The Confederacy of Software Production: Field Experimental Evidence on Heterogeneous Developers, Tastes for Institutions and Effort

(Preliminary Draft for NBER 50th Anniversary Conference on the Rate and Direction of Inventive Activity, September 30 - October 2, 2010)

Kevin J. Boudreau (London Business School) &
Karim R. Lakhani (Harvard Business School)

This version: September 3, 2010

Abstract

At least as much as other innovative and creative problem-solving sectors of the economy, software development takes place in an extraordinary range of different sorts of organizations thus forming a patchwork or "confederacy" of institutional forms. We suggest one explanation that may begin to account for this heterogeneity: workers have different intrinsic tastes or preferences for different institutional regimes. We present field experimental evidence to show that software workers who are sorted into either a cooperative or competitive regime, depending on their tastes, exerted on the order of double the effort than those in a group of randomly-assigned workers (controlling for their skills). These results suggest that sorting on the basis of institutional tastes and preferences to different types of organizations may have large efficiency implications.

Acknowledgements: We would like to thank Eric Lonstein for his assistance throughout this research project. We are grateful to Jeff Davis, Elizabeth Richard, Jennifer Fogarty and Bara Reyna from NASA's Space Life Sciences Division in helping us identify an appropriate computational engineering challenge. We sincerely appreciate the tremendous cooperation extended to us by the TopCoder team including Jack Hughes, Rob Hughes, Mike Lydon, Ira Heffan, Jessie Ford and Lars Backstrom. The M-Lab at the London Business School and HBS DRFD provided financial support for this project. All errors remain our own.

1 Introduction

Ubiquitous yet invisible, software plays an integral role in the global economy. It is essential for the effective functioning of most modern organizations, critical to the advancement of knowledge in many fields, and often indispensible to many individuals' daily activities. Concomitant with widespread use, the economic footprint of software is quite large. In 2007, the more than 110,000 firms engaged in the production and sale of packaged and custom-developed software, and related IT services generated in excess of $300 billion dollars in direct revenue (National Science Foundation 2010), making this one of the largest US industries. Purchased software is complemented by software created within user organizations (Mowery 1996) and open source software initiatives (Lerner & Tirole 2002). The extent of internal software production and investment is considerable, most firms typically spending 50% more for new, internally developed software than for software obtained through external vendors (Steinmueller 1996). In the United States alone, more than three million individuals work as software developers (King et al. 2010), the majority employed by establishments that sell neither software nor software related services (Steinmueller 1996).[1]

Among activities in the modern economy that require significant knowledge generation and use, discovery, problem-solving, and innovation, software development is distinguished by the sheer number and heterogeneity of participating organizations. Software is developed in such diverse settings as small entrepreneurial firms, departments in large multi-national organizations, universities, outsourcing consultancies, collaborative endeavors like open source software communities, and the proverbial "garage." This organizational heterogeneity has also given rise to diverse approaches to,

[1] The worldwide software market and employment levels are also significant. DataMonitor, a professional market research firm, estimates 2009 revenues of software and related-services firms to be $2.3 trillion (DataMonitor Report 0199-2139) and IDC projects the direct software developer population to exceed 17 million individuals by 2011 (IDC Report 1517514).

and perspectives on, developing software including software factories in Japan, the scientific method reflected in the practices of European electronics and technology champions, the ordered, engineering orientation pioneered by the US military and Software Engineering Institute, and the "slightly" out of control bootstrapped development methodology practiced by Silicon Valley firms (Cusumano 2004).

Even development of similar software functionality is often achieved in quite different ways not only across similar firms, but even within individual organizations and from project to project (Cusumano et al. 2003). Thus, the same firm may pursue a "waterfall" development process that utilizes military-like hierarchical command and control structures, employ small feature-teams working on delineated functions, and utilize paired and agile programming arrangements and even internal and external tournaments in developing software. Firms also continually change and tinker with their development practices in search of the "silver bullet" (Brooks 1995) to the challenge of software development (Microsoft's various changes in development process are well chronicled by (Cusumano 1991a), (Cusumano & Selby 1995), (Cusumano & Yoffie 1998), and (Sinofsky & Iansiti 2010)).

There may, of course, be any number of reasons for the wide range and diversity of modes of organizing software development and developers. Here, we explore the possibility that the extraordinary variety of organizations might simply reflect the equally large variety of workers in the industry. Since the emergence of software development as a practice and profession in the 1950s, study upon study has remarked the heterogeneity of motivations reported by software developers and characteristics that define this distinct occupation (Weinberg 1971; Beecham et al. 2008; Schneiderman 1980; Couger & Zawacki 1980). In particular, the literature has reported that programmers express strong, conflicting institutional preferences for creating software autonomously versus interdependently, i.e. working as part of a team. If these different behavioral orientations of workers were associated with distinct tastes for the sort of organization in which they prefer to work, there may plausibly be systematic sorting of workers to different sorts of organizations and productivity implications. Thus, the motivation for this work is the idea that the large, wide, and varied confederacy of software production is at least partially accounted for by the preferences of workers in the industry.

This chapter presents the results of a field experiment, intended to begin to explore the efficiency implications of possible sorting of workers in the industry. Our central goal is simply to measure whether workers assigned the institutional regime of their preference chose to work harder than those in a randomly assigned group. The randomly-assigned group would constitute the population average mixture of those that preferred the regime and those that did not. The experiment involved solving a challenging computational-engineering problem posed by NASA's Space Life Sciences Directorate, related to the space station program. The over 1000 workers in the experiment were assigned to groups of twenty which were either team "cooperative" or autonomous "competitive" regimes. In the competitive regime, individuals competed against all others in the room; in the cooperative regime, individuals were assigned to one of four teams of five workers. These two regimes hardly replicate the full variety of regimes we observe in the confederacy of software organizations. However, they do exhibit a range of starkly opposing features that accord with important distinctions in organizations and the preferences exhibited by software developers. So, we expected the sharp choice between the two regimes would be more than sufficient to exhibit effects of selection. Half of the participants were asked which regime the would prefer and were then allocated to that regime. Thus they virtually self-selected themselves to their preferred regime. The other half were randomly assigned. We used a combination of matching and randomization to match self-selection groups on skills and unobservables. We were also able to compare the effects of self-selection to the effect of formal incentives, as some groups of 20 competed for $1000 in prizes; for others there was no prize.

We found that allocating individuals to their preferred regimes had a significant impact on choice of effort level, particularly in the autonomous competitive regime, in which self-selected participants worked, on average, 14.92 hours compared to 6.60 average hours for the randomly assigned participants. The effect was also positive and significant in the team regime, in which self-selected participants worked, on average, 11.57 hours compared to 8.97 average hours for the randomly assigned participants. We found that the effect in the case of the team regime was not just the result of an initial boost of effort from self-selection to one's preferred regime, but also an amplification of this effect through social interaction, with individuals exerting higher levels of effort in the presence of teammates exerting higher levels of effort. We found no evidence of Hawthorne effects (in the sense that being given a choice per se affected choice of effort level). Re-weighting the data in various ways to facilitate comparisons between the two regimes and synthesize alternative control groups did not qualitatively change the magnitudes of estimates.

The rest of the chapter is organized as follows. In Section 2, we review software development as a practice and industry over the past 50 years, argue that it should be understood as a confederacy of organizations and workers of many stripes, and explore the possibility that the wide variety of organizational modes might exist to harness heterogeneous motivations. Section 3 describes the field experiment. We elaborate the approach at some length, given the special concerns introduced by a unique field experiment on innovation, particularly one in which selection is treatment. In Section 4 we present our sample and variables. We present our results in Section 5, and conclude in Section 6.

2 The Confederacy of Software Organizations and Developers

2.1 Software and "Development" Work

Software is the basic "information" component in information technology (IT), specifically, the sets of instructions that tell computer hardware what functions to execute and perform according to predetermined operations or in response to user or environmental input. Computers and related devices cannot work without software of one type or another. Music provides a useful analogy. Just as an instrument needs a score to produce meaningful music, so computers need software to perform productive work.

To program software code requires of developers the kind of creativity and invention common to scientific work achieved through effective problem solving and trial-and-error learning (Cusumano 2004). Development is a highly complex cognitive task that requires programmers to simultaneously understand an abstract problem and design and develop, and verify the viability of, a concrete solution (Hohmann 1997). Developers hence need to be able to rapidly understand idiosyncratic user needs in specific application domains and articulate the specific problems and sub-problems to be tackled by the software. Effective developers must also, of course, possess software programming expertise (i.e., knowledge of the science of logic and rules of programming languages) and be able to define programming plans, determine interaction profiles, and design and deliver solutions that address user concerns, all under the shadow of varied, continuously changing requirements (Hohmann 1997). Often, users do not really know what they want until a programmer has created a prototype system. Even as user requirements are changing, so is the computation environment in which the programming is to be implemented. Developers must be continually aware that hardware and operating system software on which they create are also themselves not static, and as likely as not to change even as programs are being written. Finally, code must be "perfect" in both logic and syntax, as the slightest error in either can prevent a program from executing and serving the function for which it was created (Cusumano 1991).

Brooks (1995 pg 4), reflecting on its high failure rate, difficulty, and uncertainty, termed software development a "tar pit."

> "No scene from pre-history is quite so vivid as that of the mortal struggles of great beasts in the tar pits. In the mind's eye one sees dinosaurs, mammoths, and saber-toothed tigers struggling against the grip of tar. The fiercer the struggle, the more entangling the tar, and no beast is so strong or so skillful but that he ultimately sinks.
>
> Large system programming has over the past decade been such a tar pit, and many great and powerful beasts have thrashed violently in it. Most have emerged with running systems – few have met goals, schedules, and budgets. Large and small,

> massive or wiry, team after team has become entangled in the tar. No one thing seems to cause the difficulty – any particular paw can be pulled away. But the accumulation of simultaneous and interacting factors brings slower and slower motion. Everyone seems to have been surprised by the stickiness of the problem, and it is hard to discern the nature of it."

The extreme uncertainty that underlies software development is well documented with a majority of projects failing to meet functionality, quality, schedule, or cost goals, and many canceled outright upon completion (Cusumano 1991; Cusumano 2004). Perhaps even more striking is the observation made in the 1970s, based on IBM's experience developing the more than one million lines of software code for its OS/360 project (which was late by more than a year, rife with bugs, and the half-billion dollar cost of which exceeded budget by a factor of four (Cusumano 2004)): adding human resources to an already late software development effort actually further delays its completion (Brooks 1975). Software thus represents an intellectual and management challenge to both the organizations that develop it and the workers who participate in its creation.

2.2 The Industry's History and Organizations: Emergence of a Confederacy

Subsequent to its debut in the 1950s, computing's software component has evolved from supporting specialized military, government and business applications to the status of a general purpose technology (GPT) (Bresnahan & Trajtenberg 1995; Bresnahan & Greenstein 2001). Now even the inventive activity in many fields using software is highly dependent on advances in the supporting software (Ding et al. 2010; Cusumano 2004). But from its initiation as a formal field of technology more than fifty years ago through the present day, there has been no general consensus on a "best" way to organize development efforts (Brooks 1995, Cusumano 2004, Cusumano et al. 2003).

Over the course of this more than half a century, a plethora of firms, government agencies, universities, organizations, and user communities has been concurrently developing software in many different ways (Mowery 1994). Over time, computing architectures also evolved, from centralized, mainframe machines, to decentralized minicomputers and personal computers, to networked personal and special purpose computers that can reside on a desktop or in a lap and even in a mobile phone or parking meter. The consequent pervasiveness of computing has generated ever greater demand for software and at the same time increased the number and diversity of organizations engaged in creating it.

The mainframe computer was the epitome of the centralized computing era. Because the first mainframes sold by computer manufacturers in the 1950s *did not come with any*, software was by default written mostly by staff of the user organizations (Campbell-Kelly 2003). A customer was expected to expend as much money for programmers as for a computer, it not being uncommon for the purchase of a single mainframe to be accompanied by the hiring of 30 full time programmers (Campbell-Kelly 2003). Although firms like IBM provided training and support for programmers, each user establishment organized software development according to its particular perspective on the problems at hand and the best way to solve them. The significant duplication of effort inherent in the development of even the most basic functionality, and the serious bottleneck represented by programming staff and cost, were soon recognized by users, who, with the encouragement of computer vendors, established user communities that promoted the sharing of code.[2] IBM's user community, called SHARE, within a year of its founding in 1954 had 62 member organizations each of which was estimated to have been saved approximately $1.5 million by the 300 programs distributed among them (Campbell-Kelly 2003).[3]

As business users were busily sharing code with one another, the US military, concerned with Soviet nuclear supremacy, was seeking to develop an automated air defense system capable of detecting incoming bombers and missiles. The core of the system was a Q-7 computer that was to integrate radar, communications, and technical analysis. The contract to manufacture the computer hardware, awarded to IBM, generated revenues in excess of $500 million and employed about 25% of the company's workforce, approximately 8,000 people in the 1950s (Campbell-Kelly 2003). Curiously, IBM declined to develop the software for the Q-7, claiming that it would not know what to do with the labor when the project was completed. When other vendors turned down the job for much the same reason, the US Department of Defense tasked RAND Corporation to develop the requisite software. RAND subsequently established a subsidiary company, which it called System Development Corporation (SDC), and by 1959 had 700 programmers (by some estimates, half the professional programming population in the United States at the time) supported by 1,400 additional staff working on the system (Campbell-Kelly 2003). Owing to its military origins, SDC's approach to developing software emphasized standardization, requirements formation, and agreed upon coding solutions.

In the wake of increasing demand by the military for computer software for information processing and command and control systems, large, well-established government contractors began to branch out into software to compete for the lucrative development

[2] Campbell-Kelly (2003) reports that the cost of programming a single instruction on a computer could be as high as $10 (pg 33).
[3] SHARE was not an acronym but rather a statement of purpose, specifically, to promote the practice of sharing information and programs among users (Campbell-Kelly 2003).

contracts (W. Edward Steinmueller 1996). The need to combine computer hardware and software led electrical engineering firms like General Electric and Westinghouse, computer and business machine manufacturers like NCR and Remington Rand, aerospace companies like Lockheed and Boeing, and defense contractors like TRW and the Planning Research Corporation to create separate divisions to act as system integrators, porting to the world of software processes and approaches used in the development of their respective technologies.

