than software-related. Robin, another edX Fellow, described his job as a “learning engineer.” Like Victoria, Robin had a PhD in Physics from MIT. When he finished his PhD, he had realized that he had no interest in doing particle physics research; rather, he had become far more interested in working in physics education research. He had started applying for postdoctoral and other opportunities in this field, but then he heard about edX. At edX, he said, he was more of a “learning engineer” than a ‘learning scientist.” His meaning was that he would not just be reading, or publishing journal papers on, pedagogy, learning, and cognition. Rather, he would be working to integrate actual research into courses that were being taken by learners all over the world. When this conversation took place, Robin was about to embark on working on edX’s new physics course, something he said he was really looking forward to, since physics was his main interest.\(^{32}\) It should be clear that both Robin and Victoria saw their jobs as on the borderland of corporate life, science, engineering, and education; they wished in the future to do applied research, measure what teaching interventions resulted in better learning outcomes for learners, build those interventions into future iterations of courses, and thereby promote a virtuous cycle of improvement and iteration.

\(^{32}\) Fieldnotes, 10/30/2012.
Though the early edX Fellows clearly saw their job as reinventing existing course material through inputs from the educational literature, it is doubtful they did any “learning engineering” proper in those early days. Their job in those early days (2012-13) tended to be a lot less glamorous. They assisted overworked professors and underpaid TAs who, in turn, were making MOOCs out of personal interest and enthusiasm, and working with technology that was still in its inception. There was constant scrambling to meet course deadlines, and a sense of mad improvising to surmount new problems. It was enough sometimes that the course material made it online without gaping mistakes; "learning engineering" and pedagogical innovation took a backseat, perhaps to be engaged in during the second run of the course. Every day at lunch, war stories were exchanged about overworked faculty members. One example: an edX Fellow wrote to her faculty collaborator with a list of deadlines; he replied back, curtly, telling her that there was only one deadline in this enterprise: the day the course went live. At times of high-stress deadlines, or when there was a technical glitch, everyone chipped in. As Rebecca Petersen, one of the early edX Fellows whose job was to coordinate edX’s community college efforts, told me:

\[A]\text{At the end of the day, if there were huge deadlines or where everyone was [inaudible] working on something, everyone just kind of chipped in, it didn't matter what you were, what people were doing, how many, all of us helped, in captioning video in those early days ...}^{33}

---

\(^{33}\) Interview with Rebecca Petersen, 7/22/2015.
As edX gained more partners in the fall of 2012, the edX management realized that the Fellows model of heavy collaboration was simply not extendable to more instructors, not without increasing the number of edX Fellows drastically. They started to see idea of Fellows specializing in disciplines as mistaken. As an early edX employee told me,

>[A]n idea was hatched that we will support these courses just like we did Anant’s course [the original MITx Circuits and Electronics]; we’re going to replicate the Anant course model. Okay, we can’t scale that way as an organization but we’ve hired a bunch of PhDs with content specialties in these areas. [emphasis mine] 34

"Scale" emerged as a particular way for the edX organization to think about its operations. The concept of "scale" is an engineering concept that is now widely used in the business world, especially in Silicon Valley. When engineers use the word "scale," especially as a verb, they mean: to build something is both large and small at the same time (Kelty 2000). The term is used for concepts ranging from software to organizations to business models. When MOOCs began, most talk about scale referred to the fact that these courses were taken by thousands of students. When the edX organization started to think about scale, they were thinking about courses, partner institutions, business models, work processes, and revenue, in addition to learners. The vision was that edX would have many partner institutions, with edx.org hosting hundreds of courses from these partners.

34 Interview with Rebecca Petersen, 7/22/2015.
The question of revenue had also come to the fore, after the first flush of the early months. edX had been created with a giant endowment of 60 million dollars by its two patrons in addition to some funding from foundations. But, despite its non-profit status, it was also expected to be a sustainable institution. Somewhere in the latter half of 2012, the edX management made a particular set of decisions around revenue, concentrating on two sources: learners and institutions. Learners were encouraged to take a course for a verified-ID certificate (rather than just the free version) which cost a small sum of money (usually $50 or so). The second decision was to scale up, recruit many more universities and other institutions as partners. The idea was that, in the future, edX would have hundreds of courses on its site, varying widely in types (engineering, humanities, social science, programming, data science etc.). A big course portfolio would bring in the learners (especially those who paid for the verified certificates) whose revenue would then make edX sustainable. Partner institutions paid edX a fee for being able to put up their courses on edx.org, and in return, got a cut from the revenue accrued by their courses. In the next subsection, I describe this process of scaling.

4.1.2 Scaling the Platform (2013-15)
Ribes (2014) has argued that ethnographers studying infrastructures should study actors who are tasked with "scaling" as well as the "scalar devices" they use for the purpose (e.g. the all-hands meeting, the social science survey and computational devices). The edX management accomplished its own scaling project through two tightly coupled approaches: (1) configuring the instructor and the partner institution as self-sufficient users responsible for pedagogy, and (2) transforming the edX Fellow into a "program manager," thereby shifting the role from its focus on pedagogy to a focus on management and revenue. I describe both of these in more detail below.
Chapter 5: The Tool-Builders: Computer Scientists, MOOC Infrastructures, and the Emergence of a New Expertise

<table>
<thead>
<tr>
<th></th>
<th>Number of partner institutions</th>
<th>Number of courses</th>
</tr>
</thead>
<tbody>
<tr>
<td>August 2012</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>December 2012</td>
<td>6</td>
<td>15</td>
</tr>
<tr>
<td>May 2013</td>
<td>12</td>
<td>37</td>
</tr>
<tr>
<td>October 2013</td>
<td>21</td>
<td>86</td>
</tr>
<tr>
<td>June 2014</td>
<td>38</td>
<td>194</td>
</tr>
<tr>
<td>December 2014</td>
<td>64</td>
<td>383</td>
</tr>
<tr>
<td>May 2015</td>
<td>73</td>
<td>519</td>
</tr>
<tr>
<td>December 2015</td>
<td>90</td>
<td>650</td>
</tr>
<tr>
<td>January 2016</td>
<td>91</td>
<td>650+</td>
</tr>
</tbody>
</table>

Table 4.2: The steep growth in the number of edX partners and courses. Note that in this time, the number of edX employees has only increased from around 30 to about 150.

At edX’s beginning, the course author was imagined as an actor who collaborated with edX Fellows (who also specialized in the course discipline) to create courses. Now, course authors were imagined as self-sufficient actors—perhaps supported by their institutions—who created courses with as little handholding as possible from the edX organization. In other words, the course author was transformed from a collaborator to a user of the edX software.

The process of software design is saturated with three kinds of users: users as clients (who specify the features of the software), users as actor-constructors (who are empowered through the software), and users as co-designers who reflect on the software (Hales 1994; Mackay et al. 2000; see also Woolgar 1991). In the design of the Open edX software, the learners are the key users who have to be empowered; they take courses on edx.org for a variety of reasons: self-enrichment, career advancement, and curiosity, among others. The courses are tools they use in their particular projects of self-fashioning. From collaborators, instructors too were turned into users who are both clients and actor-constructors. They were empowered to be self-sufficient, as actors who would take the building blocks of the Open
edX software and construct (at least in the ideal case) innovative courses. As clients and paying customers, they could also demand and expect certain features from the edX software as well as help in using these features. As actor-constructors, they were encouraged to reflect on the process of course-making and pedagogy in general, thereby creating courses that resulted in better learning.

It should be clear what this shift meant for instructors: they gained an amount of autonomy while also being subjected to a set of constraints. In their earlier incarnation as collaborators, they requested special features which were built for them by the Engineering team. This entailed a dependency but also an amount of power to demand particular features. As self-sufficient users, they no longer had access to the Engineering teams at edX. But they could use the building blocks of the edX software and construct courses, building on their own skill and ingenuity. They—or their home institutions—could decide the goals and the objectives of their courses for themselves, without any interference from edX. The features they requested were not however routed straight to the Engineering team. Rather these requests were collated by a Product organization and incorporated into the roadmap of the edX software, to be built in due course, a process that might take months.

The key device that allowed instructors to become self-sufficient course authors was a graphical user interface (GUI) called Studio that was built into the edX software in late 2012. Studio resembles the authoring interfaces of blogging platforms like Wordpress, Blogger, and Drupal. Courses and course components are "authored," just like a blog-post, and then "published" by pressing a button. Courses are far more complicated than blog-posts; they involve not just content (meaning videos, text, assessments, practice problems), but also grading policies, publishing dates, homework deadlines, and more. Like blogging content management systems, Studio allows authors to concentrate on the content (lectures, assessments, problems, simulations, grading policies, course deadlines) and not worry about
Chapter 5: The Tool-Builders: Computer Scientists, MOOC Infrastructures, and the Emergence of a New Expertise

its presentation. In prioritizing Studio, the architects at edX drew on the separation between content and presentation, a key innovation in the original Web 2.0, and a key part of social media platforms (Helmond 2015). Studio was the key priority of the edX organization through most of the fall of 2012. Even when operational, it was clunky at best and improving Studio remained a top priority well into 2015.

Yet, Studio alone would not have worked if it wasn’t supported by the construction of an elaborate bureaucracy. The edX Fellows were turned into a Services organization. Rather than seeing themselves as providing pedagogical support to partner institutions, Services became structured around providing managerial, technical, and best practices-type support to instructional teams (and also research teams, though that came later) at partner edX institutions. At the same time, through the Services organization, edX encouraged partner institutions to provide support to instructional teams themselves, either by creating new organizations or through their already existing pedagogy-support organizations (like the Teaching and Learning Center or the School of Continuing Education). On-site training sessions, checklists, good documentation, notes on best practices, were the tools through which Services would help partner institutions build personnel and organizations to support their instructors.

Roles that were supposed to be about "learning engineering" thus became re-conceptualized as managerial; the edX Fellows now assumed titles like "program manager" (PM), "relationship manager," "channel manager," etc; the organizational structure of edX that I described in Chapter 3 was the result. A job advertisement for the "Program Manager" position in April 2013 on the edx.org website asks for candidates who are "able to examine and explain what they do in great detail and able to think abstractly about people, time, and processes" as well as "train partners and drive best practices adoption" and "improve workflows." No mention is made of having a PhD in the disciplines, or expertise in...
pedagogy, although "2 years of experience working with University faculty and administrators" is asked for.

A job advertisement for "Director of Education Services" (who supervises the "program managers" and reports to the Vice President of Services) best expresses the role that the Services organization plays:

The Director must be experienced in capacity planning and operations, understand how to deploy lean collaboration and able to build alliances inside edX and the University. In conjunction with the Program Managers, the Director of Education Services will supervise the collection of research, the retrospectives with Professors and the assembly of best practices in course production and operations. The three key deliverables are the use of a well-defined lean process for onboarding Professors, the development of tracking tools, and assessment of effectiveness of Best Practices. [my emphasis]

A table from an "Onboarding Document" created by edX for its partner institutions is shown in Figure 4.1. The document tells partner institutions that "[a]fter completing these pages, you should have a better understanding of the people and processes involved as well as the resources required to support a MOOC effort." Primarily, the document instructs institutions to think about the different kinds of roles required to support a faculty member running a MOOC. These roles range from particular specialties needed within a "course team," e.g. a "course coordinator," "content developers," "forum moderators," and others. More importantly, it suggests "support staff" that these instructors and their course teams will need to draw on. Some of the suggested support staff roles have explicitly pedagogical specialties e.g. an "instructional designer," while others have duties that range from media and programming support, legal advice, and workflow and financial management. Institutions are encouraged to use their own existing organizational resources or create new ones.
Figure 4.1: A table from a UniversityX “Onboarding” document from edX to the participating university. The table outlines “assets,” “tools,” and “resources” for the UniversityX personnel to use in order to produce MOOCs.

When a new institution joins the edX platform, the particular PMs assigned to it start off by conducting training for the new members. Instructors and/or supporting staff come to the edX office in Cambridge for a two-day training session. They are usually taught a list of things that are deemed most important to know: how to use Studio, how course teams should be formed and how roles should be assigned, what the important milestones and deadlines in authoring a course are, how to plan the course content (especially videos, which required specialized services and expensive equipment), and so on. After they are back at their partner institutions, the PMs keep in touch with them through regular phone meetings and project
management tools, helping them out with advice when needed. This is considered essential both to making sure that the courses are of a certain quality, but also that courses get produced in time and the course production pipeline is well-oiled.

Did it work? The project of scaling was certainly accomplished. The number of edX's partner institutions and courses grew rapidly from 2013, as Table 4.2 shows. The variety of courses also expanded: edX's partner institutions now range from universities, corporate clients, advocacy organizations and non-profits; its courses too include everything from the sciences to learning Microsoft Excel. edX does not release revenue figures so it is hard to know if scaling increased revenue as it was expected to.

Yet, the process of scaling—inspired primarily by revenue concerns—has also changed the edX organization. As the course production pipeline has stabilized, Studio has matured enough to be used by technically novice instructors, and partnering institutions have taken over the job of providing them support.

The edX PM's job has increasingly shifted to questions of sales, revenue, and strategy. It is now part of the PM's role to pitch new projects to partnering institutions in an effort to recruit them into these projects. PMs are also tasked with thinking about how to market courses to increase enrollment (e.g. think about search engine optimization keywords) and to advise institutions on what the best possible mix of their courses on edx.org should be. In an interview conducted in July 2015, an edX PM described her job to me using the analogy of "portfolio management." Just as market investors aim to diversify their portfolios to manage risk, she saw her job as using the great amounts of data that edX had collected (for e.g. on how students picked a course to enroll, or how students reached a course through Google searches) to advise partners on the right balance to strike between niche courses and courses that attracted learners. This conception of the job has, in turn, changed how PMs are
recruited. Starting from disciplinary PhDs and those with an Education background, they are now increasingly more PMs who come to edX from careers in business consulting.

Some edX Fellows successfully transitioned into the managerial roles. Robin, for instance, was one of those who made the transition to a more managerial role quite successfully. On LinkedIn, one of his managers describes his transition as follows:

Robin was one of the original few at edX when I started [...]. A brilliant physicist and a gracious kind and thoughtful human being. I witnessed an incredible career changing moment when he was tasked with shifting to a process and relationship-oriented program manager role, in an agile development environment. Unlike academe, edX was a start-up on steroids, and demanded much of these program managers and all team members to deliver. [...] He quickly became the 'go to' person the whole team depended on.

But many of the initial edX Fellows have since left the edX organization. Some left early when the job transitioned into a managerial role, a few leaving for partnering institutions where they could explore their interest in pedagogy. Even Robin himself left edX in 2015 after three years. He joined another education tech startup in Boston where he could go back to his first passion: physics pedagogy. Ned Batchelder, one of edX's earliest employees explained this transition this way:

I think the people that were attracted to edX—because they were education geeks rather than technology geeks—I think those people have tended now to move out to the organizations around edX. There's a number of instances of people who were at edX when I joined who very clearly were more interested in the education aspects than in the technology aspects and those people now are at BU [Boston
The course production infrastructure within the edX platform has shifted increasingly towards a particular division of labor: edX produces the software (i.e. the technology) while the partnering institutions produce courses (i.e. the pedagogy). Only a very small minority of those who work at edX are, as an edX engineer put it to me, "education geeks." The edX Fellow role, conceived initially as a pedagogical role, has with time become a managerial one. And edX, the organization, perhaps thought of by some as an educational one, has also become more avowedly about technology. I call this phenomenon—a structuring tension, rather—as the "pedagogy agnosticism" of the edX platform. The next section teases out the implications of pedagogy agnosticism.

4.1.3 Pedagogy Agnosticism: Reasons and Implications
The story of the edX platform is a tale of two visions. In the first vision, which was instantiated mostly through the role of the edX Fellows, edX was an educational institution that contributed to digital pedagogy by collaborating with instructors from partnering institutions. In the second vision, edX took on the role of a technological company whose role with respect to its partners was managerial, coordinative, and technical—i.e. a

---

35 Interview with Ned Batchelder, 7/2/2015.
36 Interview with Ned Batchelder, 7/2/2015.
"platform"—rather than pedagogy-oriented. Could the first vision have worked? One edX old-timer argued to me that it could have. edX could have remained a "boutique" outfit that produced only a few, yet highly innovative courses, by partnering only with select universities and partners, through heavy collaborations with instructors. This could have still been a sustainable system, he argued; the prestige and social capital of the organizational actors involved, especially MIT and Harvard, would have helped solve the problem of sustainability somehow.