IBM overcame its reluctance to develop software in mid 1950s, when it helped American Airlines create the first computerized travel reservation system, SABRE. The launch of SABRE heralded the beginning of business applications that went beyond bookkeeping and accounting to exploit computers to create core competitive advantages. This new source of demand for specialized software intimately tied to computer hardware gave rise to a number of startup firms founded by ex-employees of established software integrator firms and computer manufacturers as well as user firms (Steinmueller 1996).

Until the mid 1960s, software not purpose built for specific functions tied to particular hardware configurations was either provided for free by computer manufacturers or developed by and shared freely among users. Various software functionalities it developed for different industry verticals throughout the 1950s and 1960s, for example, were given away by IBM, and users were so used to sharing that "everybody who developed a piece of software was only too happy and flattered to have somebody else use it" (Campbell-Kelly 2003).

The concept of software as a product distinct from computer hardware, that could be sold by a third-party provider to users, emerged accidently in the wake of the collapse of a business arrangement between RCA and Applied Data Research (ADR) for the development of custom software. In 1964, RCA requested that ADR create a programming analysis tool that would enable its software developers to create logic flow charts of their programs. ADR complied, but RCA declined to accept the delivered software. ADR being left with significant outstanding expenses, president Martin Goetz decided to sell the software directly to RCA computer hardware customers. The market for IBM computers being significantly larger, he reasoned that porting the software to that platform would increase sales. ADR priced the software, called Autoflow, at $2,500. Thus was the first ever software product and company established (Campbell-Kelly 2003).

Due to the continued availability of "free" software, demand for software products remained limited until 1969, when IBM, under pressure from the US Department of Justice for anti-trust violations, "unbundled" its computer hardware, software, and training and subsequently charged separately for each product (Campbell-Kelly 2003). Its decision appears to have had two important effects, (1) it set an independent and separate price for formerly "free" software for which users would now have to pay, and (2) it opened the door for other companies to market products superior to the IBM suite.

The market for software products was fueled by the debut, in the mid 1960s, of the minicomputer, pioneered by Digital Equipment Corporation. Its PDP-8 minicomputer had a relatively low technical performance specification, 6%-22% of IBM's mainframe capability, but cost a similarly low 6% of the cost of the much larger computer (Steinmueller 1996). The low price of minicomputers spurred a significant market uptake among smaller customers that could afford neither the prices charged for, nor the numbers of dedicated staff required to support, IBM equipment. Minicomputers served single purpose uses such as machine control, data entry, and data processing, and with many more uses and users, and a growing number of software providers, a decentralized model of computing began to emerge. By 1974, 2,928 software products (ranging in price from $500 to $20,000) were offered by 740 different vendors (Campbell-Kelly 2003) and the 34,000 minicomputers sold compared to 8,900 mainframes (Steinmueller 1996).

The availability of relatively cheap minicomputers also democratized the craft of software development. Putative developers no longer needed to work at large firms or startups with access to computers. As minicomputers spread throughout the economy, more users became acquainted with computing and software development.

The perhaps most interesting side effect of minicomputers was the formalization and development of the open source software ethos whereby a relatively small set of actors or even individuals could independently create relatively large system instead of resorting managerially led teams. The development of the Unix operating system within Bell Labs is an example. The first version of Unix to run on a DEC PDP-11 was written, initially by just two developers, Ken Thompson and Dennis Ritchie, in about one month in 1969 (Salus 1994). Ostensibly developed to enable the creation of a patent management program for Bell Labs, Unix's genesis was, in fact, Thompson's frustration at not being able to play his "Space Travel" video game on decent, cheap hardware. In the space of a year, Unix was a full-fledged operating system distributed freely, consequent to AT&T's consent decree, to any organization that wanted to run it. Equally important, the code could, with relatively little effort, be modified and the operating system ported to any other hardware platform. The Unix community essentially applied the ethos of the earlier days of SHARE to all aspects of program development, not just applications and rapidly many hundreds of independent contributors created software for the operating system (Salus 1994). This freewheeling development forum provided yet another sustained source of software development practices. Today, a straight line can be drawn from the birth of Unix to the creation of community-based open source software products.

But the distribution of software development among different venues did little to abate the inefficiency and ineffectiveness of many of the discipline's practices. The tar pit persisted throughout the 1970s, with many organizations struggling mightily with increasing costs, extended delays, poor quality, and an insufficient supply of software developers. Some posited as a solution embracing the notion of factory development for software. Here, Japanese firms led the way, with the American firms trying at various times, mostly unsuccessfully, to follow suit (Cusumano 1991). Core to the software factory mentality was the belief that the labor supply for software development would remain chronically low in Japan, necessitating reliance on lower-skilled workers (Cusumano 1991). The factory approach embraced and formalized the "waterfall" process that imposed a sequential separation between high-level design and detailed, functional design in programming and testing. In keeping with the factory mentality, standardization and documentation of programming, rigorous supervision of staff, and an emphasis on code reuse were designed into the system. Although many US companies flirted with, and even implemented, some of the associated ideas, software factories are still found mostly in Japan, and to a limited extent in India (Cusumano 2004).

The distributed computing paradigm, and software development business and practice, experienced a revolution with the introduction of the personal computer (PC) in the late 1970s. The introduction of the Apple II in 1977 was a seminal event. It was Apple and its many imitators that envisioned computers being not just for business and government, but also for *personal*, use. Personal computers were marketed to hobbyists, housewives, students, and educators, and software for this new market, at a price point of less than $100, was developed not by existing players but by a wave of new entrants.

The breakthrough product in this era was developed by MIT alum and Harvard Business School graduate student Dan Bricklin to make his accounting and finance homework less tedious. Visicalc, created by Bricklin and colleague Bob Frankston in the former's attic as a way to solve the problem of updating numbers in previously hand coded spreadsheets, quickly catapulted PCs from hobbyist machines to serious business tools. Many firms would buy Visicalc, which sold for less than $100, bundled with the Apple II for more than $1,200 just to access the spreadsheet functionality. Visicalc was the harbinger of small, dedicated software companies that became overnight successes by creating products for the PC market, the so called "killer application."

IBM's entry into the PC market in 1981 further legitimized decentralized computing and expanded the base of software that needed to be developed. Now PCs had legitimate business uses as well as a range of other uses as diverse as desktop publishing, graphic design, and music production. In bundling Microsoft's PC-DOS operating system with its PC, IBM provided the blueprint for entry by many new software firms with product offerings that could satisfy the demands of the expanding market.

Of the several thousand firms that entered the market during the first ten years of the PC era, fewer than 100 achieved sales success exceeding $50 million (Campbell-Kelly 2003). Many of these firms were small ventures that turned out robust software products developed by as few as one, two, or three individuals. Larger firms, after launching their initial products, when new features, updates, and ongoing maintenance were required, quickly succumbed to the vagaries of complex software development (Cusumano 2005).

Although it struggled over the years to provide the functionality, quality, and timely delivery demanded by customers (Cusumano 2004) Microsoft pioneered a model that could accommodate the increasing complexity of, and changing market conditions that impinge on, PC software development. The core of Microsoft's approach is the distribution of work among small-feature teams, each of which develops one at a time software functionalities over the design and development of which it has full control. Programmers are encouraged to innovate and experiment, but also to synchronize frequently with other teams their designs and work output. Here, making the team central enables the development of programs with as many as several million lines of code, and all activity centers around synchronizing and stabilizing the code on a regular basis (Cusumano 1997).

The networked machine dominates the current computing era. Government funded and developed for more than 30 years by DARPA and ARPA, the Internet reached mainstream consciousness with the concurrent inventions of Hyper Text Markup Language (by Sir Tim Berners Lee) and the supporting technologies of the Web browser (by Marc Andreesen) and Web server (by Rob McCool) as instantiated in the World Wide Web. Now, with islands of personal computers, mainframes, and minicomputers able to be connected globally, new information sources and services emerged to exploit the opportunities afforded by this vastly expanded connectivity. Web technologies further lowered barriers to entry for software programming. Now, anyone with a computer and network connection could write software for the Web, often developing in days functionality that would take weeks if not months to develop in traditional settings. Connectivity was extended beyond computers with the advent of Internet-connected mobile phones that further expanded the population for which software could be written. The iPhone platform alone spawned more than 250,000 software applications that are routinely developed by teens as well as by startups and established firms and organizations.

The results of this evolution is a sector of enormous scale, importance and varied organizational approaches. The software sector expends, on average, 10%-15% of revenues on R&D, equivalent to the pharmaceutical sector (Campbell-Kelly 2003; Pisano 2006). The confederacy of organizational approaches that has emerged includes individual users, communities, internal development activities, custom, packaged, and Web software developers, and software services firms employing myriad approaches. Table 1 shows there to be approximately 10 times as many organizations dedicated to the software sales business model (more than 110,000 establishments) as pharmaceutical and contract research science organizations. But the number of employees per firm is between four and ten times smaller compared to the commercial science sector. Figure 1 plots the large economic footprint of the confederacy of software developers, with 2007 US sales exceeding $300 billion, surpassing the pharmaceutical industry by about $100 billion.

<*Insert* Table 1 Number of Employees and Establishments in the Software, Pharma and Contract Research Industries (2007)>

<*Insert* Figure 1 Comparison of Revenues in the Software Industry to those of Other Sectors >

2.3 Inside The Confederacy: Software Developer Motivations and Preferences

From its inception as a field of endeavor, and as the economic and strategic significance of software development increased, there have been many attempts to comprehensively understand what drives and motivates developers. This has been an area of particular interest given that motivation has over the years been thought to be crucial to software developer productivity and quality, and has been viewed as difficult to manage (Sarah

Beecham et al. 2008; Sharp et al. 2009). The history of research on industry workers' motivations and behavioral orientations is large; Sharp et al. (2009) estimate that more than 500 papers have been written on the topic. A rising interest in varied motivations of developers has also to some degree mirrored the changes and rising heterogeneity of organizations in the industry, and as quality, completion, and turnover of employees became significant issues (Bartol and Martin 1982).

Although software development projects may end up in the tar pits, the workers who participate in creating the programs tend to have a high affinity for the task itself (Bartol and Martin 1982, Beecham et al. 2008, Sharp et al. 2009) and infrequently face employment shortages. The attraction to software variously lies in the sheer joy of building and inventing, contributing to society through useful outputs, pleasure of participating in the problem and puzzle solving activity that is at the heart of programming, and continuous challenge of learning new techniques and approaches (Brooks 1975). Software developers also tend to identify more with the profession and occupational community than with the organizations in which they toil (Couger and Zwacki 1980).

Many of the same attractors reported by software developers are reported by scientists engaged in discovery work (Merton 1973, Dasgupta and David 1994, Stephan 1996), which tends to exhibit similarly high levels of uncertainty and failure. But whereas the republic of science acts as an institutional framing device for scientific workers' work preferences by virtue of a lengthy education process culminating in a PhD, the education profile of software developers exhibits considerable heterogeneity. More than one-quarter of software developers lack even an undergraduate degree, and the 1% that earn PhDs compares to 25% among science workers (Table 2). Software developers' preferences for style and type of work thus tends be based more on idiosyncratic experiences than on any overarching organized institutional frame. Moreover, as the number of individuals engaged in programming is roughly three times the number of scientific workers (Figure 2) one can expect much more diversity in preferences, absent strong framing.

<Insert Table 2 Education Levels for Software Developers and Scientists>

<Insert Figure 2 1992-2009:Total US Software vs. Science Employment>

Until the early 1980s, work on motivation tended to focus on trying to understand if those attracted to software development were somehow "different" and distinct from those in occupations in general, and in other technical and scientific occupations in particular (Weinberg 1971; Couger & Zawacki 1980; Schneiderman 1980; Bartol & Martin 1982). The research indicated that developers were similar to those in other occupations in terms of placing high value on jobs that provided interesting work and opportunities for growth, achievement, and career recognition (Bartol and Martin 1982). However, their need for achievement and growth, in particular, was significantly higher than exhibited by those in other occupations (Couger and Zawacki 1980), and they exhibited less concern for financial incentives (Bartol and Martin 1982). The work itself hence being viewed as the main reward for software developers.

Beecham et al.'s (2008) review of the post-1980 literature on the motivations of software developers found, consistent with past findings, that what was attracting people to software development were the inherent opportunities to learn, engage in problem solving, and explore new techniques. In fact, their review identified as drivers of engagement in software development 21 distinct motivators ranging from the need for rewards and incentives linked to performance, to task identification and variety, to supportive management, to a sense of belonging to a group and concomitantly enjoying a high degree of autonomy. Table 3Table 3 presents this list of motivators.

<*Insert* Table 3 List of Motivators Described in the Literature>

Further examining the literature to derive the defining characteristics of software developers, Beecham et al. (2008) identified sixteen characteristics that range from being growth oriented, to being concerned about independence and autonomy, to requiring competent management structures. Table 4 presents this list of characteristics. But Beecham et al. (2008) caution against reading too much into the frequency with which a motivator or characteristic shows up in the literature, emphasizing that the thrust of their findings is that software developers should be viewed as having complex, heterogeneous motivations and not as being a distinct, homogenous occupational group.

<*Insert* Table 4 List of Characteristics of Software Engineers Described in the Literature>

The same stream of literature also uncovered the alarming, at least to the researchers, finding in several studies that software developers indicated the least need for social interaction both on and off the job (Couger and Zawacki 1980). Compared to those in other occupations, software developers simply preferred to work on their own and interact as little as possible with colleagues, managers, or fellow employees. Leading researchers with normative concerns for the profession were concerned about this finding. Schneiderman's (1980, pg 124) discussion of the low social needs of programmers is illustrative.

> The first two decades of programming history produced the image of the introverted, isolated programmer surrounded by stack of output. Other workers have left the office, but our intense programmer, ignoring the absence of colleagues, scribbles rapidly, with a felt pen, in hopes of eliminating the last annoying bug before a 9 AM deadline. Fortunately, this image is becoming only a wild caricature of reality. The lonely days of the programming frontier are giving way to community, interdependency, and stability. This passage is happening gradually—the pioneers still resent the settler groups and seek to preserve their freedom and independence. The image of the programmer as a "loner" appears to have some validity, but it is hopefully changing.

The formal discovery of the anti-social nature of the software developer flew in the face of the widespread belief that interdependent team structures improved productivity at the individual level and were better suited to tackling more complex tasks (Schniderman 1980, Couger and Zawacki 1980). During the late 1970s, software organizations explored various team structures with an eye to improving the reliability of software, reducing time to completion, and generally making software development tasks more predictable. Among these were traditional hierarchical teams driven by senior programmers with varying numbers of junior programmers at their beck and call, “egoless” teams that favored an egalitarian community infrastructure over a competitive environment, and “chief programmer teams” in which intellectual responsibility for the design and development of the software code rested with a senior programmer assisted as needed by a host of skilled sub-specialists (Schneiderman 1980). But even as these various team structures were being evaluated, growth in the demand for software, which skyrocketed in the wake of the PC revolution, was bringing into the market increasing numbers of developers working both within organizations and on their own, the latter, as noted earlier, often able to produce software products equivalent to those output by the development teams working within organizations.