That path that was taken led to a separation between the software and pedagogy functions, with the software being centrally managed by the edX organization, and the pedagogy becoming the province of its self-sufficient partner institutions and instructors. I call this particular division of labor as the "pedagogy agnosticism" of the edX platform. I define "pedagogy agnosticism" as the rhetorical separation between the software and pedagogy functions within the edX world—through the trope of the platform—that have resulted in a subtle expansion of the category of "pedagogy."

The definition consists of two parts. First, the separation between the software and the pedagogy is a rhetorical distinction rather than a technical one. There is already a certain model of a "course" within edX, and this is arguably a pedagogical feature of the software.

---

37 Interview, 1/22/2016.
edX courses are made of units (e.g. week 1, week 2), which are made up of sections, and subsections. Each subsection consists of a course component. Course components can be videos, text, or assessments, and it is these course components that instructors can author, which the software then puts together as a course. The structure of the course itself has stayed roughly the same ever since edX began in 2012. Many instructors I encountered seemed perfectly happy with this arrangement, or at the very least, accepted it stoically. Others were more vocal; some complained that this structure of a course corresponded roughly to how technical and engineering courses were organized, and were all wrong for the humanities and social sciences (and of course, it’s worth nothing that this structure was designed for the MIT Circuits and Electronics class). Of course, they went ahead and did it anyway; the allure of a new world of learners from all over the world was hard to resist.

But if instructors felt themselves constrained by the structure of the course, they (and the organizations that employ them) also experienced a new sense of freedom in terms of setting the goals and objectives of their courses. This is the second aspect of pedagogy agnosticism—the expansion of the category of "pedagogy" made possible by the design of the edX infrastructure. Consider the variety of classes on the edX platform: they have ranged from highly rigorous MIT-level introductory classes on college physics and electronics to general education classes on Chinese history and global architecture to interventionist classes on public health, poverty economics, and women’s empowerment to data science classes aimed at professional education to enable access to lucrative jobs in Silicon Valley to classes teaching basic skills like Excel spreadsheets and Mongodb database software. edX’s partner institutions are mostly universities (although these too are now geographically diverse, from India, Japan, China, France, Netherlands, Jordan, and others) but also include software companies, non-profits, advocacy organizations, and others. In addition, a growing number of software consultants provide services for other organizations who wish to use the Open
edX software. The primary clients of these software consultants have been corporations who wish to use the software for corporate training and even as a conduit to reach consumers.

The sense of possibility experienced by the instructors is compounded by the sheer range of learner activity. Among all the actors involved with MOOCs, learners remain truly diverse and truly mysterious, in their motives, goals, and actions. Learners come to MOOCs with a wide range of motives: some just to dip in, others for self-education, some to learn useful skills, and still others to get certificates that allow for new jobs and self-advancement. In the beginning, MOOCs were criticized for their high "dropout" rate because only 5 to 10% of learners actually earned certificates; yet, researchers have shown that learners do a lot even if they don't earn certificates: some watch the videos but don't take quizzes, others only access certain units, still others just want to sample (see Ho et al. 2015; Kizilcec, Piech, and Schneider 2013). Like many other forms of participation on the Internet, learner activity in MOOCs is diverse and hard to describe. The future of the edX, as the edX management sees it, hinges on keeping itself open to new forms of learner and instructor participation, and new models of instruction.

To look at what a "biased" pedagogy platform may have looked like, we can look at Carnegie Mellon University's Open Learning Initiative (OLI). The OLI looked very remarkably like the early version of edX to the point where term "learning engineer" was often used by the creators of OLI. OLI was started in 2001 with funding from the Hewlett Foundation in response to MIT's release of Open Courseware, a remarkably successful MIT initiative to release course materials for most of its courses to the world. Rather than going the courseware route, CMU chose to build actual stand-alone courses (i.e. courses that worked without instructors); OLI was not a pedagogy-neutral platform; it was built extensively on CMU's cognitive science and intelligent tutoring systems expertise, with an emphasis on deliberate practice and scaffolding. OLI courses were built through long hours of
collaboration between "a Carnegie Mellon professor serving as the 'faculty content expert,' a software engineer, a designer with expertise in human-computer interaction, and learning scientists," not unlike the early edX vision of collaboration between faculty, edX Fellows and edX engineers. The OLI objective was to completely transform course content through the application of cognitive science and design principles (see Walsh 2010, chapter 4, for a history of OLI). The labor was all mashed together even if the expertise of each role was clearly demarcated.

Where is OLI today? OLI achieved a great deal of acclaim for its statistics class, which was shown to have equal outcomes as an in-person statistics course (see Bowen et al. 2016). CMU used the success of its statistics class to create relatively successful introductory classes in the sciences. Because OLI members disagreed on whether their pedagogical model could be extended to humanities and social science courses, OLI remained confined to introductory science and technology classes because its creators believed that the CMU cognitive science model could only be applied to scenarios where subject-matter experts could articulate their own expertise. There were also notable tensions between the OLI personnel and the CMU administrators. The aim of reinventing a course through collaboration between learning scientists and faculty meant that the course production process was both expensive and time-consuming. When foundation funding ran out, CMU spun off the underlying OLI software (not its content, which is open) into a new for-profit start-up called Acrobatiq. The OLI personnel moved to Stanford in 2012 and a new Stanford-based OLI is being created. In perhaps what can only be called a delicious irony, OLI @ Stanford chose the Open edX software for its new incarnation; the key reason, apart from the fact that it was open-source (which its CMU version was not), and being used in-house for Stanford MOOCs, was that Open edX had a fully operational content management system i.e. Studio which the earlier
Chapter 5: The Tool-Builders: Computer Scientists, MOOC Infrastructures, and the Emergence of a New Expertise

OLI platform lacked. To summarize, pedagogy agnosticism (an openness to different kinds of pedagogy) allowed the edX eco-system to grow, in a way that OLI was not able to.

How have different constituencies reacted to pedagogy agnosticism as a structuring principle? Within the edX management, pedagogy agnosticism and the sharp increase in the number and variety of courses is looked on with approval because it indicates that edX is now more than a software company, it is now a platform which learners, instructors and researchers can use for self-actualization and to enact their own vision of teaching and learning. This definition of "platform" draws on the management and strategy literature that carries a strong cache in Silicon Valley. In this literature, the "platform" has been interpreted as a site of innovation that is helped along by an inter-locked "ecosystem" of organizational players. For example, Gawer and Cusumano (2002) take Intel and Microsoft as primary examples of how to execute "platform leadership" which is a term that refers to how these companies managed to create an organizational eco-system where they produced products (the Intel processor and Microsoft Windows respective) that were also basis of products made by other organizations, creating a stream of both innovation and revenue.

The edX engineers too argue that, on the whole, pedagogy agnosticism has lead to better software design. To the Engineering team, the early days seemed chaotic, a time when "every new course was a crisis of some sort" because it either required new features to be built into the software, and/or, it increased the load on the servers. For Engineering, this was not exactly a sound state of affairs. The notion of building special features based on the needs of specific classes did not correspond to good software engineering practice—it was inefficient. Good practice meant paying more attention to the longer-term, having a road-map for the future, and playing the long game.

But pedagogy agnosticism has its critics as well. Influential experts from the e-learning community and many existing educational researchers and learning scientists have argued
that by embedding a particular course structure into the software (one that relies on videos and is based on how engineering is taught) and then letting the instructors build on it as self-sufficient actors, edX courses are condemned to be instructionist i.e. based on a top-down models of instruction where learners are seen as passive recipients of instructor knowledge. In contrast, they have pointed to their own blog-based or tutor-based open courses as more pedagogically appropriate and learner-centered (e.g. see Bates 2014; Siemens 2012). This, of course, often strikes the edX employees and course-makers as unfair since they see themselves as learner-centered and the emphasis on videos in their classes as merely temporary while new tools and technologies are developed. Meanwhile, other accounts (e.g. Selingo 2014) show that learners enjoy listening to lectures as podcasts to and from their way to work. The purposes and objectives that instructors and learners have overlaid on MOOCs have shifted the meanings of pedagogy; moreover it is instructors and learners rather than educational experts from e-learning and the learning sciences that get to decide what constitutes pedagogy. Pedagogy agnosticism, a structuring infrastructural principle, meant to create revenue and scale, may have unintended consequences, shifting the meanings of both "course" (see DeBoer et al. 2014) and "instructor."

The sharpest criticism of pedagogy agnosticism comes from the instructors and researchers at MIT and Harvard, edX’s original patrons and first clients, who find their voices lost within the sea of other clients and institutions. As edX’s original instructors ("we were edX before there was edX," as an MIT instructor put it in a meeting), they had easy access to the edX engineers who often built their proposed features for them. Now denied access to the edX engineering team, and being forced to ask for features (or even build them on their own) through a bureaucracy explicitly constructed to insulate the edX engineering team, they react with barely concealed frustration. A Harvard researcher told me that edX, the organization, had a profound aversion to instructional design. By that, he meant that edX did not care for
the principles of good pedagogy because they consistently refused to act on the requests of Harvard and MIT course developers and researchers, people who had a sounder understanding of pedagogy than edX employees. When I asked one MIT course developer if she saw herself in the future as working for edX or Coursera, she replied firmly in the negative. edX, she said, was just a platform; it was course developers, in their pedagogical and technological capacities as "learning engineers" who did the useful things and that was what she intended to keep doing. In effect, the platform that constrained her actions was labeled as doing nothing, curiously affirming its pedagogy agnosticism. Other course developers voice more moderate criticisms. Some wish that edX released a timeline of its in-development features a few months in advance, so that they could tailor their course content accordingly.

It is unlikely that edX will abandon pedagogy agnosticism as a structuring principle of its infrastructure anytime soon. It is also likely that the software will have more and better features in the years to come (if edX and MOOCs survive, that is). And it is very likely that newer course developers will enter the edX world without knowing any other state of affairs: they use the features available in the software in the best possible way to create courses of their choice and have very little access to the edX Engineering team (except as beta-testers for new features under development). The combination of scale and pedagogy agnosticism has reconfigured the role of an instructor as well as the notion of a course, at least in the world of MOOCs.

This chapter has shown the uncertainty and surprise that platform designers themselves experience as they construct their platform eco-systems. edX chose to reinvent itself as a technical and managerial organization to achieve sustainability and scale. It's transformation into a platform was accompanied by a realization that its interests were best served by creating a cadre of program managers who had consulting experience rather than pedagogical
interests. Yet, this conception meant that edX’s initial employees—many of whom joined the organization out of an interest in education—slowly dropped by the wayside; some joined partner institutions that were now supposed to do education; still others drifted away into joining other educational companies. This has led to better software engineering processes overall but it has also exacerbated worries about whether the direction that the software develops is right.
Chapter 5: The Tool-Builders: Computer Scientists, MOOC Infrastructures, and the Emergence of a New Expertise

The world of Silicon Valley is permeated with the ideology of "tools": tools as augmenting human beings, tools as objects that human beings can use for crafting themselves, and tools themselves as an engine of innovation, if their production, circulation, and consumption are carefully governed. This dissertation argues that this ideology of tools has led to the emergence of a new form of expertise: that of the tool-building. Tool-builders do not see themselves as experts on a topic or domain; rather they see themselves as combining insights from various experts—the so-called "domain experts"—that they then build into the tool. Naturally, it is tool-builders who decide what kinds of domain knowledges to encode into their tools. Typically, there is a tension at work within these tools: they are supposed to be open-ended because users can adopt them for their own purposes of self-actualization but
they are also means of governance in that these tools are supposed to direct their users into paths that are deemed socially beneficial (e.g. the quest for innovation, the pursuit of good teaching and learning practices, producing good journalism etc.). The operating presumption within the software-as-platform is that building tools, or using them innovatively, is at the leading edge of creative work.\textsuperscript{38} Since these tools are usually instantiated as software, tool-builders require some amount of coding skills to make them. Yet, tool-builders are also constrained by interfaces: those with more access to the “core” of the software-as-platform (core code, core data) are more likely to make tools that disseminate widely than those who don’t; this asymmetry which is a consequence of institutional relations rather than coding skills goes often unnoticed.

In this chapter, I focus on the emergence of tool-building as a new form of expertise. First, I situate my contributions within the social studies of expertise; I show that tool-building is both a new form of expertise as well as a \textit{theory} of expertise. Next I turn to the key tool-builders I examine in this chapter: computer scientists interested in the study of learning. I describe how computer scientists apply their (historically specific) technical ideas to particular domains (here: learning and education) and how, through their technical practices

\textsuperscript{38} Tom Boellstorff (2008) refers to this as “creationist capitalism.” Malaby (2009) reports that making things has higher value in the world of Second Life, at least from the perspective of the Lindens who make the Second Life software-as-platform: Lindens were invariably disappointed when most of Second Life’s users preferred to simply consume rather than create.
and social imaginaries, they negotiate with other actors in that domain—a process that
accounts for their rising expert authority and has significant political implications. I do this
both by ethnographic story-telling of two key computer science interventions in the MOOC
world: learnersourcing and peer assessment. I then compare computer scientists to their
closest cousins in the education technology world: learning scientists, through which I bring
out the salience of this new kind of expertise. I conclude by offering some thoughts on what
the emergence of this expertise might mean.

5.1 The Social Studies of Expertise
The question of expertise has always been a thorny problem for STS, as well as associated
fields like history, sociology, and anthropology. The key questions have always proved
difficult to answer: what makes an expert an expert? What accounts for the rising power of
certain sorts of experts (medical professionals, social scientists) in modern societies: is it
because these experts have some sort of special access to truth or is it because of their control
of specifically modern institutions like the hospital, the laboratory, the factory, or the
university? What accounts for the differences in the relative power between experts, doctors
and nurses, psychiatrists and social workers, all of whom work with similar sorts of questions
and domains? How do experts negotiate their division of labor within institutions, in public
life, and in the realm of the nation-state? The scholarly literature divides into three strands:
the embodiment of expertise: how experts are trained and their habitus (Dreyfus and Dreyfus
1988; Grasseni 2009; Jones 2011; on habitus, see Bourdieu 1977); boundary work (Gieryn
1995) or how different experts negotiate the division of labor with each other or with non-
experts (Abbott 1988; Eyal 2013; Suryanarayanan and Kleinman 2013); and finally the
political dimensions of expertise: the role that expertise plays, and should play, in democratic
governance (H. M. Collins and Evans 2002; H. Collins and Evans 2008; Wynne 2003;
Jasanoff 2003; Hirschman and Berman 2014). In her survey of the literature on expertise,
Carr (2010) argues that there are four dimensions to the social studies of expertise: understanding the process of expert socialization i.e. their training and habitus, how these experts are evaluated and authenticated, experts’ relationship with the institutions that authorize their expertise, and the naturalization of expertise, how it comes to be seen as natural and residing in the bodies of these experts. Boyer (2005) argues provocatively that the social studies of expertise should go beyond studying experts in their workplaces; that, by limiting their studies to workplaces and institutions, social scientists end up studying experts the way they want to be studied, and thereby reproduce their subjects’ i.e. the experts’ own (hegemonic) viewpoint on expertise.39

Of these different strands in the literature, the question of “boundary work” (Gieryn 1995) and the division of labor between these experts (Abbott 1988) is most relevant to this chapter. In his seminal *The System of Professions*, Andrew Abbott (1988) argued that the professions are best considered as a system, and the starting point of studying the power of professions is to understand a profession’s “jurisdiction” i.e. the “link between a profession and its work” (20). The jurisdiction is often a point of contest or negotiation between different professions. These jurisdictional negotiations can have multiple varying outcomes: one profession may gain control over a body of knowledge, while another gets to apply that body of knowledge to

---

39 This is a provocative insight but sadly, remains unrealized in this dissertation you are reading, mostly for practical reasons.
clients or problems (psychologists and social workers); one profession may become subservient to the other in the workplace (doctors and nurses); or the professions may divide up the clients between them (e.g. lawyers and accountants). Jurisdictional contests are most subtle (and their outcomes highly contingent) when they happen in the workplace; less so when they happen within public media and public opinion; and most clear when they are legislated into law. Abbott’s second major contribution is his notion of abstraction: professions differ from other occupations in that they claim sovereignty over an abstract body of knowledge. It is the extension of this abstraction to other problems or clients that signals a jurisdictional contest. Abbott also argues that getting this abstract body of knowledge just right, not too abstract, not too concrete, is the key to controlling the jurisdiction. Thomas Gieryn has applied this notion of jurisdictional negotiations to the question of demarcation criteria for science—the so-called battle between philosophers of science like Thomas Kuhn, Karl Popper and Paul Feyerabend over what is the definition of science (see Lakatos and Musgrave 1970 for a spirited debate)—that he labels “boundary work.” Boundary work is the work of defining what constitutes science, of deciding on the boundary between science and pseudo-science (or non-science), or that between science and politics, or between science and religion, among others. This work, for Gieryn, is done not in laboratories but in public debates, and often not by scientists, but by others: "journalists, bureaucrats and lawyers" (394). Boundary work is thus about the consumption and circulation of science.