Beecham et al.’s (2008) analysis of the literature also surfaces, however, seemingly opposing findings in other research that indicates developers in other study sites have been found to sociable, identifying with a group, and willfully working in an interdependent setting. Thus, it would appear that some developers may embrace teams and prefer to work in highly socialized, interdependent environments and others prefer to work autonomously in solitary surroundings. The consequence for the literature is that the institutional characteristics of jobs and personal preferences of individuals will interact to predict software developers’ levels of effort and performance (Beecham et al. 2008).

3 Experimental Design

In the remainder of the paper, we consider the possibility that the extraordinary heterogeneity in organizations and in workers in the industry are somehow linked. Our central goal here is to estimate the extent to which assigning individuals to work within the regime they prefer (or have a “taste” for) influences how hard they work. The approach is quite simple: allow half the participants to choose between a “cooperative” or "competitive" regime and compare how they perform relative to individuals in a randomly assigned control group. Thus, although we do not directly observe individuals' behavioral orientation towards, or taste for, a given regime, we do rely directly on their stated preference. We then compare, in Figure 3 below, the efforts of group A' with A and of B' with B.

<*Insert* Figure 3 Comparison of Self-Selection into Regimes versus Random-Assignment>

3.1 Context

Given the emphasis on measure the size of effect in relation how different *types* of workers behave under different circumstances, a field setting has a clear advantage of providing more meaningful estimates than a lab setting. Nonetheless, estimation of selection effects requires an especially controlled environment. We conducted the experiment on the TopCoder open software innovation platform. This provided a field context with real, elite software developers; but, at the same time, the context provided an unusual ability to perform manipulations and to observe relevant microeconomic variables. Over the 10-day period of the experiment, participants developed computational algorithms to optimize the Space Flight Medical Kit of NASA's Integrated Medical Model (IMM) Team. TopCoder provided substantial assistance in altering the platform to enable us to run a multitude of treatments concurrently and in isolation, with setting up the NASA problem on the platform, and with running the experiment.

The solution to the real, highly challenging computational-engineering problem of developing a robust software algorithm to recommend the ideal components of the space medical kit included in each space mission was to be used by NASA. The solution had to take into account that mass and volume are restricted in space flight, and that the resources in the kit needed to be sufficient to accommodate both expected and unexpected medical contingencies encountered while in space, lest the mission have to be aborted. The content of the kit also had to be attuned to the characteristics of the space flight and crew. The challenge was thus to develop an algorithm that addressed mission characteristics that traded off mass and volume against sufficient resources to minimize the likelihood of medical evacuation. The problem, being relatively focused, was expected to be solved as a integral project capable of being divided into a set of subroutines and call programs. These sorts of projects might be solved by open source or corporate development teams composed of as many as 10 people (Carmel 1999). They are also regularly solved by participants in TopCoder tournaments.

The point of the experiment was to use self-selection as an instrument for revealing participants' (unobservable) behavioral inclinations towards one regime or the other. But if self-selection is in any way correlated with skill levels, we cannot attribute differences in outcomes exclusively to behavioral characteristics. Therefore, we must somehow remove or account for systematic differences in the skills of individuals and groups.

3.2 Assignment Procedure: "Ordered Pair" Matching on Skills and Randomization

One way to address the need to compare groups of similar composition is to exploit TopCoder's skill rating system, simply applying controls when statistically comparing outcomes between groups and individuals. However, we go further to deal with observable as well possibly unobservable characteristics with the combination of a kind of matching and randomization procedure.

The process described here is summarized in Figure 4. The goal of the process is to create groups or "rooms" of 20 participants drawn from the same skills distribution (and equivalent unobserved characteristics), but who had different tastes for the two regimes. The construction of treatment (self-selection) and control (random allocation) groups begins by dividing the participants into two groups (the self-selection and random assignment groups) with identical skills distributions (Step II in Figure 4). This is accomplished by ordering all participants in the population from top to bottom according to their skills rating. We divide this rank order into ordered pairs (top two highest skills, third and fourth highest skills, etc.) and randomly allocate one member of each to the self-selection group and the other to the random-assignment group.

We then asked just the participants in the self-selection group which regime they preferred to join (Step III). This was done in private bilateral communications between the TopCoder platform and individual participants, each of whom was asked: "Might you be interested in joining a team to compete against other teams?" Relative preference for the competitive or cooperative regime was to be indicated on a 5-point Likert scale.[4] The resulting subgroups were assigned to the cooperative and competitive regimes.

Implicit in this non-random self-selection to cooperative and competitive regimes is that these groups will not have equivalent skills differences. By assigning ordered pairs within the random-assignment group to the same regime, however, we assure that the self-selection and random-assignment groups within the cooperative and competitive regimes remained identical. We thus constructed groups identical in skills distributions that differed systematically in terms of their preferences for regimes. The self-selection group was uniformly orientated towards the regime to which it was assigned; the random-assignment group had the population average preferences, with some individuals preferring, and others not, the regime in question.

The self-selection groups of cooperative and competitive participants were then divided into groups of 20 individuals, in virtual web-based "rooms" (Step IV). Cooperative rooms were formed of four teams composed of five indivuals (also randomly formed). We "mirror" this random assignment in the randomly-assigned group, but essentially the ordered pairs to comparable groups. Thus, in randomly assigning the $1000 cash prize

[4] Participants were first asked their preference between the regimes, then given the following options: (1) I DEFINITELY would prefer to join a team; (2) I think I MIGHT prefer to join a team; (3) I am indifferent or I am not sure; (4) I think I MIGHT prefer to compete on my own; and (5) I DEFINITELY would prefer to compete on my own. They were then provided with additional descriptive details about each of the regimes and asked the same question. We then asked them to consider the possibility that both cooperative and competitive regimes were always available on the TopCoder platform and to indicate on a provided list of options what fraction of their time they would imagine spending in either regime. The order of responses, whether oriented towards the competitive or cooperative regime, was randomized. The second question (the one asked after clarifying the precise rules of each regime) was used as the basis for making allocation decisions.

awards to rooms in the self-selection group, we can then assign these prizes to the identically-distributed rooms in the random-assignment group (Step V).

<Insert Figure 4 Illustration of Assignment Procedure>

3.3 The Experimental "Institutional Regimes"

3.3.1 Competitive Experimental Regime

In creating the competition regime, we attempted to follow the existing open innovation competition (contest) regime regularly employed by TopCoder. Our primary unit of analysis of competition was the 20-person groups of direct competitors we created, this being the level at which the main prizes were rewarded. We choose $500 for the first place, and $200 for the second place, prize for each group of 35,which is considerably higher than average price levels but well within the typical range of prizes-per-contest participant. A list of competitors and skill levels and solutions submitted to date was posted for participants within a given group of direct competitors (but not to other groups).

3.3.2 Cooperative Experimental Regime

The open cooperative regime is demarcated by four, isolated, five-person "teams." Other factors that differentiated the cooperative from the competition experimental regime were communication, sharing, and information regarding competing teams and the payoff scheme.

Within-team communication was via a message board (i.e., communications within a group were common knowledge and broadcasted rather than bilateral). Code could also be posted to the discussion board, but more direct sharing of output was possible as all code submitted to be compiled, and compilations scores, were viewable by all members of a team (but not across teams). Participants could observe details of all individuals on their respective teams, but only average skill ratings and the highest ranked submissions of other teams.

The payoff scheme was broadly the same as in the competition regime, the top two submissions receiving prizes of the same amount. We did not impose "team" payoffs, simply allowed winning submitters to allocate prizes and recognition as they deemed appropriate. Coders making a submission were asked if they drew on other coders' inputs or contributions and how they felt the prize money should be allocated. An alternative (perhaps more obvious) approach would have been to impose a sharing rule, but we felt that self-determination was closer to the ethos of most collaborative development teams. We followed this approach after consulting with TopCoder executives and interviewing a number of TopCoder members. Prize magnitudes were intended to weakly, and only weakly, suggest the possibility of evenly dividing prizes among five team members. Table 5 summarizes details of payoffs. Note that the proportion of competitive and cooperative groups randomly assigned to compete for no rewards is equal. All participants who submitted code were also eligible for a custom t-shirt, and the top performing entry, regardless of prize treatment, received $1,000. The total prize pool for the experiment was $25,000.

<*Insert* Table 5 Key Features of the Experimental Regimes>

4 Sample and Variables

The total sample population was 1,098 individuals.[5] For the analysis, we dropped observations for which skill rating information was insufficiently precise or difficult to interpret. This included individuals with zero ratings and individuals who had not logged a rating in the type of competition known as "algorithm contests," on which we focused given that it is the most popular rating and most relevant to the sort of problem-solving

[5] Several hundred more had subscribed, but we decided, in close consultation with TopCoder, to keep only the 1,098 individuals who had established TopCoder skill ratings. We would have preferred to also drop those with zero ratings and without the "Algorithm" rating we treat as the salient skill rating, but this was not possible as a practical matter for TopCoder in managing its community of coders.

involved in the experiment.[6] The distribution of the 1,040 observations in the sample used in the analysis is presented in Table 6.

<*Insert* Table 6 Sample Breakdown>

It should also be emphasized with regard to our research objective of measuring the selection effects of a sort that the TopCoder membership hardly represents a random sample of individuals from the economy, or even from the software developer labor market. At the time of the experiment, some 15,000 TopCoder members relatively regularly participated on the platform. Because the population in the experiment reflects a choice to voluntarily participate, the results should be interpreted as "treating on the treated," or assigning what is a non-random population to different treatments. Although there is considerable diversity in this group, which includes individuals from many countries and from industry as well as students and researchers, it remains a subset of the wider population of the global software developer labor market, and estimates of effects of self-selection versus random assignment of workers should therefore be smaller than what might be possible were we to construct a more diverse sample from the broader labor market.

Of those in the self-selection group who were asked, 34.9% expressed a clear preference for the cooperative, 50.5% a clear preference for the competitive, regime.[7] The remaining 15.6% of participants in the self-selection group expressed uncertainty or indifference between the regimes. We interpreted this indifference to indicate some openness to the cooperative regime (TopCoder's usual regime is similar to the competitive regime). In the interest of balancing numbers in the regimes, we elected to allocate the indifferent responses to the cooperative regime, resulting in a breakdown between the competitive and cooperative regimes that was remarkably close to half-and-half. (As dropping the indifferent observations had little effect on the results, they were left in the analysis.)

[6] Experimenting with alternative approaches to including observations did not appreciably affect our results. We present results in relation to the preferred sample given that their interpretation requires the least qualification and fewest caveats.

[7] We originally targeted half the entire group of 1,098, but did not receive responses from a small fraction of individuals.

As anticipated in the matching and randomization procedure, a clear correlation between preference for the competitive regime and skill level emerged. Figure 5 presents a flexible, locally weighted kernel regression of the relationship between a preference for competition and different levels of *SkillRating*. Higher skills are clearly positively related to a preference for the competitive regime.

Given that treatments were in units of groups of 20, we created 14 groups of self-selected competitors through random assignment. We then assigned the ordered pair of each of these self-selected competitors to 14 additional "mirror" rooms of competitors. This assignment procedure resulted in 28 full rooms of 20 competitors. (We created a 29th room that had fewer than 20 participants to allow other individuals to participate.) Twelve of the 28 rooms, six assigned rooms and six self-selected rooms, competed for cash prizes amounting to $1,000 per room. We chose 12 simply because participation in the experiment exceeded expectations, and we had not budgeted for more than 12 prizes for the competitive regime. Prizes were first assigned randomly across the self-selection rooms. The "mirror" rooms of ordered pairs with corresponding assigned competitors were then also allocated $1,000 prizes.

<Insert Figure 5 Preference for Competition versus Skill Rating>

4.1 Variables

This section discusses the meaning and construction of variables used in the analysis. Table 7 presents variable definitions, Table 8 descriptive statistics.

4.1.1 Dependent Variables

We have both observational and self-reported survey measures of effort. The observational measure is the number of submissions made by each participant over the course of the 10-day experiment (*NumSubmissions*). This is a direct indication of the intensity of development, given that code testing and evaluation required that code be submitted to the platform so that its performance in relation to the test suite could be assessed and it could be assigned a score. (Participants' best, and typically last, submission became their final score.) Submitting code in this fashion was costless and resulted in virtually instantaneous feedback.

Our preferred main dependent variable records the total number of hours participants invested in the preparation of solutions throughout the course of the event. This self-reported estimate of the total number of hours worked (*HoursWorked*) was reported in a survey administered the day after the event closed.[9] (Participants were required to respond to this question electronically, as the experiment closed, in order to receive a NASA-TopCoder commemorative t-shirt imprinted with their name.) *HoursWorked* is our preferred variable, as it directly conveys meaning (and perhaps even some indication of value) and is easily interpreted. The results do not depend on which of the two measures of effort we use in the analysis.

<*Insert* Table 7 Variable Definitions>

<*Insert* Table 8 Descriptive Statistics>

4.1.2 Explanatory Variables

The key explanatory variable, *SelfSelect*, indicates whether a competitor was in a self-selection or random assignment group. A second explanatory variable, *Prize*, indicates observations/individuals associated with rooms for which there was a $1,000 cash prize. A third explanatory variable, *Competition*, is set to one to indicate the competitive, and zero to indicate the cooperative, regime.

[9] Nearly all participants who submitted solutions responded. A research assistant who contacted 100 of the non-submitters who did not respond to the first survey found that each had devoted less than one hour to the project and had not made a submission. This enabled us to complete the non-respondents by filling in zero hours as a relatively precise approximation. It became clear through interviews with non-submitters that they generally believed they would not receive a commemorative t-shirt whether they responded to the survey or not, accounting for the sharp difference in response rate between submitters and non-submitters. Worthy of note, however, is that a number of non-submitters whom we discovered had worked a non-trivial number of hours before choosing not to submit did respond to the survey.

Our measure of general ability to solve algorithmic problems is TopCoder's own rating system, which essentially calculates a participant's ability to solve problems on the basis of past performance. We refer to this variable as *SkillRating*. We use specifically the rating calculated for what TopCoder terms "Algorithm" matches, software solutions to abstract and challenging problems akin to the problem in the experiment.[10]

4.1.3 Additional Variables

In robustness tests, we use two additional variables collected for those in the self-selection group. The variable *LikertScale* captures the Likert scale responses of those asked their preferences. Recall that the numerical responses in this variable correspond to the following scale: (1) I DEFINITELY would prefer to join a team; (2) I think I MIGHT prefer to join a team; (3) I am indifferent or I am not sure; (4) I think I MIGHT prefer to compete on my own; and (5) I DEFINITELY would prefer to compete on my own. The variable *OrderofQuestion* captures whether the survey was designed to present all aspects of introducing regimes with the cooperative or competitive regime first.

5 Results

5.1 Cross Tabulation Comparisons

Given the design of the experiment, just the mean outcomes should provide meaningful comparisons, and, indeed, reveal that self-selection led to significantly higher levels, especially in the competitive regime.

The average number of hours worked by participants during the 10-day experiment was 10.54 (standard deviation = 18.74 hours). Self-selected individuals worked, on average, 13.27 hours (maximum 190 hours), randomly assigned individuals only 7.78 hours (maximum 120 hours). A coarser, perhaps starker indication of differences is the fraction

[10] This has been found through the decade of operation of TopCoder to be a robust measure of skills and is even commonly used in the software developer labor market when hiring. Nonetheless, to the extent that it might be imperfect, the randomization procedures (in particular, pair ordering and randomization of which party self-selects) should erase any possible systematic biases in estimates.

that chose to work more than 1 hour, which, for randomly assigned participants was 30% and for self-selected participants 41%.[11] *NumSubmissions* was also higher for self-selecting participants, at 2.79 versus 2.20 for randomly assigned participants.

Table 9 breaks down the effects for the competitive and cooperative regimes. Average *HoursWorked* was only slightly higher in the competitive (10.82 hours) than in the cooperative (10.27 hours) regime.[12] In both regimes, *HoursWorked* was significantly higher for self-selecting participants, the starkest differences being in the competitive regime (14.92 hours for self-selected participants versus 6.6 hours for randomly assigned participants, a 126% difference, compared to 11.57 and 8.97 hours, respectively, in the cooperative regime, a mere 29% difference).

For *NumSubmissions*, levels were also considerably higher for self-selected (3.77 submissions) than for randomly assigned (1.98 submissions) participants. That average *NumSubmissions* was lower for self-selected participants in the cooperative regime we speculate reflects greater coordination of activity across team members.[13] (Consistent with this interpretation, we find that self-selected teams posted greater numbers of intra-team communications on the private team, online bulletin board.)

<*Insert* Table 9 Comparison of Mean Effort Levels, Cross-Tabulations>

[11] Because there appears to have been a mass of participants that chose to work very little and another mass that chose a more continuous, widely distributed level of work, we report in Section 5 the results of an explicit distributional analysis.
[12] We found the differences in magnitudes to be surprisingly small and statistically insignificant, given the usual predictions of moral hazard in teams (Holmstrom 1982). Section 5.2.3 presents evidence consistent with the possibility of complementary effort choices that might plausibly be associated with any number of mechanisms such as socialization, and mutual monitoring in the cooperative regime that go beyond the usual notions of moral hazard in teams.
[13] Adding additional controls within a regression framework does not change this.

5.2 Regressions

Although the earlier comparisons of means provide meaningful results, analyzing the data within a regression framework enables us to explicitly assess the experimental assumptions and thereby better interpret the data. Ordinary least squares regression results with robust standard errors are reported in Table 10.

5.2.1 Assessing the Matching and Randomization Procedure

If the estimation procedure was effective and left no systematic differences across treatments, the estimates should be unchanged when we include skill controls.[14] We focus first on the results for the competitive regime. For ease of comparison, model (1) simply reiterates the two-way correlation of *HoursWorked* with *SelfSelection* from the competitive regime (essentially equivalent to the earlier stratified comparison of means in Table 9). Model (2) re-estimates the *SelfSelection* coefficient with *SkillRating* included as a control. The estimated coefficient is virtually unchanged, and the coefficient on the constant, which effectively captures mean effort without self-selection, changes slightly more (from 6.60 to 8.07), but the difference is statistically insignificant. To control for possible subtle non-linearities, Model (3) adds dummies for different bands of skill levels to capture possible non-linear effects, but the estimated coefficient on *SelfSelection* is statistically identical and virtually unchanged (8.36 versus 8.33). Model (4) provides a most stringest skill control by simply comparing each self-selected individual to its ordered pair, calculating the difference (implemented by simply including ordered pair fixed effects). The estimated effect is again statistically unchanged (although this most stringent control only leads to a slightly large coefficient). Given the random selection of rooms that should receive prizes, the introduction of *Prize* to the model should also not have any effect on the estimated coefficient *SelfSelection*.[15] Therefore, each of these coefficient estimates is thus statistically identical to the simple comparison of means presented in Table 9 (14.92 – 6.6 = 8.32 hours).

[14] This includes differences in skill or unobservables correlated with skill.

[15] We must go back to a model estimated on the basis of ordered pair differences, given there is not variation in Prize within ordered pairs, given the assignment procedure assured that if one member of an ordered pair was in a group with a prize, the situation would be mirrored for the other pair.

Importantly, the coefficient on *Prize* also provides some indication of the relative impact of self-selection versus the formal incentive instrument used in this context, the $1,000 prize. The coefficient on *Prize*, 9.12 hours with a standard error of 1.85 hours, is statistically indistinguishable from the effect of allowing individuals to self-select to competition for cases in which competition is their preferred regime.

An analogous set of regressions performed on the cooperative regime similarly confirms that estimates of the *SelfSelection* coefficient are insensitive to the various controls. Model (6) reiterates the two-way correlation of *HoursWorked* with *SelfSelection* from the cooperative regime (essentially equivalent to the earlier stratified comparison of means in Table 9), 2.6 additional hours for inviduals who self-selected in the cooperative regime. Re-estimating the effect on the basis of directly comparing ordered pairs (model 7) or introducing *Prize* and controls for different levels of skills (model 8) generate similar estimates. The estimated coefficient on *SelfSelection* is 2.60 hours. Model (6) essentially re-estimates model (4) with each of the controls, but for the cooperative regime. Including each of the controls does not significantly change the coefficient on *SelfSelection* (2.47 hours). Again, these estimates are statistically the same as those from the simple comparison of means in Table 9 (11.57 – 8.97 = 2.60 hours). The effect of the formal cash incentive in the cooperative regime, as estimated by the coefficient on *Prize* (9.88 hours), is essentially the same as in the competitive regime (and the self-selection effect in the competitive regime), and considerably larger than the self-selection effects in the cooperative regime.[16]

<*Insert* Table 10 >

5.2.2 Hawthorne Effects

Our goal was to use revealed preference as a means of allocating individuals to the regimes for which they have an inherent preference or taste. Therefore, the earlier

[16] As earlier noted, this result is perhaps surprising in light of the theory of moral hazard in teams (Holmstrom 1982).

regressions are intended to estimate the impact of this alignment of taste with regime on choice of effort level. But it might still be the case that individuals work harder because of the simple fact they were asked for their preference at all. For example, if individuals equated being asked their preferences with implicitly making a choice (i.e., with the expectation that their stated preference would immediately translate to an allocation decision), they could have possessed a greater sense of accountability or self-determination. To mitigate this possibility, most importantly we designed the process in a way to avoid any direct implication that a statement of preference implied a choice or that an assignment would immediately follow this statement of preference.

It is also possible that those who were asked their preferences might have had higher awareness that this particular contest was being recorded as part of an experiment. We hoped that the regularity of such TopCoder contests, use of a high profile problem from NASA, and relatively large overall cash prize as well as the use of the regular TopCoder communication channels and platform would reduce the salience of the experiment in the minds of participants.

To discern the effect of eliciting preferences *per se* (as distinct from the effect of the assignment decision following these preferences) is challenging in an experiment where assignments directly followed revealed preferences. Our approach is essentially one of attempting to detect any such effect by comparing the subset of self-selected and randomly-assigned participants that are most likely to have neutral and similar preferences. Related results are presented in Table 11.

<*Insert* Table 11 Results of Test for Presence of Hawthorne Effects>

A first step in this strategy was to synthesize a variable that influenced an individual's choice but was completely unrelated to their preferences. To create such a variable, we randomized the order in which regimes were presented. Model (1) shows that this order significantly affected the expression of preferences, and Model (2) that this correlation is unaffected by adding additional control variables. We test for Hawthorne effects by re-estimating the earlier model using the order of the question as an instrumental variable (IV).[17] We run the model on the 15% of self-selectors who expressed the most neutral response when asked to gauge their relative preferences for regimes (i.e., those who expressed "I am indifferent or I am not sure"[18]). This IV test thus offers a means to better isolate individuals in this group who were susceptible to varying their responses with order (as opposed to, say, those who expressed neutral relative preferences that, in fact, reflected equal and opposing strong preferences for both regimes). Model (3) reports results for a simple specification without the instrumental variable; Model (4) uses the instrumental variable. In both cases, the estimated coefficient on *Prizes* is essentially unchanged, but the coefficient on *SelfSelection* becomes statistically insignificant and indistinguishable from zero, particularly in the IV specification. We therefore fail to detect any evidence of Hawthorne effects.[19]

5.2.3 Interaction between Individual and Regime or with Other Individuals

The experiment was designed to capture individuals' choices of effort in response to being assigned to the institutional regime they preferred. Individuals had no prior

[17] More precisely, this instrumental variable is an indicator (1-0) switched on in cases in which the choice of regime matched the order in which it was presented.
[18] Recall that indifferent individuals were assigned to the cooperative regime (Section 3.2).
[19] We emphasize that this is not a perfect estimate of the magnitude of the Hawthorne effect, but simply an attempt to detect any evidence of such an effect. Although it is naturally appealing to focus on the self-selectors with neutral preferences (in relation to those who express clear preferences), this group is not necessarily neutral in the sense of apathetic. They simply have no relative preference for one or the other (they may have equally strong preferences, for example). The IV attempts to go a step further to isolate those individuals that were especially susceptible influence by the order of the question, and therefore just partially increase the prominence of variation associated with those who would have been more susceptible to change in response to order. Another imperfection of this test is the control group. Ideally, we would compare individuals who chose to be in their preferred regimes with identical individuals who were assigned to their preferred regimes. The econometric test, however, compares individuals with their orderd pairs, equivalent in skill—but not with the same preferences. Given that the control group will sometimes allocate individuals to their preferred regimes and sometimes not, this imperfect control group will create an upward bias in our attempt to detect Hawthorne effects.

knowledge of the identities of participants in their chosen regime, only of the rules of the game. It nevertheless remains possible that the predicted effects of self-selection might emerge from interactions among participants rather than from the regime itself. For example, a hard-working competitor or teammate could plausibly occasion an increase or decrease in one's choice of effort level. Related results are reported in Table 12.

To measure social interactions, we included measures of *other* individuals' choices of effort level. Model (1) reruns the model on the competitive regime including the mean level of hours worked by others in the same room, designated *HoursWorkedGroup*. Whether regressed on the entire sample for the competitive regime (model 1) or on the self-selecting or random-assignment subsamples (models 2 and 3), the regression finds no association with effort level. (*SelfSelection* drops out of the model when we regress the model on subsets of self-selected and random-assignment participants individually.) Including any other measures of the distribution of effort—variance, skew, or max—or estimating on the basis of differences between ordered pairs also finds no effect.

Quite different patterns are observed when the test is repeated on the cooperative regime. Model (4) includes both the effort of others in the room, *HoursWorkedGroup*, and effort of others on the team, *HoursWorkedTeam*. This regression finds a clear association between choice of effort and the effort level of others on the same team. Dropping the group level measure (Model 5) increases the estimated magnitude of the relationship between choice of effort level and the effort level of teammates from .26 to .32.[20] Accordingly, including this interaction effect in the model reduces the coefficients on *SelfSelection* and *Prize* relative to the earlier estimates. This is because any direct difference occasioned by these factors should be amplified by social interaction. The estimated effect on *SelfSelection* even becomes statistically insignificant as it drops slightly in magnitude. (Re-estimating on the basis of differences in ordered pairs, or

[20] The simple interpretation of this coefficient is an association of a .32 hour increase in effort with a one hour increase in others' average effort. It is important that the coefficient capture "reflection" (Manski XX), whereby individuals, if the team increases its effort, will increase their effort, leading the team to further increase its effort, and so on. The coefficient should thus be taken as an association that relects this process.

including other measures of the distribution of team effort, does not change this result.) The direct impact of *SelfSelection* on team effort, already found to be considerably less than in the competitive regime, is found to be smaller still and dependent on (positive) social interactions to stimulate the effect. Regressing the model on subsets of self-selected (model 6) and randomly assigned (model 7) teams finds a substantially larger interaction in the case of self selection (.34) than in that of random assignment (.23), but the difference between them is not statistically significant.

<*Insert* Table 12 Results of Tests for Presence of Within-Group Interactions Among Effort Choices>

These tests suggest that the earlier-estimated coefficients on *SelfSelection* (i.e., models (10-1) through (10-8)) should be interpreted, in the case of the competitive regime, as reflecting a direct effect of matching individuals with their preferred regimes, and in the case of the cooperative regime, as reflecting this direct effect together with its amplification by social interaction.

5.3 Reweighting the Sample and Adjusting the Control Group to Recalibrate the Self-Selection Effect

The earlier estimates in Section 5 enabled us to compare the self-selection groups to the randomly assigned groups, controlling for skill (and randomizing to exclude the effect of unobservables). Although these estimates were meaningful, in this section we reconsider the composition of the compared groups to re-calibrate estimated magnitudes of effects. Results are reported in Table 13.

5.3.1 Re-weighting the Sample to Enable Direct Comparisons between Regimes

Whereas the skills distributions were, by design, the same across the self-selection and random assignment groups, they were unequal across the competitive and cooperative regimes (Figure 8, Panel I). This was unavoidable given that preferences and assignment were correlated with skill (Figure 5). Consequently, earlier estimates of the coefficients on *SelfSelection* in the cooperative and competitive regimes should not be directly comparable if the magnitude of an individual self-selection effect is somehow related to skill.