Recently, Gil Eyal (2013) has made a call for reorganizing the sociology of professions around the sociology of expertise; he proposes that the study of experts be replaced with the study of expertise, which is separate from experts: "if we want to account for the superior and speedy execution of a task it is not enough to focus on the actors and their skills" (p871). Drawing on Actor-Network Theory (Callon 1987; Latour 1987), he argues that expertise should be studied by understanding the "networks that link together objects, actors,
techniques, devices, and institutional and spatial arrangements," (864) how these networks are made, who gets to make them, and how these networks make possible routine expert actions and utterances. The formation of these networks may not necessarily be the result of jurisdictional contests between experts; rather, they may be the result of jurisdictional contests that are initiated not by the experts but by involved and interested non-experts. Eyal illustrates this with his explanation of the autism epidemic: the systematic rise in the number of diagnosed autistic children in the United States. Eyal argues that the autism epidemic depends for its existence on “deinstitutionalization”: the move within the United States from housing mentally retarded children in institutions to having them, now defined as mentally ill rather than retarded, taken care of by their parents with support from the state. Thus:

In this series of reiterated problem solving, deinstitutionalization was an extremely significant episode because it provided the ecology within which an alternative network of expertise could be assembled. The key actors in assembling this network were not child psychiatrists, but the parents of children with autism. Seeking to modify the relations within Kanner’s network of clinical expertise, which expropriated their knowledge and stigmatized them as “emotional refrigerators,” they set up alternative mechanisms of data collection and experimentation, which bypassed the clinician. To do so, they allied themselves with behavioral psychologists and occupational therapists who, due to their peripheral position, developed therapies that blurred the distinction between mental illness and retardation as well as the distinction between lay and expert. It was this new actor-network, composed of arrangements that blurred the boundaries between parents, researchers, therapists, and activists, that was finally able to “solve” the problem. The combination of this new actor, the new ecology created by deinstitutionalization, and the capacity of the therapies to secure the cooperation of the patients themselves, is what led to the autism “epidemic.” (p868).

In this chapter, I describe a new form of expertise: tool-building. Taking up Eyal’s injunction to locate forms of expertise within the actor-networks that enable it, I show that the rise of the software-as-platform, and its accompanying normative logics, is responsible for this emergence. While tools, within the software-as-platform, are usually software programs, they become easier to build because platform architects provide interfaces (GUIs and APIs) to tool-builders. Making coding simpler has long been a goal within the world of software
Yet, the ability to write code is by no means the only thing that makes successful tools possible: the ability to build tools is contingent on access to the core of the software-as-platform. At the same time, the fetishization of tools, as objects through which humans craft themselves, comes from a very specific conception of programs and software, related to the development of Silicon Valley.

I argue that tool-building is both a way that a group of experts conceive themselves as well as their theory of expertise. Thus tool-builders may see themselves as hackers, coders, or designers who build to make some form of activity possible for their intended users. Their expertise, in some sense, is derived from their ability to use interfaces (through coding or otherwise), and from their closeness (or not) to the core of the software-as-platform. It is not derived from their closeness, or control over, a certain body of knowledge. At the same time, tool-builders see themselves as channeling a whole host of other experts—the so-called “domain experts”—whose insights they seek to build into the tool. Thus the protagonists of this chapter, computer scientists interested in doing research on learning, see themselves as tool-builders making tools for learners, instructors, and others, but channeling into these tools the insights of domain experts like learning scientists, organizational theorists and social psychologists; while the goal of their tools is open-ended, they are also encoded with insights that will, in theory, enable the establishment of good teaching and learning practices. At the same time, these tool-builders, by building and distributing these tools, are also articulating a theory of expertise. In this theory, providing people with tools that are both open-ended as well as embedded with insights is the surest way of turning these people into experts.

5.2 Computer Scientists as Exemplary Tool-Builders

While tool-building, as a framing device as well as an actual enterprise, is a ubiquitous activity in the world of MOOCs, and is carried on by, among others, instructors, platform architects, and external developers, I concentrate on computer scientists, working in
universities, who see MOOCs as a research site for doing research on teaching and learning as well as to build new kinds of tools and programs for learners and instructors. The discipline of computer science dates back to the early post-War years after the building of the first digital computers. Academic computer science was built by scientists (psychologists, mathematicians, physicists) who found themselves fascinated by scientific computing. Not surprisingly, early computer science was theoretical; interested in creating abstract theories of computation, and early computer scientists were often mathematicians, who were interested in grounding programming in mathematics and mathematical logic, interested more in the “science” of computer science (Ensmenger 2010, 111–137). While some of these early researchers, in their attempt to professionalize, saw themselves as studying computers, others suggested that the emerging field was about something much larger: algorithms, information, computation, intelligent systems and so on, as something more than just computers themselves. In the 1970s, the Association of Computing Machinery (ACM) proposed that that CS was about the study of "information": how it was created, propagated, used etc. What became the settled consensus was that CS was the study of algorithms (structured processes that could often be represented via computer programs), which could be abstracted out of their computing medium and seen everywhere in the world (Ensmenger 2010, 111–137). However, programmers immersed in the world of data processing in corporations often saw academic computer science as far too immersed in theories, with little practical importance. Contemporary strands of academic computer science, however, look far different. Computer scientists are increasingly part of the “academic capitalism” (Slaughter and Rhoades 2009) that marks the new neoliberal phase of US universities since the 1980s; they serve as consultants, on boards of directors, and are increasingly founders of new start-ups that commercialize experimental technology. It is not surprising then that MOOCs themselves arose out of experiments with technology that were carried on at Stanford
University by computer scientists, and then turned into for-profit start-ups with venture
capital. In this section, I tell the stories of two graduate students, from MIT and Stanford,
who conducted research with MOOCs; my goal is to show how computer scientists turn
Teaching and Learning into a domain into which they can intervene.

5.2.1 Juho Kim and Learnersourcing
In the summer of 2011, Juho Kim, then in the first year of his computer science PhD at MIT,
going off to do an internship at Adobe's Creative Technology Labs in Silicon Valley and
experienced something like a revelation.40 Juho had come to MIT with an interest in
"supporting people's creative tasks with better-designed tools41 and processes"; for him,
creative tasks meant "writing, programming, designing, even doing research." His special
interest was in building "crowdsourcing" workflows that aided people in performing these
creative tasks. He expected that his time at Adobe would be spent designing a new feature
for Photoshop, Adobe's signature tool for image processing, used extensively by artists,
graphic designers and other design professionals. Imagine his surprise on reaching Adobe
where he found that building new features was not a top priority, because Photoshop already

40 Interview with Juho Kim, 5/16/2014. This subsection is based mostly on the interview. I also conducted
participant-observation with Juho's group at MIT: the User Interface Design (UID) group headed by Rob Miller
from Sep 2013 to May 2014. In addition, I attended the Learning at Scale conference in both 2014 and 2015;
Juho was the first author of one paper in 2014 and was on the program committee in 2015.
41 Note the language of tools. The world of Human-Computer Interaction that Juho hails from is saturated with
this language of tools.
had hundreds of features though even Photoshop experts used less than half of them, while still managing to accomplish all sorts of intricate tasks. No, Adobe was interested in making Photoshop easier to use, or as Juho put it, “more learnable.” As a research project, Juho built a chat-room for Adobe, enabled with voice and screensharing, where novice Photoshop users could ask questions of expert Photoshop users (conceived as a crowd)—“connecting people to give peer(-to-peer) help in using Photoshop” (see Kim et al. 2012). The project satisfied both his interests: helping creative people by building tools, and using crowdsourcing methods innovatively. But it also led to something like a personal research revelation. He had found a “missing space” where computer science methods could intervene: “how can we make novices pick up those skills that experts already have in a more, you know, seamless manner?” Juho saw novice Photoshop users as “learners,” expert Photoshop users as instructors, and the process of connecting the two roles—to help novices pick up expert skills—as a crowdsourcing problem. He had found a new domain of intervention for his research that combined all his interests: learning.

Figure 5.1: A how-to video on YouTube that shows, through a step-by-step process, how to add blurs to an image in Photoshop.
As Juho worked on his “Photoshop with Friends” application, he realized that the study of learning was a “missing space” which his research could intervene in. The deficiency of his chat-room work, Juho felt, was in the availability of the experts. Short of establishing a call-center—“but that didn’t really seem like the path I wanted to take”—how could he create a standing reserve of expert Photoshop users that novices could draw on at will? Taking a further step back from this formulation, he focused on what Photoshop novices actually did when they needed help; he found that they looked up resources on the Web and that how-to tutorial videos on the Web (see Figure 5.1) were a key resource for them. “It’s maybe not as powerful as having someone next to you,” he thought, “but we can overcome the difficulty of finding the person on the fly so it felt like, you know, a nice tradeoff and a worthwhile tradeoff to make.” And yet, these videos had their problems: “some of those videos are just very poorly created, [they are] hard to search [for], hard to find, hard to find specific parts of the video that are just relevant to you, so all sorts of limitations came up, and those actually turned into research questions. So my thinking was: [it] might be much harder to find people than to fix videos [laughs].” His research question became: “how can we turn [these plentifully available how-to videos] into useful learning materials in your own context and so that’s how my whole current line of research on online learning videos started.”

Having decided that how-to videos on YouTube would be the focus of his research, Juho spent most of 2012 building new tools and techniques to process these videos to make them better learning resources (see Nguyen, Kim, and Miller 2013; Kim 2013); the domains he explored included Photoshop, cooking, and makeup; these domains involved, as he put it, “skill learning” or “procedural learning” if not “conceptual learning.” His key insight was that learners needed to be given a high-level decomposition of the key steps in these how-to videos if they were to make best use of it; he called this “subgoal labeling” or as he put it to me, “reverse engineering an existing video to turn it into a bullet point outline.” With such a
high-level summary, learners could easily tell if the video was relevant for their purposes, and if so, they could handily navigate to the portion of the video that was most relevant to them. He applied a battery of techniques (image processing, video analysis, text mining, natural language processing, and even crowdsourcing, on which more below) to already-existing how-to videos that decomposed them into their subgoals, and turned them into more digestible units for novice learners who wanted to use them. He built a video browser that he called Toolscape that summarizes how-to videos and displays “visual summary of the [how-to] procedure, providing step-by-step information such as intermediate results and actions in between.”

In this time, he also watched the “Year of the MOOC” unfold all around him in Cambridge, Massachusetts as MIT and Harvard launched edX. And MOOCs, he realized, were the perfect vehicles for his research. As he told me:

[MOOCs] just seemed like a great experimental platform for me because it just has all the components that I was excited about. It leveraged the web resources, reaches out to all these people, has a crowd, that, it has all sorts of opportunities for things like crowdsourcing, or it’s all about making things scalable, and it’s got tons of videos that I can look at so it seemed like a perfect combination of everything I’ve ever wanted. And another sort of more strategic decision or more practical thing was that how-to videos are, you know, a nice subset of educational videos. When people talk about education, people normally think about classes, math, science, exams, homework and how-to videos have none of that. So if you want to really say something about online education, you might either need to go into like schools and classrooms and look at exam questions but then MOOC seemed to be a nice middle-ground for me [...] because of all those reasons and I thought: MOOC videos might be interesting to look at as our next line of research after how-to videos.

In this account, Juho is expressing a number of complicated points, fusing technical and institutional logics. First, his realization that he would never be taken seriously as researcher of online learning if his analysis did not include formal educational videos like lectures. Yet, finding ways to figure how these videos were used was institutionally difficult because of the problem of access. MOOCs—by fusing the formal and the informal in education—had created a “nice middle-ground” for researchers like him. Studying MOOC videos and
building new interfaces for them was a “strategic” decision for Juho: it allowed him to both claim formal learning as a field for intervention as well as blur the common-sense distinction between how-to videos on YouTube and MOOC videos by fusing them into a category of “educational videos.”

His next steps were clear to him. He secured an internship at edX in the summer of 2013. While his qualifications for it are undeniable, he was helped by the fact that he and his adviser already had links to the edX organization—which emerged right out of the MIT computer science department and which, in its initial incarnation as MITx, was housed right in the same building that Juho worked. He also considered internships at Coursera and the Khan Academy. As an MIT researcher who knew people at edX, he and his adviser felt that working for Coursera would be inappropriate. He liked the Khan Academy but was unsure whether they would let him pursue research there for his internship (which he preferred). Of course, it also helped that edX was closer and he would not have to move from Cambridge, where he already was.

At edX, he conducted a data analysis of edX lecture videos for four initial edX courses from 2012. This involved yet another step in his research. In his prior work on decomposing how-to videos into their high-level constituent sub-goals, he had relied on content-analysis techniques from AI and crowdsourcing. Now, he had access to data about how learners used these videos, something that he couldn’t possibly access before because the data on how YouTube videos are used is zealously guarded by YouTube. He found in his analysis that there were certain patterns of interaction that corresponded to certain forms within the video. For e.g. a slide transition might lead learners to seek back in the video timeline to understand something from the previous slide; correspondingly the duration of the video timeline just prior to a slide transition was heavily clicked on. Juho called these heavily clicked-on regions of the video “interaction peaks.” Mining the video interaction data thus allowed Juho
and his collaborators to understand which areas of the video might be more relevant for learners (Kim et al. 2014). He then used the insights he learned from his edX work to build LectureScape, a video browser for lectures, a variation on Toolscape, his tool for browsing how-to videos (Kim et al. 2014). Unlike Toolscape, which relied on video content analysis and crowdsourcing, LectureScape relied on its built-in “data-driven interaction techniques”; it uses existing video browsing data to make guesses about what parts of the video were important to learners.

Juho hopes that LectureScape will eventually be integrated into the edx.org website (i.e. the “core” software) rather than just being available as a standalone tool. The process of integrating new tools like LectureScape into the core software, he argues, “should just follow the process of you know me having this vision, and then implementing it, and then convincing people who can actually make it happen at scale.” In that sense, his trajectory shows how new tools can be integrated into the software-as-platform. He gained access to an institution (edX) and its “core” data, built a tool that makes use of this data but which is available to edX learners and instructors, and now hopes that his tool will eventually be folded back into the core of the software. But integrating the tool back into the core software is easier said than done. "[T]here are a lot of steps that we have to take to really get it out there. So we have an internal version working but we have to go through the accessibility
Juho reconciles all of these disparate strands of his research on educational videos, broadly construed, through his concept of “learnersourcing,” a variation of crowdsourcing, that he constructs as his central contribution, both in his dissertation and his research statement that he used to apply for computer science faculty jobs. In his thesis abstract, he defines the concept as “a form of crowdsourcing in which learners collectively contribute novel content for future learners while engaging in a meaningful learning experience themselves.” Both points in this definition hold deep meaning for computer scientists. In learnersourcing, “learners contribute novel content for future learners,” in the same way crowds aid the process of heteromation by contributing content that serves as inputs to programs. And yet, unlike crowds, who get a (usually paltry) monetary benefit for their contributions, the process is intrinsically useful to the crowd of learners who obtain a “meaningful learning experience.”