To enable more direct comparisons of the magnitude of effects in the cooperative and competitive regimes, we re-weight the data from the competitive regime to have the same skills distribution as that of the cooperative regime (as in Figure 8).[21] As reported in model (XX), re-estimating the model with the competitive data, but re-weighting to match the skills distribution of the cooperative data, increases the estimated coefficient on *SelfSelection* from 8.32 hours (model XX) to 10.28 hours.[22]

<*Insert* Figure 6 Skills Distribution in Cooperative and Competitive Regimes>

5.3.2 Synthesizing an Alternative Control Group to Recalibrate Self-Selection Magnitudes

The estimated self-selection effect will, of course, depend on the control group to which the self-selected group is compared. The earlier regressions essentially compare the self-selection group (in which 100% of participants preferred the regime) to a population average group. Although we do not directly observe the institutional preferences of members of the random-assignment group, we should expect the same distribution of preferences as the self-selection group, given the matching and randomization procedure. Based on the stated preferences of those within the self-selection group, we should expect roughly half (50.7%) of randomly assigned participants to prefer the competitive regime. (For compactness, we refer to propensity for competition rather than to propensity for both competition and cooperation at each turn.) important to note, however, the

[21] We re-weight these observations using an algorithm that will enable us to later simultaneously re-weight on a second dimension (propensity to choose competition). We divide the sample into 250-point *SkillRating* blocks and re-weight observations within each of these blocks to match the weight in the corresponding block in the cooperative regime.

[22] This result accords with the fact that the self-selection effect interacts negatively with skill level (not reported), and that the competitive regime has a disproportionately high proportion of high-skill competitors. Consequently, re-weighting the estimates towards lower-skill participants should increase the estimated effect.

proportion that prefers competition at any particular skill level should not be expected to conform to this aggregate average, as we saw a clear correlation between skills and preference for competition (Figure 5). The control group for the competitive regime should therefore include individuals more similar to the self-selection group within the band of participants with relatively high skills. This makes the self-selection group more similar to the control group among higher-skilled participant and less similar among lower-skilled participants, in terms of the propensity or inclination to prefer competition. (A reverse situation exists for the cooperative regime.) As a result, the earlier estimates of the self-selection effect in the competitive regime implicitly placed greater emphasis on the effect among lower-skilled participants. Although this meaningfully reflects the self-selection process observed here, the analysis investigates whether neutralizing this compositional effect significantly affects the magnitude of the estimate.

To synthesize estimates in which the effective propensity to choose competition does not change with different skill levels within the control group, we re-weight the sample on the basis of propensity to compete (while continuing to re-weight on the basis of skills, as above). Because we do not observe it, we estimate propensity to compete by constructing a propensity score, which we do by first modeling the likelihood within the self-selection population of choosing competition. In a Logit model, we regress an indicator variable switched to one for individuals who selected into competition on the basis of demographic variables collected by TopCoder when its members originally signed up. This includes dummies for country of origin, type of developer (student, professional, part-time, etc.), age of participant, and stated motivation for joining the TopCoder platform (learn, make money, meet other coders, etc.). This is a purely empirical model and so we simply add many explanatory variables. We use these same data and model coefficients to generate propensity scores for individuals of the randomly assigned control group. Consistent with the composition of the control group being similar to that of the self-selection group, a fitted curve of estimated propensity scores for the randomly assigned control group is similar to a fitted curve of the actual fraction of self-selection participants that chose competition, as in Figure 7 below. (Again, the high skilled competitors have a higher propensity than the aggregate average to choose competition and the low-skilled are lower than the aggregate average.)

<*Insert* Figure 7 Propensity to Compete Across Different Levels versus the Overall Population Average>

Model (XX) re-estimates the model for the competition regime, further re-weighting the control group data such that the average propensity score at every skill level is held to the population average (50.7%), as described above, and simultaneously re-weighting such that the skills distribution virtually matches that within the cooperative regime.[23] Consistent with a small self-selection effect in the case of relatively high-skilled participants, the coefficient on *SelfSelection* decreases slightly when the control group is normalized to shift emphasis towards the effect among higher skilled participants. Model (XX) estimates the same model with the data for the cooperative regime, which, sensibly, slightly increases the magnitude of the coefficient.

Re-weighting the sample and synthesized control groups enables more direct comparisons of effects across the cooperative and competitive regimes as well as unbiased estimates the self-selection and formal instrument (prize) effects. These are therefore our preferred estimates. To provide a more palpable sense of the substantive impact of allocating individuals to their preferred regimes, we use this approach to present the distribution of choice of effort for both self-selection and random assignment, as in Figure 8.

[23] This is possible because at every skill level there are ample observations both above and below the population average propensity. The re-weighting approach consisted of dividing the observations into ascending blocks of 250 *SkillRating* points and then linearly adjusting the weight of the observations within each block to match the population average propensity (i.e., the weight was $1+\omega \times$ Propensity Score, solving for the ω in each block). Each entire skills block was then reweighted to lead each group to have the same skills distribution.

<*Insert* Figure 8 Distribution of Effort Choices for Self-Selected and Randomly-Assigned Participants (Reweighted to Match Skills Distribution and with Skills-Neutral Control Group)>

6 Conclusions

Software design and development is done in settings as diverse as small entrepreneurial firms, large multi-national organizations, universities, outsourcing consultancies, collaborative efforts like open source software communities, and the proverbial solo developer in the garage. More software development occurs within user organizations than is done by firms dedicated to selling the software to customers. Perhaps at least as much as in other areas of the economy involving problem solving and innovation, there is a heterogeneous "confederacy" of organizations engaged in software work. In this paper, we explore the possibility that the presence of this wide range of organizations might in part reflect the varied and idiosyncratic behavioral orientations and sources of motivation of the workers who create the product, that is, the software developers. In short, we hypothesize that different "types" of developers may sort themselves into different "types" of organizations, and that this could have implications for efficiency.

We begin to explore this possibility via a field experiment in which we allocated individual workers to either a competitive (autonomous) or cooperative (team) software development regime. This regime to some extent represents the basic elemental form of how programming activity is organized by software development managers. We were mainly interested in determining whether assigning individuals to the regime of their preference affected how much effort they would exert over a 10-day period. The experiment was performed on the TopCoder platform, on which we created isolated "virtual rooms" in which groups of 20 individuals either worked autonomously, competing with one another to develop code, or worked cooperatively in teams of five that competed against three other teams. Each room of autonomous individuals and cooperative teams worked on the same software development task, a challenging "real-life" algorithmic optimization problem supplied by the Space Life Sciences Directorate at NASA. The challenge was to create a medical kit that optimized the mass and volume of the contents while minimizing the probability of astronaut evacuation in the event of a medical emergency in space.

Our approach was simply to ask half the subjects their preferences, and then assign these individuals to their preferred regimes. Thus, the first half of participants virtually "self-selected" into the competitive or cooperative regime. (Strictly speaking, we assigned these individuals to their preferred regimes in a way intended to be perceived as independent of their statement of preferences.) Here, we were attempting to determine if individuals' expressed "taste" for one regime or another subsequently affected the level of effort they expended. We used a combination of matching and randomization approaches to assure that the other half of participants were identical to the self-selected group in distributions of observable skills and other (unobservable) characteristics, but had different (population average) tastes for the regimes. The control group should thus be understood not as having opposing preferences, but rather as having a distribution of tastes, some preferring and some not the regime to which they were assigned. Our approach is also a departure from traditional experimental setups in which subjects are randomly assigned to treatments. Because we were interested in "selection as treatment," one half of the population had its preferences met.

Across the 1,040 individuals in the sample there was a wide distribution in effort levels chosen, with an average of 10.54 hours and a wide skew to the right of several dozen hours. A large minority, roughly a third, worked less than one hour. We found that allocating individuals to their preferred regimes had a significant impact on choice of effort level, particularly in the autonomous competitive regime, in which self-selected participants worked, on average, 14.92 hours compared to 6.60 average hours for the randomly assigned participants. The effect was also positive and significant in the team regime, in which self-selected participants worked, on average, 11.57 hours compared to 8.97 average hours for the randomly assigned participants. We found that the effect in the case of the team regime was not just the result of an initial boost of effort from self-selection to one's preferred regime, but also an amplification of this effect through social interaction, with individuals exerting higher levels of effort in the presence of teammates exerting higher levels of effort.

We found no evidence of Hawthorne effects (in the sense that being given a choice per se affected choice of effort level). Re-weighting the data in various ways to facilitate comparisons between the two regimes and synthesize alternative control groups did not qualitatively change the magnitudes of estimates.

Perhaps the most remarkable feature of these results is not just the positive effect of assigning individuals to regimes for which they have a taste or preference (controlling for skills and other characteristics), but the magnitude of the effects. That individuals worked such a large fraction of additional hours across the regimes is notable given that the experiment was run over a 10-day period of regular work weeks for most participants. The boost in effort was also large in relation to the average number of hours worked overall, 10.54 hours. Perhaps most telling, the effect of allocating individuals to their preferred regimes had an impact that was roughly equivalent to the effect of applying formal, high-powered incentives. Randomizing which rooms competed for $1,000 cash prizes and which rooms did not, we found that this formal incentive resulted in, on average, over 9 more hours worked.

We emphasize that these results are a preliminary step towards understanding the role and presence of heterogeneous individuals and heterogeneous organizations in the production of software (and perhaps in the economy more generally). The assignments applied in the experiment were exogenous and taken out of the context of a competitive economy or any sort of market equilibrium. Further, we examined the effect of one-sided sorting rather than two-sided matching, as when an employer or team leader might also screen, and found evidence of the potential for individuals' "tastes" for different sorts of institutions, and even simply the mechanical rules of the game, to have (very large) implications for efficiency. We might imagine more socialized identities and characters of "living" organizations further moderating behaviors and performance implications. This was even strongly suggested by social interactions found in the cooperative regime.

Although we tried, as best we could, to isolate an interaction between individuals' tastes and preferences for institutional regimes and the rules of the game, and despite a context that, with one-shot interactions over a short (10-day) period through a sparse and narrow digital communications medium, could hardly be described as "socially embedded," the effect of group interactions nonetheless surfaced in the results.

Although our field context is limited to software development, the core task required for success, creative problem solving, is representative of an increasing and important part of the economy. Fifty years ago, most US workers were engaged in the production of goods and services. Today, roughly half of the more than 40% of workers that have over the past decade come to be classified as information processors (Radner 1993) are considered problem solvers engaged in professions like management consulting, social policy development, programming, law, and medical and scientific research (Hong & Page 2001). Thus, our findings have general implications for the organization of creative problem solvers engaged in inventive activity. Inasmuch as managers have significant discretion over how problem-solving work is organized, the preferences of workers and design of work may be important considerations with significant implications for efficiency, productivity, and performance.

7 References

Azoulay, P., Zivin, J. & Manso, G., 2009. Incentives and Creativity: Evidence from the Academic Life Sciences. *NBER Working Paper #15466*.

Bartol, K. & Martin, D., 1982. Managing Information Systems Personnel: A Review of the Literature and Managerial Implications. *MIS Quarterly*, 6, 49-70.

Beecham, S. et al., 2008. Motivation in Software Engineering: A systematic literature review. *Inf. Softw. Technol.*, 50(9-10), 860-878.

Belenzon, S. & Schankerman, M.A., 2008. Motivation and Sorting in Open Source Software Innovation. *SSRN eLibrary*. Available at: http://papers.ssrn.com/sol3/papers.cfm?abstract_id=1401776 [Accessed September 3, 2010].

Bresnahan, T. & Greenstein, S., 2001. The economic contribution of information technology: Towards comparative and user studies. *Journal of Evolutionary Economics*.

Bresnahan, T.F. & Trajtenberg, M., 1995. General Purpose Technologies 'Engines of Growth'? *Journal of Econometrics*, 65(1), 83-108.

Brooks, F.P., 1995. *The mythical man-month: Essays in software engineering (Anniversary Edition)*, Reading, MA: Addison-Wesley.

Brooks, F.P., 1975. *The mythical man-month : essays on software engineering*, Reading, Mass.: Addison-Wesley Pub. Co.

Campbell-Kelly, M., 2003. *From Airline Reservations to Sonic the Hedgehog: A History of the Software Industry*, Cambridge, MA: MIT Press.

Carmel, E., 1999. *Global software teams : collaborating across borders and time zones*, Upper Saddle River, NJ: Prentice Hall.

Cockburn, I.M. & Henderson, R.M., 1998. Absorptive Capacity, Coauthoring Behavior, and the Organization of Research in Drug Discovery. *Journal of Industrial Economics*, 46(2, Inside the Pin-Factory: Empirical Studies Augmented by Manager Interviews: A Symposium), 157-182.

Couger, J.D. & Zawacki, R.A., 1980. *Motivating and managing computer personnel*, New York: Wiley.

Cusumano, M. et al., 2003. Software Development Worldwide: The State of the Practice. *IEEE Softw.*, 20(6), 28-34.

Cusumano, M. & Selby, R., 1995. *Microsoft secrets: how the world's most powerful software company creates technology, shapes markets and manages people*, New York, NY: Free Press.

Cusumano, M.A., 1991a. *Japan's software factories : a challenge to U.S. management*, New York: Oxford University Press.

Cusumano, M.A., 1991b. *Japan's software factories : a challenge to U.S. management*, New York: Oxford University Press.

Cusumano, M.A., 1997. How Microsoft makes large teams work like small teams. *Sloan Management Review*, 39(1), 9-20.

Cusumano, M.A. & Yoffie, D.B., 1998. *Competing on Internet time: lessons from Netscape and its battle with Microsoft*, New York: The Free Press.

Cusumano, M.A., 2004. *The Business of Software: What Every Manager, Programmer,* *and Entrepreneur Must Know to Thrive and Survive in Good Times and Bad*, New York: Free Press.

Dasgupta, P. & David, P.A., 1994. Towards a new economics of science. *Research Policy*, 23, 487-524.

Ding, W.W. et al., 2010. The Impact of Information Technology on Academic Scientists' Productivity and Collaboration Patterns. *MANAGEMENT SCIENCE*, mnsc.1100.1195.

Hohmann, L., 1997. *Journey of the Software Professional: The Sociology of Software Development*, Upton Saddle River, NJ: Prentice Hall.

Holmstrom, B., 1982. Moral Hazard in Teams. *The Bell Journal of Economics*, 13(2), 324-340.

Hong, L. & Page, S.E., 2001. Problem Solving by Heterogeneous Agents. *Journal of Economic Theory*, 97(1), 123-163.

King, M. et al., 2010. *Integrated Public Use Microdata Series, Current Population Survey: Version 3.0*, University of Minnesota. Available at: http://cps.ipums.org/cps.

Lakhani, K.R. & Wolf, R., 2005. Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects. In *Perspectives on Free and Open Source Software*. Cambridge, MA: MIT Press.