Rob Miller, Juho’s adviser, admitted that he (a pioneer in crowdsourcing research) has never felt comfortable using traditional crowdsourcing methods (using platforms like Mechanical Turk) because the "power relationship between the workers and the requesters is so asymmetric." Learnersourcing was more exciting because "it's a place where we can apply

---

42 This quote is in response to a question that Juho was asked in his job talk for Cornell’s Information Science department. The video of the talk can be found here: http://infosci.cornell.edu/events/colloquia/information-science-colloquium-videos/juho-kim-colloquium-video-2-25-15.
crowdsourcing ideas, where crowdsourcing is really, I mean, necessary, and where the returns to the crowd-workers feel more noble than they do in other areas.\footnote{Interview with Rob Miller, 7/25/2014.}

Juho further divides his approaches to learnersourcing between “passive” and “active” learnersourcing. Passive learnersourcing “uses data generated by learners’ natural interaction with the learning platform” e.g. by collecting “playing and pausing events from the video player.” Active learnersourcing “prompts learners to engage in specific activities, with the purpose of providing pedagogical benefits (for the current learner) and collecting useful information (for future learners) at the same time” e.g. learners may be asked to “summarize video segments they watched, to answer embedded questions, and to explain concepts discussed in the instructional material in their own words” (Kim 2015, 30-31). Through both processes, videos are augmented with metadata that can aid future learner experiences. Juho told me that while he viewed both approaches are fundamentally similar and integrable, the fact that he had applied them to two different classes of videos (YouTube how-to videos, edX lecture videos) had made them harder to reconcile. For how-to videos, he argued, it was very clear that the right abstraction was the decomposition of the video into a set of “steps” that corresponded to particular parts of the video. For lecture videos, it was unclear what the right unit of abstraction was that summarized the video. On the other hand, he had found empirically that how-to videos also possessed “interaction peaks,” parts of the video that
were clicked on more than others. The way to integrate both passive and active
learnersourcing, he argued, was to both understand what caused interaction peaks in how-to
videos and create learning experiences for lecture videos where, by designing prompts where
students explained what they learnt in a video, and collecting and collating that information
from thousands of students, new things could be learned about the lecture itself. This, he
believes, is part of the future of learning research.

Three points must be noted about learnersourcing. First, Juho’s two modes of
learnersourcing, active and passive, bear a startling, and not at all coincidental, resemblance
to the techniques by which online platforms like Facebook both collect data about their users
while simultaneously trying to shape their activity. In a New York Times op-ed titled “How
Not to Drown in Numbers,” two data scientists from Facebook and Google (Peysakhovich
and Stephens-Davidowitz 2015), argue that clickstream data cannot be the only way to
understand people (i.e. users) because it misses crucial aspects of trying to understand the
user experience. They therefore advocate two complementary approaches: “big data” in the
form of clickstream data describing behaviors and “small data” in the form of old-fashioned,
yet strategically administered, surveys. The resemblance between these methods and Juho’s
passive and active learnersourcing should be clear. It should also be clear that these
approaches travel between industry and academia through internships and jobs; there is a
revolving door between computer science academics and those in the industry.

Second, by using learnersourcing methods on broadly construed category of “educational
videos,” which range from how-to videos found on YouTube about various informal topics to
actual lecture videos to teach formal subject-matter, Juho’s work can be put to work by
multiple institutions. Learnersourcing is not just relevant to higher education institutions, but
also to Amazon or YouTube who both traffic in educational videos, but targeted to informal
learners like consumers and the general public. Contrast this with the work of learning
scientists explored in the last section who construe their domains of intervention very differently, typically in terms of particular subject-matter or disciplines: mathematics, physics, chemistry, aerospace engineering. By taking up both formal and informal learning as just points on a continuum, computer scientists slice up the world in a radically different way that allows them to appeal to multiple constituencies, namely institutions that traffic in both formal and informal learning.

Third, while Juho’s research has been the most ambitious outcome of learnersourcing, the concept has been used by other researchers associated with the edX platform in multiple ways. Piotr Mitros, who works for edX and has the title “Research Scientist” (and who was Juho’s mentor during his internship there) has argued for “learnersourcing of complex assessments”: in this argument, by recruiting top learners of a course, instructors and curriculum experts can enable a large-scale creation of assessments that can improve the course itself, while facilitating better learning outcomes for the learners who contribute to this process (Mitros 2015). Elena Glassman, Juho’s fellow advisee at MIT, used the concept for extracting “personalized hints”: that the knowledge of students created through “their own experience struggling with a particular problem” can be transformed into “hints for fellow students” (Glassman et al. 2016).
5.2.2 Chinmay Kulkarni and Peer Assessment

PeerStudio is an influential peer assessment tool used in MOOCs. It was built in 2012 by Scott Klemmer, then a professor at Stanford, and his graduate student Chinmay Kulkarni in collaboration with Coursera engineers. Every learner was asked to assess three to five of his peers’ work by working through an instructor-provided rubric; a grade for each assignment was then calculated as a weighted average of the peer assessments that it received. The arrangement was based on reciprocity; learners received a grade only if they had assessed a certain number of peers. By having the staff grade a set of assignments, and then making sure that of the five assignments to be graded, each grader got one assignment that was staff-graded, Kulkarni and Klemmer were able to prove that peer grades corresponded roughly with instructor grades (Kulkarni et al. 2013). Coursera could now safely plow ahead with its plan to host a variety of classes of various stripes; peer assessment allowed Coursera to significantly expand its course portfolio.

In this section, I tell the story of how PeerStudio came to be built and used. Scott Klemmer’s interest in self and peer assessment began in 2005 when he became interested in using the Studio method of design schools in his HCI class at Stanford for computer science undergraduates. In the Studio method, students spend some time in class in a studio-like

---

44 Klemmer and Kulkarni helped Coursera design its initial peer assessment system. Later, they built their own independent system—PeerStudio—once again, a tool, that could be hooked into other systems, Coursera’s or edX. This allowed them, as researchers, to tweak the system so that they could conduct research on teaching and learning practices.
setting working on their projects; the work of their peers is immediately visible to them and there is the possibility for a great deal of loose interaction. Immediately, a problem became apparent: the fairness of the grading. Students in his class—since they came from a more quantitative computer science background—questioned sometimes the necessarily qualitative and subjective standards by which the grades were assigned. While initially, Klemmer, who comes from a design background himself thought, "Wow, the amount of feedback we gave these students was so much better than I ever got at RISD. These ingrates don't know what they're talking about," he later came around and decided that the students may have a point. Over the next two or three years, he tried several methods to make the grading process more transparent.\(^4\)

In 2009, Klemmer became interested in self and peer assessment as a teaching tool and read David Boud's (a learning scientist) book "Enhancing Learning through Self Assessment," (Boud 1995), a book that he recommends enthusiastically to others who are interested in using peer assessment as a tool in their own teaching. In his class for the fall of 2010 at Stanford, Klemmer introduced a tool for self-assessment that he built in collaboration with his graduate students. Every week they followed a ritual. Students would attend a 12-15 people studio every week; at the end of the studio session, students would assess their own

work using a rubric created by Klemmer and his TAs. On the weekend, the TAs would grade what the students had submitted without looking at the student’s name or the self-assessment. They followed a simple and transparent rule (which was announced to the students): if the student’s self-assessed grade was close to the TA’s grade, the student would receive his own grade; otherwise, he would receive the TA’s grade. To his surprise, Klemmer found that there was good agreement between the self-assessment score and the score assigned by the TA.

In the fall of 2011, as the next batch of Stanford MOOCs were being prepared (after the success of the first three), Andrew Ng asked Klemmer if he wanted to teach his HCI class online to a more mass audience. Klemmer replied that he would be interested in doing one in the spring of 2012 but crucial to making the class work would be to build a scalable peer assessment system. Over the next few months, Klemmer worked together with Daphne Koller and other Coursera employees to design such a system. The graduate student who played a key part in this design was Chinmay Kulkarni.

Chinmay Kulkarni came to graduate school at Stanford in 2010 interested in HCI, after a previous stint working in programming languages. He preferred HCI, he told me, because he wanted to work closer to the interface between users and the system. \(^{46}\) When Kulkarni joined Klemmer as his TA for the proposed Coursera class, he had a different goal in mind; his

\(^{46}\) Interview with Chinmay Kulkarni, 4/20/2015.
research interests, like Juho Kim’s, lay in building tools to aid creative people. By helping Klemmer develop the peer review system for Coursera, he thought he would be able to get access to a corpus of creative work. Since in every week of the class, the students would be taking one step towards designing a full-blown smartphone app, Chinmay thought that he would get a dataset where he would be able to analyze, for instance, how the first step in designing a smartphone app looked for many people, and then the second, and so on; a corpus that could help him design tools that aid creative work. Unfortunately, reality belied his expectations: there was too much variation in what their learners did that the corpus itself had little value to him, programmatically. But along the way, he realized that his considerable design talents might be applied to the world of education itself. “Actually it’s kind of neat to work on the education thing,” he remarked to me because “everything was pretty broken.”

From that beginning, when they designed PeerStudio in collaboration with Coursera engineers, and showed that the peer assessment grades were roughly equivalent to instructor-assigned grades, Kulkarni and Klemmer went on to do a number of ingenious experiments, using the iterations of their own HCI class on Coursera. (Klemmer has referred to this time as “the most exciting time of his life.”47) They gave peer graders feedback on how their assessments compared with instructor assessments. They experimented with ways to encourage learners to give each other more qualitative feedback, settling on offering learners

47 Keynote speech delivered at the Learning Analytics and Knowledge (LAK) conference in March 2014.
pre-formatted lists that they could check off; here they relied on the finding that there are certain kinds of commonly-found errors in particular assignments. They built a machine learning program to assess and grade short-answer questions (Kulkarni et al. 2014). The catch was that the output of this program would not be used for actual grading; rather, the output of this program—the grade and the certainty with which it was computed—would be used to decide the number of peers needed to assess it. The more uncertainty the program expressed in its result, the more the number of peer graders required. This arrangement reduced the overall amount of peer assessment required, which was important for MOOCs because it is a time-consuming activity for learners. In a subsequent paper (Kulkarni, Bernstein, and Klemmer 2015), they modified their system so that learners could ask peers to assess drafts of their assignments. A similar reciprocal system was designed: a learner was allowed to get feedback on his drafts if only he offered feedback on drafts of other learners. The system yielded two benefits: first, the scale of MOOCs meant that most learners got some form of feedback on their drafts within a few hours. Second, and consequently, these learners ended up with better learning outcomes, in this case, better grades.

Chinmay Kulkarni remembers the early experiments as a classic "hacky" endeavor. At the time, they did the first experiments, he told me, Coursera did not have an experimentation (or A/B testing, as it is commonly known) feature, so he essentially improvised by adding code to certain parts of his course. "At the time, Coursera didn't actually block JavaScript. So I could basically insert JavaScript which would change the interfaces that people saw." He attributes his competitive advantage to the fact that he was willing to do hacky things. "A lot of people at the time wanted to run these A/B experiments but none of them really wanted to muck around with writing JavaScript. So given that I didn't care, and I could write Javascript, gave a nice leg-up so we could do the research much faster than if we had waited for Coursera to add an A/B support." He added that this was definitely one way that "outsiders"
like him differed from traditional education researchers who would certainly not have thought of doing what he did.

In doing these experiments, Kulkarni and Klemmer (assisted at different times by undergraduate research assistants and other TAs) were performing two roles simultaneously. They were both researchers and instructors; designers and users; they were building software tools for themselves to use. This is another characteristic of the hacker ethic: using software that you design gives you more understanding of its faults; in the software industry, this is sometimes referred to as “eating your own dog-food.” It also allowed them to develop PeerStudio rapidly: the cycle of design, testing, iteration, reiteration, went on furiously, along with a number of experiments to determine what teaching and learning practices produce the best learning outcomes.

Of course, Kulkarni and Klemmer did not build PeerStudio just for themselves; they meant it to be used across classes. They therefore also started to recruit other instructors who were interested in using PeerStudio for their classes (MOOCs or on-campus) as well as in research to understand the best practices around peer assessment. In this respect too, they adopted a lean design style. Chinmay told me that the instructor interface of PeerStudio was the last thing he built. Rather, with early instructors, he would have them email him the assignment and the rubric which he then inserted himself into PeerStudio. “That gave me some idea of how the rubric is used,” he told me. “It also meant that people could start using PeerStudio a lot faster, and most importantly, it also told [him] what you should build on the instructor end.”

When I asked Chinmay how much he needed to coach the instructors—“teaching the teachers” best practices as his paper had put it—he told me that the instructors he dealt with rarely needed any coaching. He compared what he needed to do with instructors to what AirBnB and Kickstarter did for their users, not coaching per se, but merely providing them
with the best practices. "If you look at say AirBnB’s listing, sort of process, right, they’re not coaching people on how they should create listings, but they are giving you examples, they are telling you, here is a template, here are the things you should fill out, etc etc, right, I don’t know if Kickstarter has changed but they had a similar sort of thing, include a picture, do this, include a video and something, so I don’t think it’s coaching, I think."

Some defining properties of PeerStudio must be noted. Rather than creating machine learning programs that could grade essays, Kulkarni and Klemmer drew on an alternative tradition within computer science: the notion that programs, rather than being autonomous, need to be supervised carefully by humans and that intelligence is a property of humans and programs working together rather than humans or programs alone. Unlike the early days of computer science when creating autonomous programs was the holy grail, many computer scientists today focus on building intelligent systems that consist of both humans and programs working in concert. In this arrangement, the software itself functions as a kind of modulating medium; the software is supervised by humans but it also coordinates the actions of these humans who work with and through it. Ekbia and Nardi (2014) have described this social organization as “heteromation,” a regime of governance where humans, construed as both workers and users, are “fashioned as computational components” for tasks that are critical to computer programs. PeerStudio is a software infrastructure that embodies the principle of heteromation.

PeerStudio is an exemplar of the kinds of interventions computer scientists seek to make in the study of learning; it is an actual software infrastructure used by working courses “in the wild” rather than something that is tried within a laboratory or with only certain kinds of courses. It is also a research tool, both for its architects (Klemmer and Kulkarni) as well as the instructors who use it for their classes. Rather than automating the grading away, it accomplishes it by having learners themselves perform part of this effort in assessing their
peers' work; a design choice its creators acknowledge is drawn from the principles of crowdsourcing. At the same time, they argue that peer assessment is unlike crowdsourcing at least in that the act of assessing one's peers enhances learning outcomes; it has value in itself for the learning process. PeerStudio also preserves a large role for the instructor who gets to both create grading rubrics (crucial to the success of the enterprise) as well as supervise the entire process. The designers of PeerStudio argue that the tool actually empowers the instructor; discrepancies between the instructor-provided feedback and the peer-provided feedback are a signal that there is something wrong with the assignment itself. Finally, in building PeerStudio, its creators both defer to existing experts on learning. From these experts, Kulkarni and Klemmer borrow certain principles: peer and self assessments improve learning outcomes for learners, the quality of the peer and self assessment, and the possibility that it will improve learning outcomes, depends on the quality of the rubric etc.

Making PeerStudio work as a learning tool deployed in MOOCs thus requires careful nudging of both instructors and learners. In another paper, the designers of PeerStudio (Kotturi et al. 2015) describe a series of heuristics to overcome what they call "adoption challenges for global-scale peer learning." In plain terms, they are setting forth a series of rules ("sociotechnical remedies") on how best to get both instructors and learners use PeerStudio for maximal impact. First, instructors should be encouraged to make activities like peer assessment part of the course itself, perhaps by assigning credit. Second, instructors should make learners aware of their responsibility towards their fellow-learners. Klemmer and Kulkarni therefore suggest, building on their own example that designers of systems like PeerStudio should build into their software heteromated tools for emailing learners e.g. instructor-supervised software that alerts learners through an email when they leave a peer assessment incomplete. They conclude with the suggestion that software developers of peer learning systems must try to "teach the teachers" by providing them with successful examples
of software usage by other teachers. In other words, both learners and instructors are users of PeerStudio whose actions have to be carefully regulated through the design of the system itself.

Finally, PeerStudio is a content-agnostic system (just like Juho's systems for "educational videos"). It is meant to be used for open-ended assignments but those assignments could be from humanities, social science, or design courses. It is not even necessary that the system be used for formal educational work at all; it might very well be used for an actual crowdsourcing task. Again, this becomes possible not because of a grand theoretical framework but by having certain key users (in this case, instructors, who know their discipline) customize the software for their own purposes (in this case, by creating the right kind of rubric, the right kind of incentives for learners, etc.). The principles of heteromation get deployed to create a more expansive system not tied to a particular subject-matter.

All in all, PeerStudio embodies the kind of intervention into learning research that computer scientists hope to make. It is a program, a tool, that is content-agnostic and works for any course with open-ended assignments. It allows instructors to customize the software, both in terms of the rubrics that learners are presented, as well as some of the ways that learner activity is structured. Finally, it guides instructors themselves into certain courses of actions, for e.g. by giving them examples of good and bad rubrics. Its creators argue that it empowers both learners and instructors; learners learn better through the act of structured peer assessment; in addition to being spared some of the burden of grading, instructors get a data-driven way of assessing whether a particular assignment is right for learners.

5.3 Comparing Computer Scientists to Learning Scientists

Where do computer scientists fit in the world of learning research? The historian Ellen Lagemann (2002) argues that from its inception in the early 20th century to today, American
education research has been shot through with two key tensions: first, about the methods through which the research is to be conducted, and second, about how this research is to be translated into educational practice (see also chapter 4 of Labaree 2006). Both questions go to the heart of the enterprise. Is educational research to be conducted in laboratories or schools? With qualitative or quantitative methods? By enlisting teachers or leaving them out? These tensions have been re-configured and re-interpreted by new actors who seek to intervene in education research. For instance, Edward Thorndike and other behaviorist psychologists offered very definite answers to these questions in the early 20th century: education was to be studied using measurements, in laboratories, and by psychologists alone without any teacher involvement. This solution was widely accepted in the light of the emerging professionalization and specialization of research within the academy, the growing prestige of psychology as a discipline, and the low status of largely female teachers. Other battle took place during the Cold War over physics (Rudolph 2002) and mathematics (Phillips 2014) curricula, pitting newly empowered disciplinary experts against educationalists from Education schools.

Figure 5.2: A computer scientist's answer to the question of how the field can contribute to the study of learning and the design of educational technologies.
Not surprisingly, in the light of their own high-status as high-flying innovators, and the architects of software infrastructures used by large populations within the US, computer scientists have their own responses to these tensions, which they use to get their foot into the door of learning research. Their answer to both questions is: software. On the first question, of how educational research is best done, they stress building practical software systems that work. They argue that it is only by building systems and trying them out to see if and how they work, and then rebuilding them to fix errors, that (learning) research can truly be done. One of them characterized it as the “build a little, get feedback, correct, build a little more” approach. On the second question of how educational research should inform practice, computer scientists argue that if research results can simply be inscribed into software, then this will structure the practices of learners and instructors; this, they suggest, is simpler than traditional institutional processes. Scott Klemmer referred to this as “baking pedagogy into software” (see Figure 5.2)48. Another computer scientist referred to it as “packaging”—a unit of software and a term intimately familiar to software engineers. As he put it:

*I think the reaching across the aisle challenge [i.e. integrating learning research into practice], the artifact version of it, is how many of the best practices, from the education literature, that are successfully adaptable in this context, or maybe can even be improved in this context, can be packaged into software so that people who do not necessarily know that research still benefit from them.*

---

48 Scott Klemmer, keynote speech at the Learning Analytic and Knowledge Conference (LAK) in March 2014.
49 Armando Fox, computer scientist at UC-Berkeley at the conclusion of the Learning with MOOCs conference, Cambridge, MA, 2014.
The answer to the first tension—build software—arises out of practical considerations and the existing practices of computer scientists. Computer scientists, except those who work in highly mathematical areas like the theory of algorithms, are expected to build practical working systems. They build systems, try them out to see if the systems work, then rebuild them again to fix errors and so on and on. They argue that it is only by a process of rapid iterative design—building, prototyping, measuring how a prototype works, and then rebuilding. Speaking about the world of Artificial Intelligence (AI) at MIT that he was an integral part of in the late 1980s, Philip Agre (1997) writes: “Building things was truly the end purpose of the hacker's work, and everything about the methods and language and value system of the AI world was organized around the design and implementation of working systems. This is sometimes called the ‘work ethic’: it has to work.” Computer scientists may have frameworks, ideas, approaches or theories but none of these matter unless they are also embodied in working systems. Ultimately, rapidly building working systems also takes precedence over extensive planning; as famously written in one of the early documents that shaped the Internet, “We reject: kings, presidents, and voting. We believe in: rough consensus and running code.”

The answer to the second tension—“baking pedagogy into software”—is explicitly ideological in that it specifies a preferred channel of governance. Computer scientists studying learning with MOOCs are intimately familiar with the nudging techniques of social psychology and behavioral economics used in platforms like Facebook, and popularized by policy-advisers like Sunstein and journalists like Charles Duhigg (2014). A survey of MOOC research published in Science points out that “experiments [such as incentivizing learners to participate in discussion forums with the promise of a digital badge to improve overall learning outcomes], often inspired by psychology or behavioral economics, are widely under way in the field” (Reich 2015).
There is a second reason why “baking pedagogy into software” is an ideological assertion. It allows computer scientists to intervene in a domain—the study of learning and building learning infrastructures—without specifying the nature of their own expertise and deferring to established experts whom they configure as “domain experts.” The assertion implies that computer scientists are not experts on pedagogy, yet they can take the lead on building systems while remaining free to pick and choose pedagogical insights to bake into the software, all the while remaining neutral system-builders.

The novelty of computer scientists’ (i.e. the tool-builders) approach to learning research will become clear when we compare them to learning scientists, their closest collaborator-competitors in the study of learning and ask how they viewed instructors. The community that calls itself the learning sciences dates from the early 1980s and was formally established in 1991. The learning sciences community designs what it calls “learning environments.” The term is mainly used for particular kinds of educational software that they design, but it is also used broadly to refer to classrooms and workplaces where learning is also seen to occur. The learning sciences are an interdisciplinary community with the constituent disciplines being “cognitive science, educational psychology, computer science, anthropology, sociology, information sciences, neurosciences, education, design studies, instructional design and other fields” (Sawyer 2006, xi). The goal is to understand the “cognitive and social processes that result in the most effective learning” (2006, xi) and then incorporate these into learning environments. In practice, educational psychologists and cognitive scientists are the dominant group both in terms of numbers and in their success; between 2003 and 2006, the learning sciences secured more than 100 million dollars in NSF funding.

Computer scientists differ from learning scientists in three key ways: their design processes, their relationship to instructors, and their understanding of their own expertise. I explore these differences in the rest of this section.
The relationship between the computer scientists and the instructors (for whom they design) is a designer-user relationship. The computer scientists build tools; the learners and instructors use these tools in order to learn and teach. Mike Hales (1994) has described three kinds of users based on different styles of design activity: users as clients who are meant to be satisfied, users as actor-constructors who are meant to be empowered, and users as co-designers who are meant to reflect on the artifact. In the imaginary of the computer scientists, both instructors and learners fall squarely into the second category; these are actors meant to be empowered through the software. Instructors are also clients. They play a role in the design process but always as users, not as collaborators. By configuring instructors as users, computer scientists ensure that their design process is fast, rapid, and iterative.

Rather than approaching instructors as users in their design process, learning scientists view instructors (or curriculum experts) as collaborators or subject-matter experts. This flows directly from how they conceive of their own expertise. Learning scientists argue that the development of learning environments requires three kinds of experts: (a) Subject-matter experts who know the domain (e.g. high-school algebra). This role could be filled by a curriculum expert and/or a teacher. (b) Learning experts who understand the process of "learning" which is a cognitive process that is independent of the particular subject-matter (i.e. learning scientists themselves). (c) Technology experts who develop and build the system. Note that the same person might occupy multiple roles in an actual project, but this is the often explicitly stated model of the division of research labor (e.g. Corbett and Koedinger 1998). Not every project requires all three experts but the best and most innovative projects require all three. All three expertises together—content, learning and technology—are more than just a sum of their individual parts. Note that there is also a clear delineation of the kinds of expertise required, with content and learning expertise being privileged over technology design. The design process is also long and costly. Learning
scientists working on a particular class of learning environments called “intelligent tutoring systems” report that it takes "200 hours" of labor “to deliver a full hour of instruction” (Corbett and Koedinger 1998). Others have described the process as a “huge investment of time,” (italics in the original) especially for the "content expert" (Walsh 2010).

For learning scientists, the role of instructors in the design process is as collaborators and subject-matter experts rather than users. This is so because for learning scientists, the process of design is about getting the content-expert “to totally re-envision their courses” (Walsh 2010, 100). Moreover this re-envisioning cannot happen without the active intervention of the learning scientist. The content-expert’s perspective is only the “one among many” (Walsh 2010, 100) that are incorporated into the design process. For learning scientists, the design process is also tied tightly to the content, which is often middle-school mathematics, physics, or science. Often, as is the case with learning scientists, when the client is the highly regulated K-12 system in the US, curriculum experts and instructors are actively needed in the design process.

Computer scientists, in contrast, design with the explicit goal of not being tied to any specific content. They therefore allow instructors a good deal of leeway in terms of the instructional material: its goals and its methods. Rather than transforming instructor’s visions, they see themselves and their programs as empowering instructors; in particular, by placing instructors in particular supervisory positions with respect to the working of the software. While they too want to shape instructor practices, they see this happening less through outright transformation and more through nudging techniques, documentation and examples. Instructors and learners, in the computer science view, are like any other platform users: the renters and rentees on AirBnB, the entrepreneurs and funders on KickStarter, or the employers and workers on Amazon Mechanical Turk.
At Stanford's Lytics lab where a group of PhD students in the computer and learning sciences met together regularly, tensions sometimes arose between the two sides, focusing often on the point whether researchers in learning should see themselves as tinkerers or scientists. Typically, it would begin after a talk and demonstration of a more computer science-inspired project. The learning scientists in the room would ask what research question the project was solving. The computer scientists would suggest that the project was fulfilling a need, or a projected need, but that once the system was built, they would actually look for research questions that it could answer. Typically, this led to more discussions on how research questions and hypotheses were to be integrated into system-building.

Figure 5.3: The Lytics Lab workspace, Stanford University. Meetings took place around a whiteboard with around 10-15 people in attendance at every meeting. Photo from the Lytics website.

After one such exchange, when the learning scientists in the room were skeptical of whether the system under discussion was even addressing a research question, a computer scientist sent around an email where he humorously and self-consciously caricatured the two sides and how they think about each other. Titled “On the Politics and Practice of Two Project Design Approaches,” he argued that there were two approaches to system design within the Lytics
Lab, the “top-down” and “bottom-up” approaches and he had observed a preference for one approach, if not outright disdain for the other, amongst lab members, “simmering in our discussions; a latent variable in our midst.” Using the term "social scientist" to refer to the learning scientists, and “technologist” to refer to the computer scientists, he argued that while discussions within the lab had always remained cordial, “wars had been fought over less” and the two sides were in danger sometimes of caricaturing each other. The caricatures, he said, went like this:

Social scientist [on technologists]: These technologists sit around building stuff they think is cool. Then they parade that around, looking for an application. Their naïveté about what humans want, or can work with, and about what's important is bottomless. Also: they exhibit a glaring inability to speak in terms others can understand.

Technologist [on social scientists]: Social scientists just ridicule what we build; I have yet to hear solid proposals for what might work better. They sit around for months, arguing about obscure theories. If they ever do produce an outcome, they exhibit a glaring inability to speak in terms others can understand. Their naïveté about what systems can, and cannot do is bottomless. They think all it takes is to hire a few code monkeys who will design and implement software that works, is extensible, and can be maintained. And then they don't even have money to hire those monkeys.

In the caricature, the learning scientists find “cool stuff” system-building pointless unless there is a research question; on the other hand, the computer scientists find the learning scientists too immersed in their abstruse theories and utterly ignorant of the time and effort it took to build and maintain complex systems. He went on to argue that both top-down and bottom-up approaches to system design had their advantages. In particular, as a computer scientist, and with a partiality towards bottom-up approaches, he found that bottom-up approaches that used rapid iterative design were tremendously generative, raising new and new questions that a top-down theory-driven model of inquiry would not be able to attain. In particular, if learning scientists and computer scientists were to collaborate, he argued, the research “must excite both parties. A setup of ‘we decide what is important, you implement’ is as unfortunate as ‘I built this thing; it's done, eat it’.”
As the incident above shows, the relationship between computer scientists and learning scientists is complex. In their work, computer scientists turn learning scientists into domain experts whose theories they test and implement in particular settings. (Computer scientists also draw on domain experts like social psychologists, management theorists, among others.) Domain experts are deferred to when it comes to theory, but not necessarily about questions of what systems to build or which theories to test; that choice is for the system-builders, i.e. computer scientists, to enact. Even as domain experts, learning scientists do not get full authority: for instance, computer scientists argue that it is an open question of whether these theories would “scale.” In his keynote at the LAK conference in 2014, Scott Klemmer complained that the education community, like the design community, often falls back too often on the answer “it depends.” He argued that education research is really difficult to understand, and especially to separate the wheat from the chaff. Domain experts could help computer scientists, he argued, by contributing, for instance, the ten most important things that educators should know.

One computer scientist told me that he thought learning scientists were too scientific, that they wanted things to be too perfect but sometimes it was better to wade in quickly and accomplish something that was quick and dirty. Software could never be perfect but it was important to get in there and build it. Computer scientists also characterized learning scientists as too closed off; too interested only in thinking about technologies that helped them modify their theories a tiny bit, and closed off to other technologies that wouldn't fit into their theories.

The learning scientists, for their part, argue that the computer scientists do not draw enough on their work. They are equally critical of the experiment-till-you-get-it-right ethos of the computer scientists. As one learning scientist put it sarcastically, referring to the tendency of the computer scientists to be unacquainted with prior work done in the learning sciences: why
spend an hour in the library when you can spend twenty hours in the lab? And yet, the learning scientists also see in MOOCs a promising research infrastructure to produce new knowledge about learning, as well as a site to deploy their own hard-won knowledge on the topic of learning.

One computer scientist told me that the difference between computer scientists and learning scientists was really a difference in world-view. As a computer scientist, he was definitely interested in inscribing learning science insights into his systems; but what he did not build into his systems was their worldview. As he put it:

I think it's a clash of views in some sense. So, in some sense, [long pause], this is generalizing and only slightly true, but the learning science view of the world is: everything is contextual, we have to be extremely deliberate about everything that you do, and you have to be careful about the recommendations that we make, and we have to be careful about the changes that we create. In contrast, the computer science sort of view of the world is yes, everything is contextual but 80% of it is the same, and it is better to have something out there faster than it is to have nothing out there at all. And well, okay, we agree with you that video lectures are a crummy way of teaching, but look, we can actually teach so many more people at least a little bit and to us, that is valuable. To the education people, I don't think that teaching people just a little bit is actually what they want to do. Right, they want to really teach people. [...] And when I take on information, and theory, and findings, from the learning sciences, I'm careful to tell you that I do that but also, in some sense, I don't take their worldview, right, I take the notion that yes, this is one kind of feedback that is more valuable than this other kind of feedback, we should build this sort of thing into our systems, but I also don't stay true to the notion that all of this feedback is contextual, and you want to be building things which are very very contextual, right, so to me, it's a little bit, so I take their theory but not their worldview. [my emphasis]

As this quote shows, the difference between the two communities can be traced to their fundamental identities as researchers: computer scientists see themselves as hackers and tinkerers, dancing on the edges of disciplines and systems, while also producing sound knowledge claims; knowledge claims must be mediated through actual systems. Learning scientists see themselves as scientists, arguing that the best way to do educational research is to use (cognitive) learning theory as a driver for formulating research questions. Unlike computer scientists who are proud of their hacky, lean, and fast design process, learning
scientists almost glory in the fact that their own design process is long and costly, both in terms of time and resources put in. The difference also manifests in their research output; learning scientists publish, like most scientists, in peer-reviewed journals. Computer scientists publish in conferences, which, for most part, function like de facto journals requiring full papers, double-blind peer review, and very low acceptance rates, but generate a very large number of papers every year. The annual conference calendar and the hacker-ethic of rapid software development accounts for the often astonishing productivity of computer scientists. Some of my interlocutors submitted as many as 4 first-authored papers to the ultra-prestigious CHI (Human Factors in Computing Systems) conference in the same year.