Lakhani, K.R. & von Hippel, E., 2003. How Open Source Software Works: Free User to User Assistance. *Research Policy*, 32(6), 923-943.

Lazear, E., 2000. Performance pay and productivity. *The American Economic Review*.

Lerner, J. & Tirole, J., 2002. Some simple economics of open source. *The Journal of Industrial Economics*, 50(2), 197-234.

Lerner, J. & Tirole, J., 2002. Some Simple Economics of Open Source. *Journal of Industrial Economics*, 50(2), 197-234.

Manso, G., 2009. Motivating Innovation.

Mowery, D.C., 1996. *The international computer software industry : a comparative study of industry evolution and structure*, New York: Oxford University Press.

National Science Foundation, 2010. *National Patterns of R&D Resources: 2008 Data Update*, Arlington, VA: National Science Foundation. Available at: http://www.nsf.gov/statistics/nsf10314/pdf/nsf10314.pdf.

Pisano, G.P., 2006. *Science business : the promise, the reality, and the future of biotech*, Boston, Mass.: Harvard Business School Press.

Radner, R., 1993. The Organization of Decentralized Information Processing. *Econometrica: Journal of the Econometric Society*, 61(5), 1109-1146.

Roach, M. & Sauermann, H., 2010. A taste for science? PhD scientists' academic orientation and self-selection into research careers in industry. *Research Policy*.

Salop, J. & Salop, S., 1976. Self-selection and turnover in the labor market. *The Quarterly Journal of Economics*.

Salus, P.H., 1994. *A Quarter Century of UNIX*, Reading, MA: Addison-Wesley.

Schneiderman, B., 1980. *Software Psychology: Human Factors in Computer and Information Systems*, Scott Foresman & Co.

Shapin, S., 2008. *The scientific life: a moral history of a late modern vocation*, Chicago, IL: University of Chicago Press.

Shapiro, J.S. & David, P.A., 2008. Community-Based Production of Open Source Software: What Do We Know About the Developers Who Participate? *SSRN eLibrary*. Available at: http://papers.ssrn.com/sol3/papers.cfm?abstract_id=1286273 [Accessed September 3, 2010].

Sharp, H. et al., 2009. Models of motivation in software engineering. *Information and Software.*

Sinofsky, S. & Iansiti, M., 2010. *One strategy! : organization, planning, and decision making*, Hoboken, N.J.: Wiley.

Steinmueller, W.E., 1996. The U.S. Software Industry: An Analysis and Interpretive History. In David C. Mowery, ed. *The International Computer Software Industry: A Comparative Study of Industry Evolution and Structure*. New York: Oxford University Press.

Stephan, P., 1996. The Economics of Science. *Journal of Economic Literature*, 34(3), 1199-1235.

Stern, S., 2004. Do scientists pay to be scientists? *Management Science*, 50(6), 835-854.

Weinberg, G.M., 1971. *The psychology of computer programming*, New York,: Van Nostrand Reinhold.

8 Tables

Table 1 Number of Employees and Establishments in the Software, Pharma and Contract Research Industries (2007)

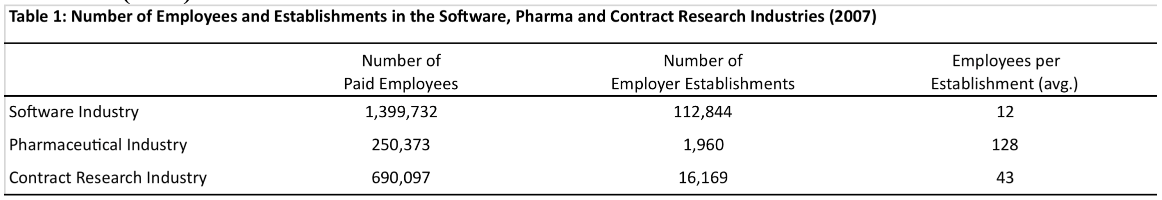


Table 1: Number of Employees and Establishments in the Software, Pharma and Contract Research Industries (2007)

	Number of Paid Employees	Number of Employer Establishments	Employees per Establishment (avg.)
Software Industry	1,399,732	112,844	12
Pharmaceutical Industry	250,373	1,960	128
Contract Research Industry	690,097	16,169	43

Source: U.S. Bureau of the Census - 2007 Economic Census

Table 2 Education Levels for Software Developers and Scientists

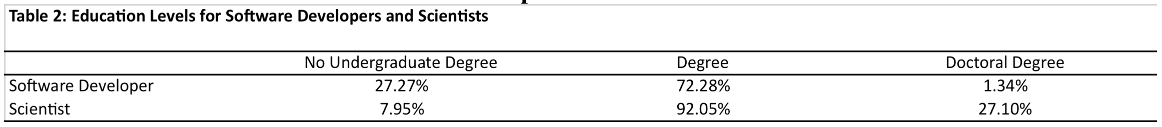
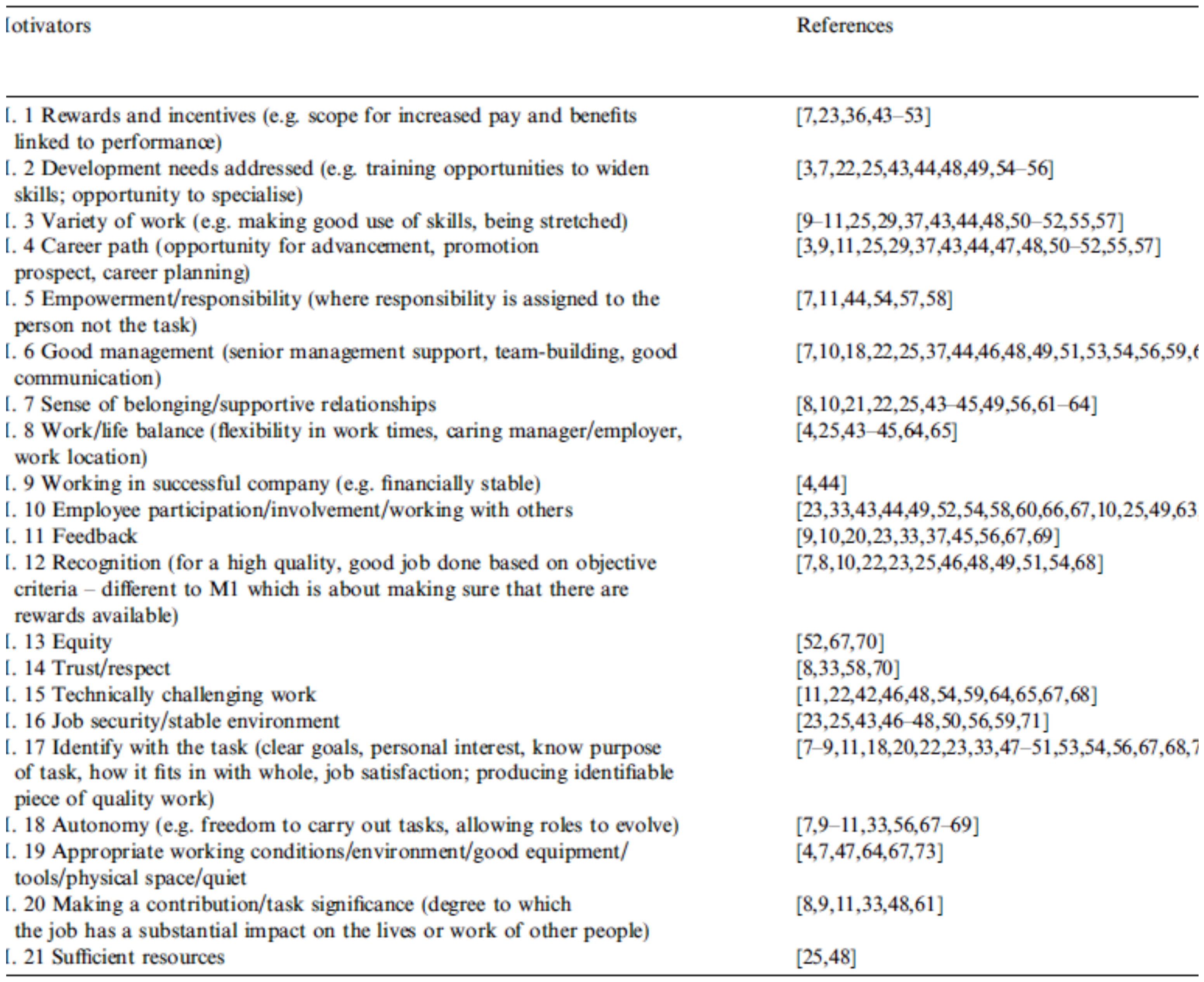


Table 2: Education Levels for Software Developers and Scientists

	No Undergraduate Degree	Degree	Doctoral Degree
Software Developer	27.27%	72.28%	1.34%
Scientist	7.95%	92.05%	27.10%

Source: Miriam King, Steven Ruggles, J. Trent Alexander, Sarah Flood, Katie Genadek, Matthew B. Schroeder, Brandon Trampe, and Rebecca Vick. Integrated Public Use Microdata Series, Current Population Survey: Version 3.0. [Machine-readable database]. Minneapolis: University of Minnesota, 2010. http://cps.ipums.org/cps

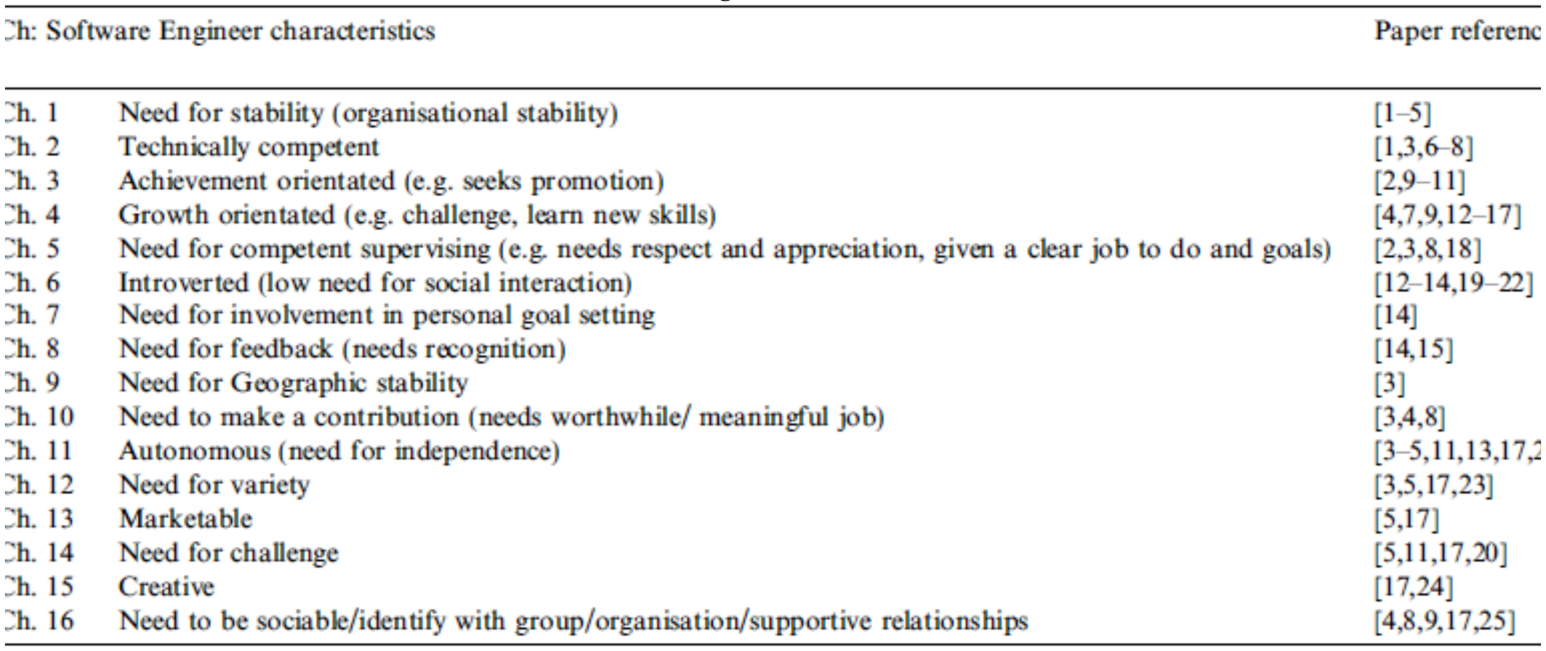
Table 3 List of Motivators Described in the Literature



[otivators	References
[. 1 Rewards and incentives (e.g. scope for increased pay and benefits linked to performance)	[7,23,36,43–53]
[. 2 Development needs addressed (e.g. training opportunities to widen skills; opportunity to specialise)	[3,7,22,25,43,44,48,49,54–56]
[. 3 Variety of work (e.g. making good use of skills, being stretched)	[9–11,25,29,37,43,44,48,50–52,55,57]
[. 4 Career path (opportunity for advancement, promotion prospect, career planning)	[3,9,11,25,29,37,43,44,47,48,50–52,55,57]
[. 5 Empowerment/responsibility (where responsibility is assigned to the person not the task)	[7,11,44,54,57,58]
[. 6 Good management (senior management support, team-building, good communication)	[7,10,18,22,25,37,44,46,48,49,51,53,54,56,59,6
[. 7 Sense of belonging/supportive relationships	[8,10,21,22,25,43–45,49,56,61–64]
[. 8 Work/life balance (flexibility in work times, caring manager/employer, work location)	[4,25,43–45,64,65]
[. 9 Working in successful company (e.g. financially stable)	[4,44]
[. 10 Employee participation/involvement/working with others	[23,33,43,44,49,52,54,58,60,66,67,10,25,49,63
[. 11 Feedback	[9,10,20,23,33,37,45,56,67,69]
[. 12 Recognition (for a high quality, good job done based on objective criteria – different to M1 which is about making sure that there are rewards available)	[7,8,10,22,23,25,46,48,49,51,54,68]
[. 13 Equity	[52,67,70]
[. 14 Trust/respect	[8,33,58,70]
[. 15 Technically challenging work	[11,22,42,46,48,54,59,64,65,67,68]
[. 16 Job security/stable environment	[23,25,43,46–48,50,56,59,71]
[. 17 Identify with the task (clear goals, personal interest, know purpose of task, how it fits in with whole, job satisfaction; producing identifiable piece of quality work)	[7–9,11,18,20,22,23,33,47–51,53,54,56,67,68,7
[. 18 Autonomy (e.g. freedom to carry out tasks, allowing roles to evolve)	[7,9–11,33,56,67–69]
[. 19 Appropriate working conditions/environment/good equipment/ tools/physical space/quiet	[4,7,47,64,67,73]
[. 20 Making a contribution/task significance (degree to which the job has a substantial impact on the lives or work of other people)	[8,9,11,33,48,61]
[. 21 Sufficient resources	[25,48]