5.3.1 The Politics of Expertise
Institutionally, these disputes between computer scientists and learning scientists play out in research group meetings, conferences, within dissertation committees, and in front of funding agencies. In the rest of this section, I focus on the emergence of one such site: a new conference on learning that aims to bring together the two groups. In 2013, "inspired by the emergence of Massive Open Online Courses (MOOCs) and the accompanying huge shift in thinking about education," a group of computer scientists came together to create a new conference called Learning at Scale (henceforth L@S). The goal of this new conference was to promote "interdisciplinary research at the intersection of the learning sciences and computer science." The primary impetus for L@S came from computer science's premier professional body, the Association of Computing Machinery (ACM). The first conference was held in March 2014 in Atlanta and deemed successful by the organizers; the conference is now in its third year and going strong. The creation of an ACM-sponsored and affiliated conference is an important event in the world of computer science because conferences occupy a somewhat unique place in the field in a way that they do not in other disciplines; conferences indicate that a particular topic is important for the community in general. A
conference that is high-quality and highly selective (goals that the L@S organizers openly profess) is an even bigger signal. By founding a high-status conference to conduct research on teaching and learning, computer scientists were signaling to each other—and to other educational technology researchers including learning scientists—of their interest in the topic. The computer scientists mentioned in the previous sections—Juho Kim, Scott Klemmer, Chinmay Kulkarni—have been key participants in this conference, serving on the program committee, and presenting highly well-received papers there.

The organizers of L@S saw themselves as faced with three major challenges. First, they wanted the conference to be not just about MOOCs but about something broader, but something that could be legitimately seen as a topic of computer science. Second, they wanted to "plant a flag" and bring together all the various sub-disciplines of computer science who worked on learning infrastructures in one place since these sub-disciplines usually published in separate conferences. Third, they wanted a "high-quality" forum of participants from the both the learning sciences and computer science.

In general, the tone of the L@S conference “Call for Papers” (CFP) has been conciliatory and deferential towards existing experts in the learning sciences. The goal of the conference is always stated as encouraging the building of practical systems (with learning science knowledge encoded into them), and rarely, if ever, is there any mention of theory. When I interviewed computer scientists who took an active interest in the conference, submitting papers there, they admitted upfront that they did not consider themselves experts on learning, or learning scientists. They stressed that they wanted to collaborate and bring in learning science findings into their systems. Juho Kim, for instance, told me that he did not see himself as a “[learning] theorist” or an “educational researcher” but his vision and his expertise lay in “building new things that enable new ways to learn.” His contribution, as he
saw it, was to take classroom theories of learning and see if they held up in the wild (i.e. on the Internet).

Yet, tensions between the two communities have also come to the fore at particular moments through the conference from its original Call for Proposals (CFP) to the question-and-answer sessions that play out after paper presentations. The original CFP for the 2014 conference (which went online in 2013) contained a striking passage that suggested that researchers who developed “intelligent tutoring systems” or ITSEs should submit to their own conferences rather than L@S:

All topics must tackle topics "at scale." For example, a paper that would not qualify for Learning at Scale would be one about an intelligent tutoring system that behaves no differently with one student than with thousands, or which does not improve after being exposed to data from previous use by many students; such work should instead be submitted to a conference like ITS[Intelligent Tutoring Systems] or AIED [Artificial Intelligence In Education].

By adding this paragraph to the CFP, the L@S 2014 organizers were accomplishing two things. First, they were setting out a definition of “scale.” Second, they were trying to create a space for new kinds of research that was not intelligent tutoring systems, the most prominent way that computer scientists had intervened in education research until then.

The first point is about “scale.” The definition of “scale” here is deeply oriented towards current times. As Chris Kelty (2000) has pointed out, engineers use “scale” in an arguably unique way: scale means something being small and large at the same time, and elegantly so. The term is used these days not just to software but to business models and organizations.

ITS researchers and learning scientists that I talked to objected to this definition of “scale” arguing that their systems used existing data about learner usage to improve its performance and this should count as “scale”; the point of this definition, they suggested, was to discourage them from submitting papers to the conference. The L@S organizers disagreed, arguing that merely improving the system’s performance using existing data was not the
point; scale meant an "order of magnitude change" to a "metric of merit" that transformed a "quantitative change" into a "qualitative one," that "allows you to do things that weren't possible before."

But, and second, it is entirely possible that the point of the paragraph was also to provide a space for other kinds of learning researchers and ensure that the L@S conference would not be overrun by ITS researchers and would have its own independent identity. The ITS community is the oldest community within the learning sciences where computer scientists play a prominent part. This community usually meets in two conferences that alternate with each other: the ITS conference and the AIED conference, both of which date from the mid-1980s and have a substantial attendance. The ITS community is organized around Carnegie Mellon University (CMU) as its focal point: CMU researchers have produced both the algorithm which is most widely deployed (with other researchers producing endless variations) as well as the data that many researchers use. The L@S organizers saw two problems with these conferences. First, that "not many first-rate CS people go to them (with no offence to those who do)" while "the L@S roster is a who's who of some parts of CS."

But also, and equally important, they wanted a forum of participants who built "end-to-end systems"; they did not want a forum where existing algorithms were simply applied to educational data which they saw as increasingly the case with the ITS, AIED, and EDM conferences. ITS researchers had their own problems with the framing of the L@S conference, ribbing that it was better thought of as "teaching at scale" since the papers were all mostly about supporting learning rather than learning itself.

The offending paragraph was considerably watered down in the CFP for L@S 2015. Instead of "intelligent tutoring systems," it merely says "systems." And the line about submitting the paper to AIED or ITS was taken out completely. That year, there were 4 ITS-centric papers in the main proceedings.
Chapter 5: The Tool-Builders: Computer Scientists, MOOC Infrastructures, and the Emergence of a New Expertise

All topics must tackle topics "at scale." For example, a paper that would not qualify for Learning at Scale would be one about a system that behaves no differently with one student than with thousands, or which does not improve after being exposed to data from previous use by many students.

The L@S conference has presented computer scientists, who have taken the leading role in its organization, with the question of how best to collaborate with learning scientists, whether by merely drawing on their work, or by explicitly devising joint projects with them. Learning scientists (including the ITS community) face a similar problem: how best to influence the system-building of the computer science community, and how best to gain access to the infrastructures of MOOCs to try out their ideas. These questions are complicated by the fact that the two groups, as we have seen, have different working styles, very different approaches to how they conceive of their work, and of course, different identities. For instance, at the first L@S conference in 2014, there were at least three points when learning scientists (or education researchers) walked up to the microphone to express their frustration that the discussions were not incorporating more existing work in the learning sciences. Lori Breslow, the director of MIT's Teaching and Learning Lab used the most striking anthropological imagery about "tribes" and "cultures" to refer to the two groups:
A Final Bit of Advice

- Make sure you are building on what is already known about learning processes, fostering learning, assessing learning, and/or designing learning experiences.
- Make your primary goal to address an important set of issues; technology as the way to do that
- Assemble a team with all the needed expertise!!!
- Involve learning scientists in design of your innovation and its use, not simply used as evaluators and methodology people
- Make design and refinement a first-class piece of your research and development plan (become familiar with Design-Based Research and Design-Based Implementation Research)

Figure 5.4: The Program Manager of NSF’s CyberLearning Initiative advises computer scientists on the best approaches to securing NSF funding.

I’m on the educational side of the house [...]. I’m working with a lot of computer scientists so over the last year and a half, sometimes I feel like I’m doing cross-cultural communication [laughter from audience], so I wonder if as we begin the next couple of days, if you [referring to the keynote speaker] could give us some advice about how these two different tribes can best work together.

This question of what the best model of collaboration depends on another factor: funding. The learning sciences have been tremendously successful in being funded by federal agencies like the NSF, to the extent that one could argue that the field itself was created by NSF interest in teaching better science and mathematics to middle-school students. At L@S 2014, one of the invited talks was by Janet Kolodner, a founder of the learning sciences and then the Program Manager of NSF’s program on "Cyberlearning: Transforming Education," gave a talk on funding opportunities at the NSF. Kolodner was very clear that grant applications would not be funded unless they cited the relevant literature or included the relevant learning science expertise.
Finally, a little bit of advice. Based on some of the things I've heard here. I think that it's really really important for any of these programs to make sure you're building on what is already known about learning processes: fostering learning, assessing learning, or designing for learners; all of these programs that I talked about require that and it's really important not to be technology-centric. Well, if you're dealing with infrastructure, you're technology-centric, I understand that, but otherwise, make sure that your primary goal is to address an important set of issues with the technology as a way to do that. Even a platform, an infrastructure, that's the way to do it. Make sure you assemble a team with all the needed expertise, you don't have to know all those journals I mentioned earlier, if you have people on your team who know those journals.

While computer scientists have taken the lead in building learning infrastructures, they are still negotiating the framing of their involvement with the study of learning. The L@S community has been building on its key asset: the presence of world-class computer scientists from elite institutions. Yet, it has had to negotiate three main issues: (1) how to integrate existing learning science work while still maintaining a distinct identity, and indeed, taking the lead in the field, (2) how to collaborate with learning scientists, rather than just drawing on the literature, and (3) how to negotiate with funding agencies whose officers are likely to be learning scientists themselves and who are likely to demand such collaboration. The future of the L@S conference and its community of practitioners have much to tell STS about negotiations of expertise, and the rise of computer science as a regime of expert knowledge.

5.4 Conclusion: A New Form of Expertise?
Computer science as an academic discipline has always had a porous boundary, and the field itself is full of what might seem to be paradoxes. It is a "science" but most of its practitioners make and maintain new kinds of software; they are technologists. They privilege systems rather than ideas; a system that implements certain ideas is considered better than the ideas themselves. It is ostensibly about "computers" but its founders, in their effort to differentiate themselves from electrical engineers, were very clear that it was about "computation" which was any activity that could be expressed in the form of rules and algorithms, i.e. in formally specifiable terms (Mahoney 2011). These contradictions however have made computer
science an extraordinarily fertile and porous field, capable of expanding or contracting its boundaries to incorporate new domains and activities for analysis.

While academic computer science dealt comfortably in abstractions (while also building systems), programmers in the workplace struggled to establish themselves as a profession. Jurisdictional contests dogged the software profession from its very beginning: in particular the conflict between programmers and mid-level managers over the role of computing in the workplace. Programmers, in their various incarnations as coders, systems analysts, and consultants, tried to assume—using as their bargaining chip, their technical skills and the increasing role of data processing in the day-to-day working of corporations—the jobs that mid-level managers did (Haigh 2001; Ensmenger 2010). Managers succeeded in this fight by virtue of being able to label programmers as mere technicians.

The status of both academic computer science and software engineers has arguably changed with the rise of the Internet and the ascension of Silicon Valley as the nation’s greatest site of innovation. Software engineers and computer scientists have been successful at building software infrastructures that are now used ubiquitously, in workplaces and at home. This, in turn, has empowered new actors—“geeks” (Kelty 2008), “hackers” (Coleman 2012), and Silicon Valley entrepreneurs (Streeter 2010; Turner 2006)—to construct technical practices (e.g. open-source sharing, open licenses) and ethical representations (e.g. the theory of “open-source governance”) that re-orient existing configurations of power and knowledge (Kelty 2008). These actors have turned their attention and technical expertise to activities usually imagined as strictly “social” by designing software infrastructures for brokering services (e.g. AirBnB, Uber), the distribution of cultural products (e.g. Netflix, Hulu), and new forms of paid labor (e.g. Amazon Mechanical Turk); computer scientists, in addition, seek to become social science knowledge producers and have called in recent years for a new kind of “computational social science” that draws on “big data” (e.g. Lazer et al. 2009; Lohr 2013).
This chapter has demonstrated that the question of who builds new infrastructures—and by implication, who has technical and institutional access to them—is crucial to understanding how computer scientists (and other experts) intervene in new domains. In the first instance, elite computer scientists at Stanford and MIT created MOOC infrastructures: a platform for formal classes that resembled in look, feel and working, the many sites on the Internet dedicated to informal learning. Early Coursera classes drew heavily on Stanford faculty; Scott Klemmer was recruited early on to teach his class on human-computer interaction. And it was to Scott Klemmer (and not say, David Boud, the learning scientist whose book Klemmer read in 2009 that inspired him to try out self-assessment in his Stanford classroom) that Coursera turned to when they became interested in the question of how open-ended assignments would be assessed and graded. Klemmer and his student, Chinmay Kulkarni, were able to conduct early experiments (sometimes through sheer ingenuity) to both validate the efficacy of peer assessment as well as how peer assessment might be improved in a massive online context. Designing rapidly, at start-up pace, drawing on the literature of the learning sciences (and social psychology and management theory), and using their own class as an experiment, and themselves as users, they were able to help Coursera architect a key component that helped sustain its online empire. Juho Kim’s story is similar. Interested in transforming how-to videos into robust learning resources, MOOCs were an ideal vehicle for his research. His internship at edX in the summer of 2013 was as much a result of his proximity to edX as it was of his considerable talents; it is inconceivable to me that edX, an organization that is mostly software engineers, could have employed a learning scientist intern. Working at edX allowed Juho to build his LectureScape browser and solidified his claim that lecture videos and how-to videos were all on the same continuum of educational videos. If the integration of LectureScape into the edX platform is still far from happening, that is more a consequence of the fact that the edX platform has matured, and modifying its
core features requires far more time, effort, and expense. On the other hand, edX and Coursera employees are regularly found at the Learning at Scale conference, and ideas demonstrated at this conference are far more likely to end up as features in the software of edX and Coursera.

This chapter has also demonstrated that even as computer scientists are entering new domains, they remain tied to the idea that they are doing computer science rather than a different discipline, expressing their findings in terms of abstract concepts like peer interaction, video analytics or even, learning at scale. These abstractions perform a powerful function, linking two different domains together: on the one side, the elements of software architecture and on the other, substantive topics like brokering, cultural distribution, or education. The links to the former are tight; the links to the latter are scruffier. The scruffier links allow computer scientists to intervene in more domains; the tight links help them connect back to software architects who design computing infrastructures (through institutional channels like conferences, internships, and perhaps even consulting). The result might be a leveling of social categories—like “learning,” “brokering” etc.—through the fact that these activities are carried out in infrastructures that are designed to look very similar from a system architecture perspective.

The question of abstractions ties into another debate within STS studies of expertise. Eyal (2013) critiques Abbott’s (1988) notion that the level of abstraction of professional knowledge is key to whether a profession is able to securely control its jurisdiction; a profession whose knowledge is too abstract fails to anything useful; a profession whose knowledge is too tied to particular domains risks losing its jurisdiction. In response, Eyal argues that the notion of an expert knowledge that needs to be at just the right level of abstraction, neither too abstract nor too concrete, is insightful but it only shifts the question a level down. For an analyst, it is not an easy task for an analyst to figure out the “optimality”
Chapter 5: The Tool-Builders: Computer Scientists, MOOC Infrastructures, and the Emergence of a New Expertise

of the abstract knowledge deployed by an expert community until after the jurisdictional negotiation is (mostly) complete. Eyal suggests instead that we “replace “abstraction” with the notion of “immutable and combinable mobile” and investigate the chain of transcriptions by which an expert statement or performance is conveyed along the network toward its “centers of calculation” (2013, 874). This chapter demonstrates the validity of this analysis: the abstractions produced by computer scientists are abstract enough to apply to a wide range of infrastructures (for learning, for brokerage), yet concrete enough that the engineers building these infrastructures can translate these abstractions into decisions for system design; there is a revolving door between the computer science abstractions and the centers of calculation where these infrastructures are built that is traversed frequently, through internships, employment, conferences, and consulting. What it also shows is that this link—“the chain of transcriptions”—is by no means stable and finalized at this moment in time.

While computer scientists are able to recruit domain experts into their own work by building domain findings into infrastructures, it is by no means clear why domain experts could not do these themselves. And indeed, in recent years, we see economists explicitly venturing into the computational design of markets. Social psychologists and behavioral economists have had a great deal of success getting policy-makers to pay attention their findings. My analysis in this chapter suggests that while computer scientists have a head-start in terms of having Silicon Valley’s attention and the ability to actually build and code infrastructures, they will face competition from the domain experts they rely on, especially behavioral economists and social psychologists who may develop both: the ability to speak directly to Silicon Valley engineers as well as computational skills to actually build new kinds of infrastructures.

To summarize, this chapter has documented the strategies through which computer scientists intervene in a new domain (here, the study of learning). They emphasize building software infrastructures that work, drawing on rapid cycles of iterative software development. They
also turn existing actors and incumbent experts within a domain into *users*, who supervise the software, or as *domain experts*, to be deferred to, and whose findings are encoded into the infrastructures that these computer scientists build. Superficially, this resembles one of Abbott’s jurisdic- tional settlements: one profession takes over the task of producing abstractions, the other the task of dealing with clients. But looked at closely, this is not the case. The deference that computer scientists extend towards domain experts does not extend all the way; it is computer scientists themselves who decide what domain knowledge to encode and what particular tools to build. These interventions shift the meanings of expertise within a domain as well as expand the computer scientists' own jurisdiction and increase their social authority, making them experts on multiple jurisdictions and domains. Their modesty about being mere system-builders is thus both true and profoundly transformative for the social domains they intervene in.
6 CONCLUSION

In January 2015, Harvard educational researcher Justin Reich, co-author of two full papers at L@S 2015 (see Mullaney and Reich 2015; Lamb et al. 2015), but also someone who identifies more with the education research community (rather than computer science), wrote an article for *Science* that took stock of research done around MOOCs. His article argues that rather than merely analyzing the exhaust pipe of data generated by MOOCs, newer studies would have to go beyond mere data analysis to actually find out under what conditions learners learned best; the way to do this was to have courses with sophisticated assessments\(^{50}\) that drew on education research, and to conduct causal experiments to figure out which

---

\(^{50}\) By "assessments," he did not mean standards-based summative testing. In general, among MOOC researchers, whether from the learning sciences or computer science, testing is held in low regard.
teaching materials delivered best course outcomes. Yet, he cautioned that these experiments had to go beyond ""domain-independent" plug-in experiments" by which he meant experiments that could be deployed in any course independent of content, for example, those that dealt with meta-cognitive aspects (such as learner motivation or incentives). While these experiments, he averred, had the advantage that "successful interventions [...] can be adapted to diverse settings," however, "[t]his universality is also a limitation: These studies cannot advance the science of disciplinary learning." To address this, he concludes there must be more experiments that look at the impact of specific learning materials and strategies, yet he worries that these are not happening because they require "more intentional nurturing" whereas "[p]lug-in experiments fit more easily in the siloed structures of academia." Reich's argument mirrors the argument made in Chapter 5; he argues that computer scientists and other MOOC researchers are concentrating on “baking pedagogy into software” but not engaging with disciplinary learning (i.e. with physics or chemistry learning in particular).\(^5\)

A few days after this paper was published, on January 20 2015, Reich came to Stanford to give a talk. Reich's talk, hosted by Stanford's newly formed Vice Provost of Online Learning\(^5\) (in charge of Stanford's MOOC program) at the newly redesigned Barnum Digital Learning Center.

\(^5\) In Chapter 5, I argue that disciplinary-specific learning research requires collaboration with instructors and curriculum-experts which does not fit into the working practices of most computer scientists. It also requires making technology secondary to learning, turning computer science into just a "technical" discipline, a vision that leading computer scientists do not buy into.

\(^5\) In 2015, VPOL was renamed to Vice Provost of Teaching and Learning (VPTL).
Learning Hub, was packed with people; the audience included many researchers at Stanford studying teaching and learning, both from computer science and the School of Education. The talk itself was largely a summary of his *Science* paper; it was time, he said, that MOOC researchers studied not just learner persistence or engagement but *learning itself*.

When the floor was opened to questions, the first question came from Chinmay Kulkarni, designer of PeerStudio, and a computer scientist rather than an educational researcher. He asked what he self-deprecatingly called his “stupid question of the day.”

*I guess this is my stupid question of the day. We are assuming that everybody in these classes wants to learn in ways which are very similar to ways the university student learns. Do we know what these guys want? For instance, when I looked at forums on classes, they’re not saying: my achievement levels are up. Or they’re not saying: I learnt this great thing about social psychology today. It’s: this class told me something today that I saw in real life! And that’s really fascinating! Right, so, do we need to change our models of evaluations such that we support what students actually come to courses for, which may be different? [my emphasis]*

The phrase “stupid question of the day” should alert us to the fact that the question poses fundamental issues in the study of learning. What Chinmay was questioning was what counts as a valid educational goal, and who gets to decide this. Typically, experts on learning have decided what counts as an educational goal and what counts as a good assessment to evaluate those educational goals. MOOCs had changed this picture for many of my interlocutors, changing the definition of what “course” and “learning” meant. MOOC courses have ranged from introductory classes in computer science and physics to classes meant to raise awareness of poetry and global architecture to more interventionist classes on public health and women’s welfare to classes on Mongodb and Microsoft Excel to professional education classes on “data science” to help corporate employees pursue professional advancement in Silicon Valley. These courses, with their blend of formal and informal matters of education, have arguably shifted what a “course” means and thereby what a “learning objective” for a course can be. Reich replied to Chinmay’s question by answering that it was best to leave the
definition of a course objective to particular instructors themselves. But this answer itself is an indication of a shift: that instructors, and not say curriculum or learning experts, get to decide what learning objectives are, and that these learning objectives had to go beyond a certain understanding of learning per se, that it had to consider learners who did not necessarily share the learning objectives that an instructor of a course had in mind.

In my two years of fieldwork around MOOC system builders, instructors and researchers, I observed these sorts of conversations numerous times: debates about what the objectives of learners are, about who gets to set the learning goals, and how best to measure and understand them, if they are not set by incumbent experts like learning scientists. Rather than seeing these conversations as simply a response to demographic and technological changes (as my interlocutors often did), this dissertation has shown that they are the result of adopting a new model of socio-technical organization, the software-as-platform, into an existing institutional arena of teaching and learning; the result is, not unnaturally, an ongoing "boundary work" negotiation between expert communities. These questions are truly fundamental: is the learner best considered a user of software or a student of a higher education institution? Are instructors to be understood as co-designers or just another class of users of these software systems? Finally, is a learning infrastructure best seen as an example of an educational institution or a "platform" like YouTube and Amazon with architects and a core and periphery of complementors and users? The settlement of these questions about the nature of learning goals and who sets them, I argue, also determines how learners are studied, who gets to study them, and how learning infrastructures are designed. The debates I describe are happening within the world of MOOCs but given the players involved in this project—venture capital, elite universities, and software experts—these debates have the potential to be important for the future of higher education.
Chapter 6: Conclusion

What role does the software-as-platform play in these debates? I argue that it functions as a dispositif, a term used by Michel Foucault (1980, and rendered by his translators as "apparatus") to mean a "thoroughly heterogeneous ensemble consisting of discourses, institutions, architectural forms, regulatory decisions, laws, administrative measures scientific statements, philosophical, moral, and philanthropic propositions" (194). Here is Foucault:

> What I'm trying to pick out with this term is, firstly, a thoroughly heterogeneous ensemble consisting of discourses, institutions, architectural forms, regulatory decisions, laws, administrative measures, scientific statements, philosophical, moral and philanthropic propositions—in short, the said as much as the unsaid. Such are the elements of the apparatus. The apparatus itself is the system of relations that can be established between these elements. Secondly, what I am trying to identify in this apparatus is precisely the nature of the connection that can exist between these heterogeneous elements. Thus, a particular discourse can figure at one time as the programme of an institution, and at another it can function as a means of justifying or masking a practice which itself remains silent, or as a secondary re-interpretations of this practice, opening out for ait a new field of rationality. In short, between these elements, whether discursive or non-discursive, there is a sort of interplay of shifts of position and modifications of function which can also vary very widely. Thirdly, I understand by the term 'apparatus' a sort of—shall we say—formation which has as its major function at a given historical moment that of responding to an urgent need. The apparatus thus has a dominant strategic function. This may have been, for example, the assimilation of a floating population found to be burdensome for an essentially mercantilist economy: there was a strategic imperative acting here as the matrix for an apparatus which gradually undertook the control or subjection of madness, mental illness and neurosis (p194-5).

The software-as-platform, this dissertation has argued, is exactly such an apparatus. Its emergence can be traced to the practices of the software industry, and in particular, to the

---

53 Dreyfus and Rabinow (1983, 120-21) however prefer to interpret dispositif as "grid of intelligibility" rather than as just "apparatus."
discourses of Silicon Valley as the premier centre of the software industry. Its introduction into the world of education too comes from “responding to an urgent need.” American higher education is a lumpy entity consisting of elite research universities, teaching schools, liberal arts colleges, community colleges, professional and extension education and for-profit schools (Stevens and Kirst 2015). U.S. higher education is now widely seen by various actors as being in crisis: from matters as diverse as the high price of college (Archibald and Feldman 2010), cutbacks state funding despite high demand (Newfield 2011) and the inability of universities to provide for working adults who attend school part-time, now the majority of higher education students (Deil-Amen 2015). Partly because of their flexibility suits non-traditional students, for-profit schools have seen an increase in enrolment; yet their high cost and poor job placement rates are largely responsible for increasing student debt (Mettler 2014). Finally, there is the increasingly publicized finding that higher education students do not learn (Arum and Roksa 2011)—a sign perhaps that the accountability revolution may have reached higher education.

In public discourse on MOOCs and American higher education, this story about American higher education—a political economic and demographic one, it should be noted—is routinely told. In two recent books on higher education and MOOCs (Carey 2015; Selingo 2015) by journalists who cover education, MOOCs arose organically from this state of affairs, and may have the seeds of a solution. Carey, who is more hyperbolic, suggests that the MOOC is the first step along creating the University of Everywhere, a university where instruction is available cheaply to whoever who wants it, in whatever fashion he wants it (slow-paced, fast-paced etc.), for whatever purpose he wants it (for professional development, out of interest, to learn a skill). Both these authors suggest that American higher education, with its high costs, low learning, and its emphasis on elite schools, is not serving the student who most needs it: the part-timer, the adult, the working professional.
The students who take MOOCs in the United States, they argue, come from these same publics. Of course, this has been amply demonstrated in the research studies of MOOC learners (Ho et al. 2015) and it comes through in many of the revenue-focused initiatives put out by edX, Coursera and Udacity in the United States: most of these initiatives focus on professional education and new kinds of skills that corporations are in need of like Python programming and data science.

However, there is another story about MOOCs and MOOC infrastructures that is often talked about, especially in the locales explored in this dissertation: the sites where software, courseware and knowledge are produced. This is a story about technological change, about MOOC infrastructures serving as hubs of “innovation” within the world of teaching and learning. The simplest—crudest—version of the story of technological change went something like this. The following is an extended quote from an engineer from Google speaking at an event organized by LearnLaunch, an educational technology start-up incubator in Boston:

*Education is rapidly changing right now. And what's really going on is that technology, I don't know, in the 90s, was really driven by enterprise software. Right? All the innovations were around how to make the enterprise more effective. And then suddenly what happened is there was an explosion in the consumer market. And innovation became even more rapid. And then, at that point, the enterprise couldn't keep up with it because there was so much investment in the existing infrastructure—in the existing hardware, and in the*
existing software solutions—and so what we’ve seen over the last 10 or 15 years with the growth of the Internet and the explosion of consumer use is a huge amount of innovation. And now what we’re trying to do is fold that back into the enterprise. And schools are just another enterprise—with some additional constraints. [...] So the reason there’s so much opportunity in the ed tech community is because there is now the ability to rapidly adopt emerging software quickly in the classroom, that has never been the case before. And so effectively, all the benefit we’ve got from the advancement for consumers is now possible in the classroom. Once that innovation loop starts, it will move very very quickly and we just don’t know quite when it will explode, but we’re starting to see results quickly. [Emphasis mine.]54

This is clearly a self-serving story for the relevance of Google products to educational technology: notice the number of times the word “innovation” makes its appearance! Yet, the story relies on two different motifs that I found throughout my fieldwork: the comparison of the student to the software user that is seen as hugely productive in terms of designing new software for teaching and learning; and the idea of a “loop,” whereby one improvement is succeeded by another improvement, all of them building on each other to create a fast-moving system of teaching and learning that gets better and better.

Rob Miller, a professor of HCI at MIT, and Juho Kim’s adviser, who has been interested in building new tools for online learning, “helping crowds teach each other,” as his website puts it, told me a slightly different, more nuanced story:

54 The event was called “Ed-Tech State of the Union” and took place in an office near Kendall Square, Cambridge, MA on May 29, 2014. The Google Engineer who spoke was Steve Vitro who managed many engineering teams related to education at Google’s office in Boston.
During the 90s, the 90s and the early 2000s were kind of the period when bulk of the world’s population had to be brought into computing, it was the period of the age of the Web and subsequently the period of the rise of the smartphones, right? [Me: Right], and hundreds of millions, and then billions, of people, were brought into using computers at that point, right, orders of magnitude increase in population, and during that kind of transition, what actually matters the most is learnability, its building systems that people can learn, and not necessarily efficiency, right, not necessarily productivity. [emphasis mine]55

Miller’s story, despite its reference to the rise of the Web users, is also an internalist story about technology. The rise of the Web and the expansion of computing made the learnability of software all-important, he argues. His suggestion is that the rise of the Internet made the principles of learning more relevant to software design, which, he suggests, was perhaps too occupied with notions of productivity and efficiency. But he is also arguing that the time has now come that the principles of software design can now make their way back into the design of learning infrastructures.

A sophisticated articulation of this story, where political economic events are given a strong technological flavour, can be found in the writings and talks of George Siemens, a figure I analyzed in Section 2.2. Siemens is an interesting figure in the world of MOOCs: while he played no part in the Stanford revolution of 2011, he nonetheless occupies an important position in it today. He works primarily as a synthesizer: someone who has been able to leverage the sudden burst of interest in online learning by skilfully articulating a vision of the

55 Interview with Rob Miller, July 25 2014.
role of online learning in the future; a vision, that I argue, appeals funders, educational researchers, technologists, and computer scientists, alike.

**Assertions:**

1. Knowledge needs are today defined by complexity and interconnectivity
2. Emerging employment opportunities are knowledge-based
3. The idea of a university is expanding (complexifying)
4. Knowledge institutions mirror the architecture of information

*Figure 6.1: A slide from George Siemens’ talk at MIT in July 2013*

In Siemens’ writing and talks, this kind of story about technological-demographic change is most fully articulated. His paper “Connectivism: A Learning Theory for the Digital Age” (Siemens 2005), published in the International Journal of Instructional Technology and Distance Learning, is a good example; according to Google Scholar, the paper has been cited 3607 times as of this writing (more than 2000 of these came after 2011), and made Siemens’

---

56 I saw him talk twice, always in a full room (at MIT as well as the International Conference of the Learning Sciences in 2013 and 2014), and attended three days of workshops at the Learning Analytics and Knowledge Conference in 2014 where he was present and spoke extensively.
name in the world of distance learning. The paper makes an argument that the world has changed, and therefore the three epistemological frameworks for learning—behaviorism, cognitivism, and constructivism—as well as the organizational frameworks of higher education are no longer enough. There is too much knowledge in the world, and no amount of learning in higher education institutions as they currently exist, will equip any learner for organizational life as a knowledge worker. Siemens there proposes something he calls “connectivism,” which is both a theory of learning and a framework for the reorganization of higher education institutions using technology. In his talk at MIT in July 2013, one of his slides carried the following prominent text: “education systems track the architecture of information of an era.” This was intended both as a descriptive and prescriptive statement: something that should happen, but was also already happening. The architecture of information of today’s digital age, according to Siemens, is “open, distributed, scalable, social, generative, networked, self-organized, adaptive, global” [these were bullet points on the slide]; thus, the architecture of higher education institutions need to adapt (see Figure 6.1).