Source: Beecham et al. 2008, pg 868

Table 4 List of Characteristics of Software Engineers Described in the Literature



Ch: Software Engineer characteristics		Paper referenc
Ch. 1	Need for stability (organisational stability)	[1–5]
Ch. 2	Technically competent	[1,3,6–8]
Ch. 3	Achievement orientated (e.g. seeks promotion)	[2,9–11]
Ch. 4	Growth orientated (e.g. challenge, learn new skills)	[4,7,9,12–17]
Ch. 5	Need for competent supervising (e.g. needs respect and appreciation, given a clear job to do and goals)	[2,3,8,18]
Ch. 6	Introverted (low need for social interaction)	[12–14,19–22]
Ch. 7	Need for involvement in personal goal setting	[14]
Ch. 8	Need for feedback (needs recognition)	[14,15]
Ch. 9	Need for Geographic stability	[3]
Ch. 10	Need to make a contribution (needs worthwhile/ meaningful job)	[3,4,8]
Ch. 11	Autonomous (need for independence)	[3–5,11,13,17,2
Ch. 12	Need for variety	[3,5,17,23]
Ch. 13	Marketable	[5,17]
Ch. 14	Need for challenge	[5,11,17,20]
Ch. 15	Creative	[17,24]
Ch. 16	Need to be sociable/identify with group/organisation/supportive relationships	[4,8,9,17,25]

Source: Beecham et al., pg 867

Table 5 Key Features of the Experimental Regimes

	Competition Regime	Cooperation Regime
Group Size	20 competitors	Four five-person teams (assigned)
Payoffs	Total: $1,000, divided five ways $500 – best submission $200 – second best submission $125 – third best submission $100 – fourth best submission $75 – fifth best submission	Total: $1,000, divided five ways $1,000 – divided among winning team members (according to average of team members' suggestions)
Communications and Code Sharing	None	Private team message board and ability to send directed messages
Information	Competitors "see" who is in the group, their ratings, and top code submissions to date.	Competitors "see" other teams' best scores to date and their own team members' detailed information and statistics and background.

Table 6 Sample Breakdown

All Participants	Competition	Cooperation	
Randomly-Assigned	259	266	525
Self-Selection	257	258	515
	516	524	1040

Competed for Cash Prize	Competition	Cooperation	
Randomly-Assigned	112	117	229
Self-Selection	116	120	236
	228	237	465

No Cash Prize	Competition	Cooperation	
Randomly-Assigned	147	149	296
Self-Selection	141	138	279
	288	287	575

Table 7 Variable Definitions

	Variable	Definition
(1)	*HoursWorked*	Number of hours worked by an individual participant during the course of the experiment
(2)	*NumSubmissions*	Number of solutions submitted to be compiled, tested and scored by an individual participant during the course of the experiment
(3)	*SelfSelection*	Indicator switched to one for participants who were asked their preferences regarding the regimes and subsequently assigned to their preferred regime
(4)	*Competition*	Indicator switched to one for a participants in the competitive regime; zero for participants in the cooperative regime
(5)	*Prize*	Indicator switched to one for participants within a group of 20 that competed for a $1000 cash prize
(6)	*SkillRating*	Measure of general problem solving ability in Algorithmic problems based on historical performance on TopCoder platform
(7)	*LikertScale*	1-4 integer response indicating relative preference for competitive regime, where higher scores indicate greater relative preference for competitive regime over cooperative regime
(8)	*OrderofQuestion*	Indicators switched to one when questionnaire to elicit preferences was designed to present competitive regime first and cooperative regime second

Table 8 Descriptive Statistics

Variable	Num. Obs.	Mean	Std. Dev.	Min	Mean	(1)	(2)	(3)	(4)	(5)	(6)	(7)
(1) *HoursWorked*	1040	10.55	18.75	0	190							
(2) *NumSubmissions*	1040	2.50	5.63	0	42	.57						
(3) *SelfSelection*	1040	.50	.50	0	1	.15	.05					
(4) *Competition*	1040	.50	.50	0	1	.01	.07	.01				
(5) *Prize*	1040	.45	.50	0	1	.25	.20	.01	-.03			
(6) *SkillRating*	1040	1250.72	538.39	188	3797	.03	.20	.01	.18	.07		
(7) *LikertScale*	515	3.30	1.38	1	5	.07	.17	n/a	.88	-.03	.18	
(8) *OrderofQuestion*	515	.49	.50	0	1	.03	.01	n/a	.07	.00	-.01	.10

Table 9 Comparison of Mean Effort Levels, Cross-Tabulations

COMPETITION REGIME

RANDOMLY-ASSIGNED PARTICIPANTS				SELF-SELECTED PARTICIPANTS			
Variable	Mean	Std. Error of Est. Mean	Population Std. Dev.	Variable	Mean	Std. Error of Est. Mean	Population Std. Dev.
HoursWorked	6.60	.84	13.46	HoursWorked	14.92	1.53	24.99
I[HoursWorked > 1}	.29	.03	.45	*I[HoursWorked > 1}*	.48	.03	.50
NumSubmissions	1.98	.31	5.00	*NumSubmissions*	3.77	.44	7.22

COOPERATIVE REGIME

RANDOMLY-ASSIGNED PARTICIPANTS				SELF-SELECTED PARTICIPANTS			
Variable	Mean	Std. Error of Est. Mean	Population Std. Dev.	Variable	Mean	Std. Error of Est. Mean	Population Std. Dev.
HoursWorked	8.97	.98	15.70	*HoursWorked*	11.57	1.10	17.61
I[HoursWorked > 1}	.32	.03	.47	*I[HoursWorked > 1}*	.34	.03	.48
NumSubmissions	2.44	.35	5.53	*NumSubmissions*	1.78	.25	4.07

Table 10 Regression Results of Tests for Randomization and Assignment Procedures

	Depenedent Variable = *HoursWorked*							
	Competitive Regime					Cooperative Regime		
Model:	1	2	3	4	5	6	7	8
Explanatory Variables	Two-Way Correlation	Linear Sklls Control	Skills-Level Dummies	Ordered Pair Differences	Prize Control	Two-Way Correlation	Ordered Pair Differences	Prize Control
SelfSelection	8.33***	8.33***	8.36***	8.71***	8.32***	2.60*	2.50*	2.48*
	(1.75)	(1.75)	(1.76)	(1.79)	(1.71)	(1.47)	(1.43)	(1.40)
Prize					9.14***			9.88***
					(1.85)			(1.48)
SkillRating		-1.09	-4.87		-3.60			2.01
		(1.59)	(4.30)		(4.19)			(4.22)
Skills Dummies			Yes		Yes			Yes
Constant	6.60***	8.07***				8.97***		
	(.84)	(2.28)				(.98)		
R-Squared	.04	.04	.05	.55	.09	.04	.55	.09

Notes. *, **, and *** indicate statistical significance at the 10%, 5% and 1% levels, respectively; heteroskedasticity robust standard errors reported; number of observations = 516 participants in the cooperative regime; 524 in the cooperative regime.

Table 11 Results of Test for Presence of Hawthorne Effects

	Dependent Variable = *LikertResponse*		Dependent Variable = *HoursWorked*	
Model:	1	2	3	4
Explanatory Variables	No Controls	Controls	OLS	IV
SelfSelection			3.72	-.05
			(2.61)	(4.40)
Prize		-.10	9.39***	9.43***
		(.12)	(2.77)	(2.79)
SkillRating		.37	3.47	3.79
		(.37)	(8.03)	(8.20)
Skills Dummies		Yes	Yes	Yes
QuestionOrder	0.27**	0.28**		
	(.12)	(.12)		
R-Squared	.01	.05		

Notes. *, **, and *** indicate statistical significance at the 10%, 5% and 1% levels, respectively; heteroskedasticity robust standard errors reported; number of observations = 156 participants.

Table 12 Results of Tests for Presence of Within-Group Interactions Among Effort Choices

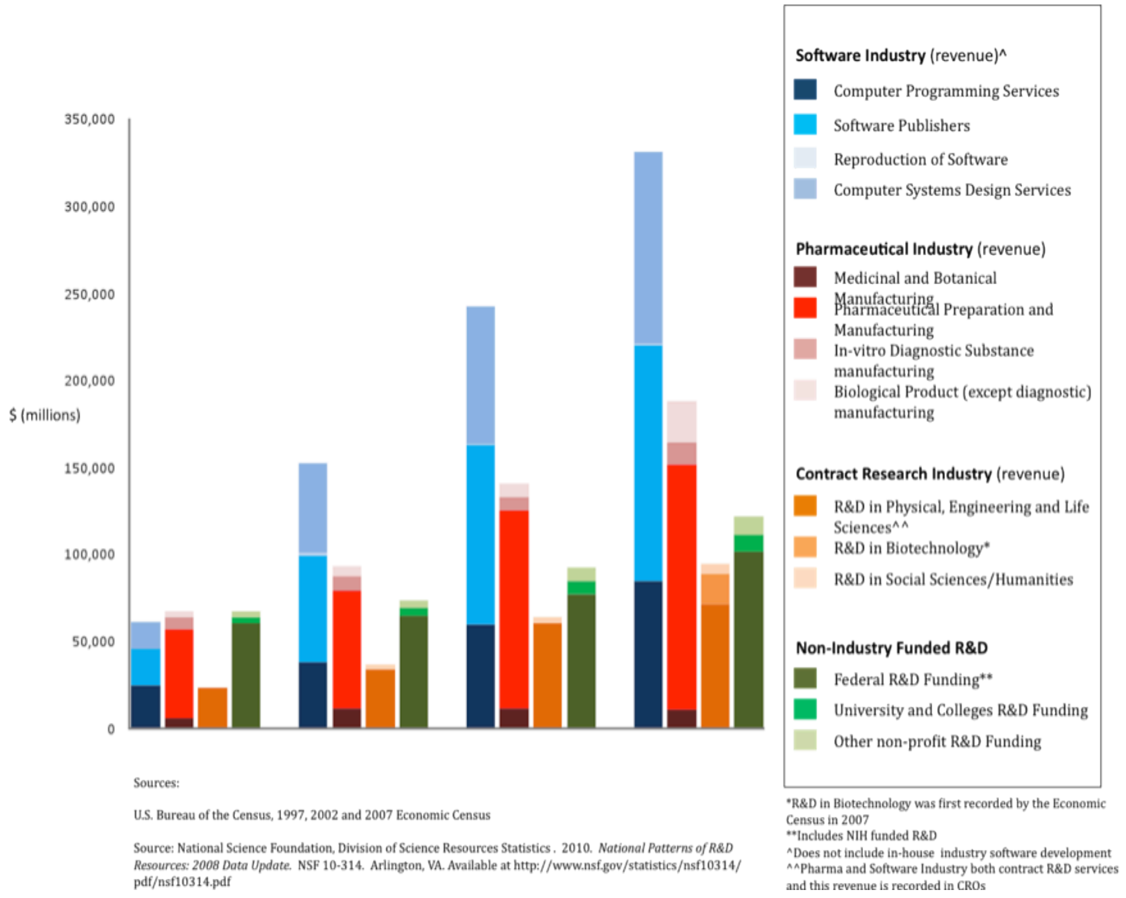
	Depenedent Variable = *HoursWorked*						
	Competitive Regime			Cooperative Regime			
Model:	1	2	3	4	5	6	7
Explanatory Variables	All Competitive	Self-Selected	Randomly Assigned	All Cooperative	All Cooperative	Self-Selected	Randomly Assigned
SelfSelection	7.43***			1.30	1.69		
	(2.25)			(1.30)	(1.34)		
Prize	8.16***	9.93**	6.56***	5.13***	6.69***	8.75***	4.95**
	(2.69)	(4.75)	(2.28)	(1.94)	(1.53)	(2.14)	(2.15)
HoursWorkedGroup	.11	.11	.06	.21			
	(.18)	(.24)	(.26)	(.19)			
HoursWorkedTeam				0.26***	0.32***	0.34***	0.23**
				(.09)	(.09)	(.12)	(.11)
SkillRating	-3.46	1.59	-8.46*	2.76	2.54	5.78	-.69
	(4.18)	(7.23)	(4.36)	(4.12)	(4.12)	(6.08)	(5.53)
Skills Dummies	Yes	Yes	Yes	Yes	Yes	Yes	Yes
R-Squared	.09	.06	.10	.15	.15	.20	.10

Notes. *, **, and *** indicate statistical significance at the 10%, 5% and 1% levels, respectively; heteroskedasticity robust standard errors reported; number of observations = 516 participants in the cooperative regime; 524 in the cooperative regime.