Beyond the recommendation that learners need to be encouraged to talk to each other, and the assertion that the new architecture of knowledge has consequences for the design of learning environments, I have found very little that is concrete in Siemens’ oeuvre. But his extensive writings—on his blogs, in white papers and position papers—and the conferences he often organizes (prominently the Learning Analytics and Knowledge or LAK conference that he started in 2011) as well as his ability to convince funders and foundations, his views are a good reflection of the world-view of MOOC makers and researchers. In this view, information technology and the reorganization of information (primarily through the emergence of the Internet) has transformed the nature of work and the process of learning. In such a world, it is almost the duty of technologists to build (software) tools to support
Chapter 6: Conclusion

learning and to reconfigure institutions—both of which should “mirror” the organization of information aka the Internet.

These stories wound their way into providing elaborate theoretical justifications for technical choices that were often made in the heat of nitty-gritty technical work, of just getting systems to work. When Chinmay Kulkarni was preparing his job talk—an elaborate exercise that lasted weeks—he went through several explanations of why his work on creating peer assessments was important. In his actual job talk, he told a particular story beginning with the fact that MOOC learners were un-conventional; they were often working adults with a degree (55% of them for the HCI class that formed the basis of his work). What this indicates, he says in his talk at the University of Washington, is an “enormous thirst for learning.” How, then, to translate what (clearly elite) universities have been doing so well for their populations of undergraduate and graduate learners to these new populations? Kulkarni outlines three things that are difficult to do, to “scale,” for these new populations: providing useful feedback to learners on open-ended work, structuring interactions between learners who may be completely different from each other (they have “diverse perspectives”), and finally, the ability for learners to be able to revise their work to gain mastery. These three needs—“motivations” for his research—allow Kulkarni to segue into a description of his research: building learning infrastructures at scale that provide these three affordances.

When he concludes his talk, Kulkarni talks about future work that he plans to do.

One way of looking at my work is to take this view of learning as something that transforms novices into experts. And you’ve seen traditionally how it’s done with deliberate practice: students revise, they get feedback, they keep improving, and the traditional way of doing it is with the help of a coach. Over the last decade or two, we’ve seen intelligent tutoring systems, which have enabled more students to do deliberate practice in domains like mathematics. What this talk shows is how we can broaden the domains for deliberate practice to more creative work and more open-ended work.

But all of this is still along one dimension: getting students from being a novice in one domain to being an expert in the same domain. But if you look at some
of the most interesting jobs that we've created over the last 10 years, they're jobs that did not exist in the decade before. So we have this opportunity actually to help students learn across domains and learn all through their life. So what I want to do in future research is to enable people to become adaptive experts who take what they know in one domain and use it to become experts in other domains. [my emphasis]

Here Kulkarni is describing both a theory of the world and a theory of learning. The picture of the world he brings up is one that is moving very very fast. The world requires systems that allow translation and learning of skills across domains. This fast-moving world, when paired with the non-traditional learner with which Kulkarni began the talk, builds up to a picture of the learner who is a different creature today than he was before. The story Kulkarni uses here, like Siemens’, describes the changes over the last 30 years as solely technological. Technology has complexified jobs, and therefore changed what learning means. What they omit are the political economic aspects of the story: the stagflation of the 1970s, the rise in inequality, the shifting of risk onto the individual, preferring instead to concentrate on the “thirst for learning” and the ongoing march of technology. What these stories do then is to link—associatively, if not causally—technological changes with theories of learning.

It is this orientation towards the future, that learning in the future will look different from what it does today, that it will turn into an “ofttouted and idealized ‘hiding-in-plain-sight’
ubiquity, blurring boundaries between formal and informal education,” 57 that lies behind the pedagogy agnosticism of MOOC infrastructure (see Chapter 4). This is not to say that the edX infrastructure is neutral with respect to the structure and type of courses that are produced on and through it. Rather, it means that the infrastructure has been built by its architects to cater to as many possible pedagogical styles and methods as possible, including the ones that don’t yet exist or aren’t recognized as such, the hope being that some of it will inch towards what its architects believe will be the pedagogy of the future in an information-centric world. The contrast with Carnegie Mellon’s decade-older Open Learning Initiative (OLI) is instructive here. OLI was built as a proof of the effectiveness of a certain theory of learning (a cognitive-science driven theory of learning that is at the heart of intelligent tutoring systems). The edX infrastructure has been designed to achieve multiple objectives: make itself self-sustaining (i.e. capable of institutional reproduction); and produce learners, knowledge about learning and innovation—through careful governance—that will move towards what its architects believe is the pedagogy of the future.

It is in pursuit of these goals that the architects of MOOCs have adopted the tenets of the software-as-platform. Some of the constituent normative logics of the software-as-platform fit in perfectly with the normative logics of the world of education. Personalized learning,

57 The phrase occurs in a report issued by the Computing Community Consortium which can be read in its entirety here: http://archive2.cra.org/ccc/files/docs/meetings/OnlineEducation/CCC-MROE-Report.pdf.
using intelligent machines, was a goal of behaviorists like B. F. Skinner as well as cognitivists like Herbert Simon for at least fifty years, if not more (see Watters 2014 for some recent posts in the blogosphere that go into some of this history). Personalization through the software-as-platform however is slightly different in that MOOC architects hope to achieve personalization less by discovering the right theory of learning then by allowing a thousand flowers to bloom and picking those that work best (through a suitably pragmatic criterion of “best”). Data-driven teaching too is not a new concept in the world of education research, whose singular concern has been to bridge the gap between “research” and “practice,” i.e. bring education research into teacher training (Lagemann 2002). Data-driven teaching through the software-as-platform takes two forms: building new tools for teachers (dashboards, A/B testing), and nudging them to use those tools. Again, the point is to not emphasize any particular theory of how learning works but to instead grope towards a a set of solutions that “work” by trying out many different solutions. Those working with MOOC infrastructures—whether from traditional education research or the new computer scientists—take the elements of the software-as-platform to reinvent what they see as traditional (data-driven teaching, personalized learning, learner-centered pedagogy) or non-traditional goals (just-in-time learning, mobile learning).

It is in this respect that MOOC infrastructures are different from previously existing ones for online learning. It is not that online learning did not exist before. It has arguably existed for at least two decades in most public universities, community colleges and for-profit colleges; yet, at least in typical colleges and universities, the position it occupied was a peripheral one: in extension schools, for professional education and reserved for the non-traditional student. This world of online learning typically consisted of a delivery system, known as a Learning Management System (LMS), LMS vendors (like Blackboard and Moodle), LMS specialists, educational technology experts, and instructional designers with varying amounts of
traditional faculty involvement (sometimes tenured faculty, at other times adjuncts). The workforce for this effort was thus distinct from the typical academic workforce (of discipline-based PhDs, for instance); many were (and are) trained in Education schools with Masters degrees (Reiser and Dempsey 2011).

The world of MOOCs looks considerably different across multiple dimensions. Where distance education departments once purchased software tools such as LMSes to deliver their product, the MOOC providers are themselves software companies that primarily employ software engineers (there are no content experts, for example, at edX); their product is a software-driven platform that universities and professors use to create, build, and execute courses. Engineers at edX and Coursera are typically not graduates of education schools, nor do they have any particular expertise in teaching and learning; their prior educational experience is likely to be an engineering degree, and their experience often comes from working at companies like Netflix, Amazon and Google. They are kept insulated from university instructors—who, at this juncture at least, seem to be mostly tenure-track faculty—through an organizational layer of relationship managers. The organizational imperative is to make university instructors as self-sufficient as possible: the primary way of doing this is by building new kinds of authoring software and tools like dashboards that assist instructional staff. The student data generated by MOOCs circulates among the MOOC provider, the university MOOC organization, and university instructors and researchers—who are more likely to be computer scientists than from the traditional learning sciences or education research. There is a great deal of talk about new institutional models for collaboration between instructors and researchers to make experimentation (or “A/B testing” in software parlance) easier (e.g. see Reich 2015); A/B tests are exalted because they can produce casual claims about how teaching practices relate to student learning.
This dissertation contends that this world of MOOCs is being brought into being by actors (primarily academic computer scientists) who are both intimately familiar with and self-consciously inspired by the work practices and accomplishments of Internet platforms like Amazon and Google and the rhetoric of Silicon Valley. It documents the translation of these platform techniques into teaching and learning, and how these are being used to reconfigure educational institutions i.e the process of platformization. I show that the significance of the MOOC world to higher education lies in the process of translation of the normative logics of the software-as-platform (which favor certain kinds of expertise over others) into the world of higher education, perhaps more so than it does in the high-flying techno-utopianism that sometimes swirls around MOOCs.

If MOOC architects are viewed as education reformers, how are they different from the countless reform movements in American education? Tyack and Cuban (1997) have shown that the progressive movement for change in public schools in the first half of the twentieth century was driven mostly by school administrators who had graduated from recently-established schools of education; interestingly insofar as this movement was progressive, it was about administration (about tracking, social efficiency, etc.) rather than learner-centered pedagogy (Labaree 2006). Reform movements in the second half of the twentieth century sprang from different sources: disciplinary scholars made a move to reassert their hold on the setting of the school curriculum displacing educationists. These scholars were motivated by various ideologies: the physicists wished to assert the priority of basic science rather than its technological applications (Rudolph 2002), the mathematicians wanted to show that mathematics was about reasoning rather than calculation (Phillips 2014). Into these concerns, one must add the fear that the US was falling behind the Soviet Union in “scientific manpower” (Kaiser 2002). Another strand arose from the emerging identity-based civil rights movements in the 1960s, which made schools and colleges the focus of their attention;
education came to be seen as a crucible through which long-established inequities in American life (race and gender) could be addressed and resolved. Successive American governments have made education a funding priority in different ways (even if not always successfully), while “diversity” became a way for both employers and higher education institutions to incorporate the demands of the civil-rights movements into their own organizational frameworks (Loss 2014).

The MOOC movement is a consequence of the rise of new actors—“geeks” (Kelty 2008), “hackers” (Coleman 2012), and Silicon Valley entrepreneurs (Streeter 2010; Turner 2006)—who have been successful in constructing technical practices (e.g. open-source sharing, open licenses) and ethical representations (e.g. the theory of “open-source governance”) that reorient existing configurations of power and knowledge. The academic-military-industrial complex has been partly replaced by academic capitalism (Mirowski 2011; Slaughter and Rhoades 2009): a world of technology-transfer offices, licensing of university-produced research to corporations, and free movement of money and people between the academy and start-ups, particularly in computer science and biotechnology. Finally, computer scientists have turned their attention and technical expertise to activities usually imagined as strictly “social”; these researchers have called for a new kind of “computational social science” (Lazer et al. 2009). The central concept of these new actors—the platform—is now seen as applicable to a wide variety of tasks: hailing cabs (Uber), hiring workers (TaskRabbit), and last, but not the least, building learning infrastructures.

While the MOOC movement draws from many of these previous strands in American education reform movements (learner-centred pedagogy being one), its governing ideology is neither one of social efficiency nor the Deweyan image of a school as a laboratory for life. Rather, MOOC architects seem animated by what Thomas Malaby (2013, 295) has called “technoliberalism”: “[an] intense suspicion of vertical authority, a commitment to making
technology universally accessible and beyond institutional control and a deep faith in the positive aggregate effects that follow from individual use of this technology for the purposes of creative expression.” Technoliberalism has rooted itself in Silicon Valley, through a curious linking of the New Age ideology of personal liberation to the project of technological development, in particular of computing (Kelty 2014; Turner 2006). MOOC platform builders are by no means free market ideologues but they do not seem to have faith in bureaucracies or institutions. Rather, they see individuals (learners, teachers, researchers) as holding a latent set of creative faculties, which can be brought out through carefully structured—engineered—technologies that these individuals are given access to. This dissertation illustrates how technoliberalism is not just some free-floating ideology, but also guides actual practices of system development within the MOOC world through the process that I call “platformization.”

Where might all this lead? As of now, it seems hard to say for higher education in general. But some consequences seem clear, for example, in the world of education research. Chapter 5 documented how research on education itself is being transformed through the entry of computer scientists, the “tool-builders” inspired by Internet platforms, who seek to build tools for teachers and learners. Ellen Lagemann (2002) has argued that the history of education research shows certain fault-lines. In the early 20th century when this research emerged, the faultline was epistemological: how was learning in schools best studied? Psychologists won this debate decisively: it was to be studied quantitatively, in laboratories and by experts alone (rather than with teachers). In the second half of the 20th century, behaviorist psychology was replaced by cognitivism and cognitive science (Gardner 1987) which construed behavior not as a sequence of stimulus-responses but as something driven by mental representations that changed with time in response to environmental factors. Out of cognitive science emerged the learning sciences where theories of “situated learning” (Lave and Wenger 1991)
have been encouraged along with ideas of “design-based research” (Sawyer 2006); this field has benefited from vast amounts of NSF funding. The victory of certain actors in the debates of education research was aided by ongoing developments. The emergence of behaviorist psychology at the beginning of the 20th century and its eventual victory as the method to understand the educational process (Lagemann 2002) fitted the emerging ecology of education: the low-status of (female) teachers, the professionalization of research within the academy to be conducted by high-status researchers, and the emergence of academic disciplines. Cognitive science was successful partly because it appealed to the ideology of the “open mind,” seen as a bulwark against the closed mind of the Stalinist Soviet Union (Cohen-Cole 2016).

The move to make data widely available to various actors (through carefully constructed interfaces) as well as the development of A/B testing tools to encourage experimentation in the world of MOOCs portends a change (if not a disruption) for both the methods and the product of educational research. The MOOC software-as-platform is a massive, standardized, experimental apparatus. Unlike traditional learning science researchers, the new actors who seek to do research through MOOCs see less value in psychological or cognitive models of learning; they have no objection to theories but they are far more interested in looking for interesting and useful patterns that can be used to implement solutions to outstanding problems in pedagogy. The MOOC platform therefore is a potential game-changer with the possibility of a cultural shift in the study of learning—from a science-based model of learning occupied with research questions to an engineering-based model occupied with solving problems.

Only time will tell what the impact of MOOCs and MOOC infrastructures will be on the American higher education sector. What is clear is that some university administrations (like those at Stanford, Harvard, and MIT) as well as philanthropic foundations in the US have
increasingly turned to thinking about MOOC infrastructures to address crises in US higher education, such as decreases in state funding for public institutions, rising costs, lack of student learning, and the inability of universities to accommodate the part-time learner. MOOC architects, inspired by the software-as-platform, and drawing on the precedents and work practices of Silicon Valley, increasingly frame MOOC infrastructures as sites where teaching is “data-driven,” A/B testing is fast and easy and available to all, the process of learning is “personalized” by humans and programs acting in concert, and a generative process of innovation can happen to transform teaching and learning. This framing allows them to situate the ideal higher education institution as an exemplary Internet platform, and thereby bring the norms of Silicon Valley into higher education, as well as increase their own social authority as institution-builders and knowledge producers. If anything, it is this framing of the university as a “platform” that might be the biggest consequence of MOOCs.
https://books.google.com/books?hl=en&lr=&id=E2BdY6WQo4AC&oi=fnd&pg=PA1&dq=janet+abbate&ots=TBevTPfJm&sig=lczygAyMnIhZxCpQN017STc_p9A.


References


References


References


References


References


References