Table 13 Re-Estimated Effects with Reweighted Skills and Synthesized Control Group

To be inserted

9 Figures



Sources:

U.S. Bureau of the Census, 1997, 2002 and 2007 Economic Census

Source: National Science Foundation, Division of Science Resources Statistics . 2010. *National Patterns of R&D Resources: 2008 Data Update.* NSF 10-314. Arlington, VA. Available at http://www.nsf.gov/statistics/nsf10314/pdf/nsf10314.pdf

*R&D in Biotechnology was first recorded by the Economic Census in 2007
**Includes NIH funded R&D
^Does not include in-house industry software development
^^Pharma and Software Industry both contract R&D services and this revenue is recorded in CROs

Figure 1 Comparison of Revenues in the Software Industry to those of Other Sectors

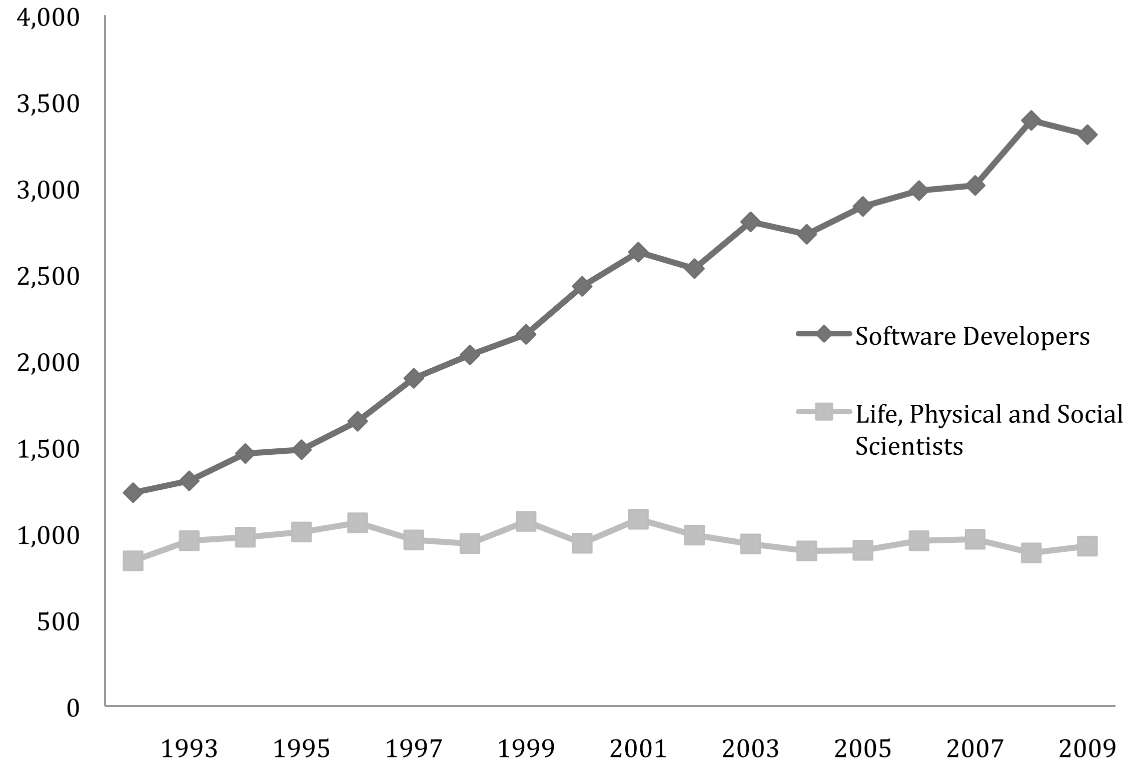


Figure 2 1992-2009: Total US Software vs. Sciences Employment (Thousands)

SOURCE: Miriam King, Steven Ruggles, J. Trent Alexander, Sarah Flood, Katie Genadek, Matthew B. Schroeder, Brandon Trampe, and Rebecca Vick. Integrated Public Use Microdata Series, Current Population Survey: Version 3.0. [Machine-readable database]. Minneapolis: University of Minnesota, 2010. http://cps.ipums.org/cps/citation.shtml

Self-Selection (Treatment Groups)	A’	B’
Random Assignment (Control Groups)	A	B
	Cooperative Regime	Competitive Regime
	Experimental Regimes	

Figure 3 Comparison of Self-Selection into Regimes versus Random-Assignment

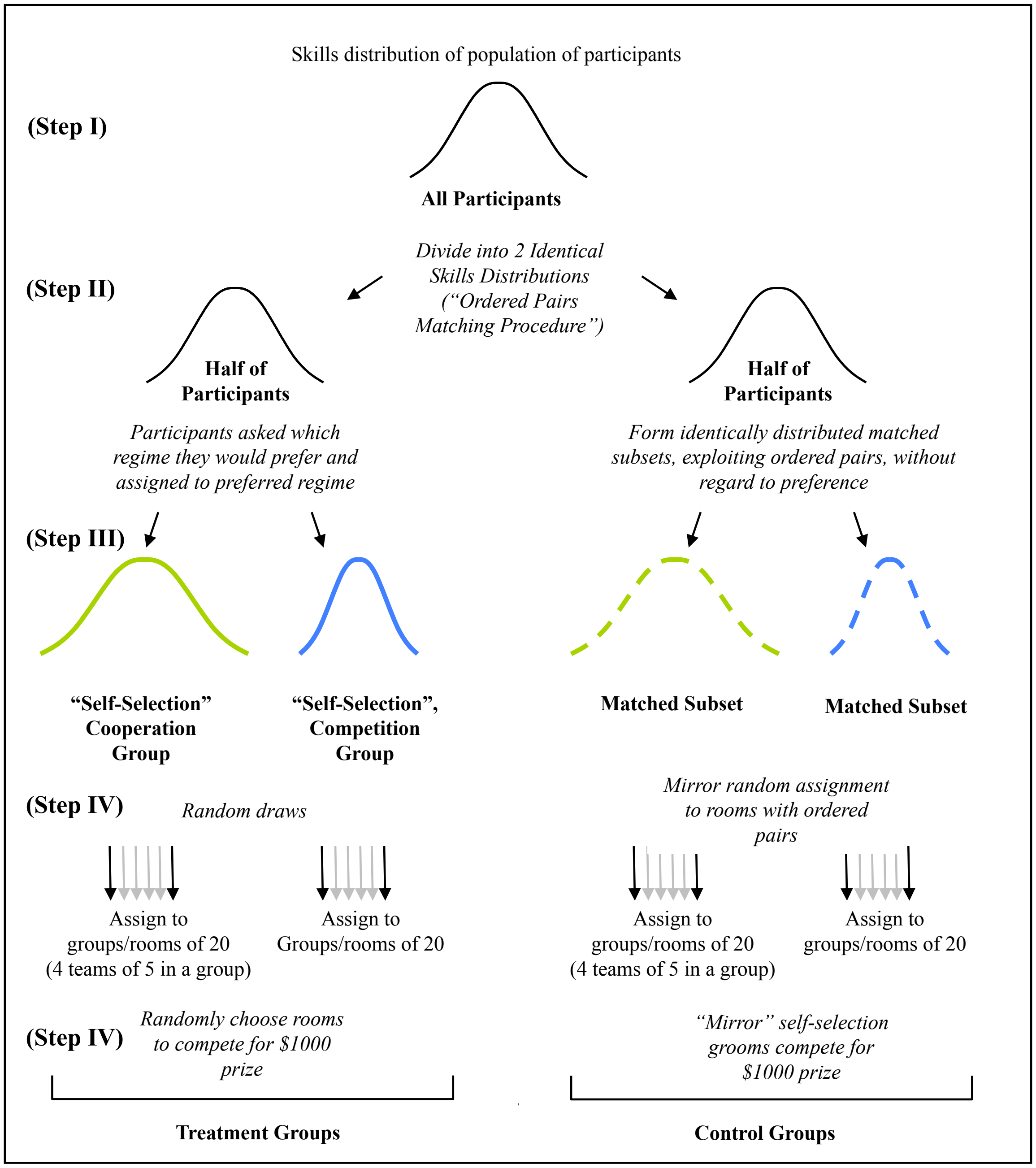

Skills distribution of population of participants
(Step I)
All Participants
(Step II)
Divide into 2 Identical Skills Distributions ("Ordered Pairs Matching Procedure")
Half of Participants
Half of Participants
Participants asked which regime they would prefer and assigned to preferred regime
Form identically distributed matched subsets, exploiting ordered pairs, without regard to preference
(Step III)
"Self-Selection" Cooperation Group
"Self-Selection", Competition Group
Matched Subset
Matched Subset
(Step IV)
Random draws
Mirror random assignment to rooms with ordered pairs
Assign to groups/rooms of 20 (4 teams of 5 in a group)
Assign to Groups/rooms of 20
Assign to groups/rooms of 20 (4 teams of 5 in a group)
Assign to groups/rooms of 20
(Step IV)
Randomly choose rooms to compete for $1000 prize
"Mirror" self-selection grooms compete for $1000 prize
Treatment Groups
Control Groups

Figure 4 Illustration of Assignment Procedure

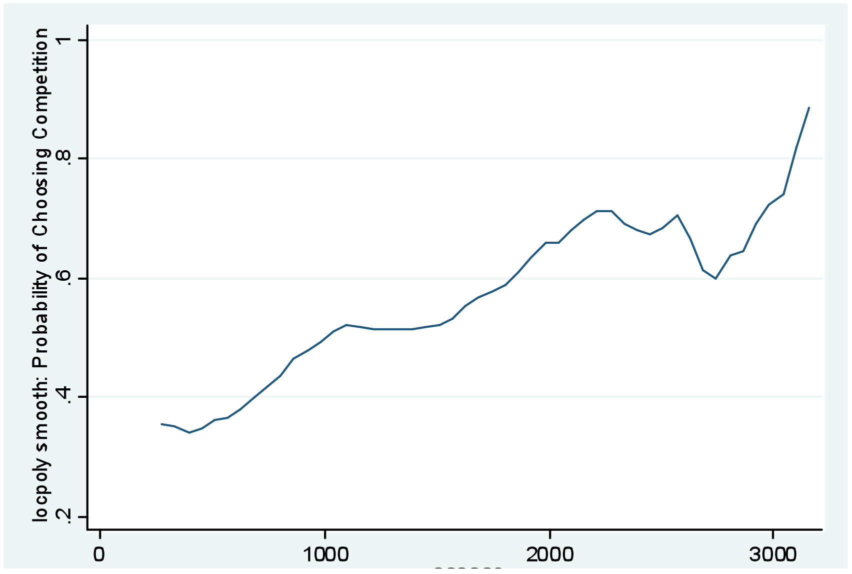


Figure 5 Preferences for Competition versus *SkillRating*

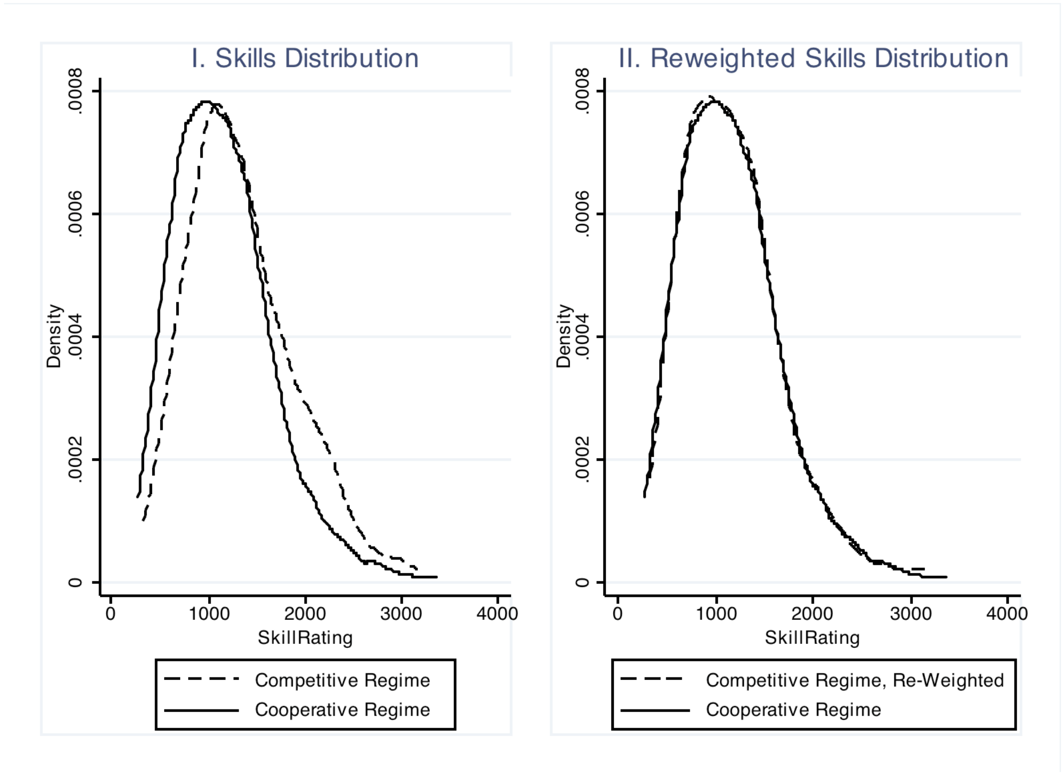


Figure 6 Skills Distribution in Cooperative and Competitive Regimes

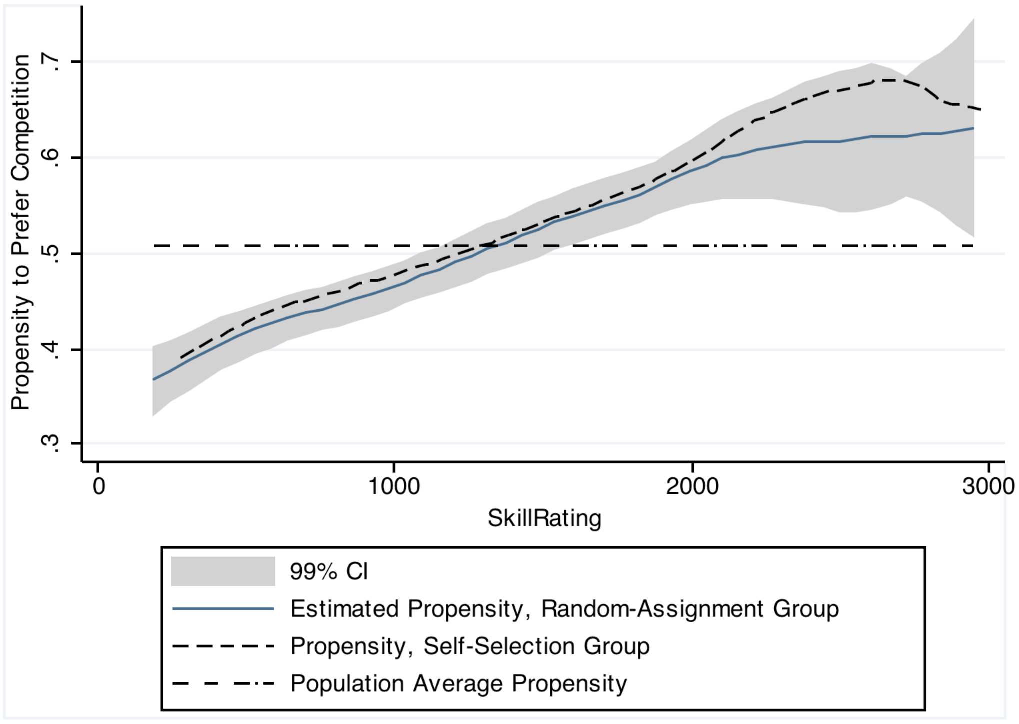


Figure 7 Propensity to Compete Across Different Levels versus the Overall Population Average

To be inserted

Figure 8 Distribution of Effort Choices for Self-Selected and Randomly-Assigned Participants (Reweighted to Match Skills Distribution and with Skills-Neutral Control Group)