Screening for Engineering Talent

Gábor Nyéki*

[Note: Preliminary and incomplete. Do not circulate.]

Abstract. Software firms hire engineers whose latent skills affect the quality of firm output. Hiring high-skill engineers is difficult because accurate screening technologies are costly. However, if high-skill engineers learn difficult programming languages more easily than low-skill engineers, then teams within a firm could recruit them by adopting a difficult language. I build a signaling model of hiring and production, and show that in a separating equilibrium, (i) difficult languages tend to be chosen by smaller teams, (ii) the self-selection of high-skill engineers may lead a team to choose a difficult language even if it is a worse fit for its problem domain, and (iii) if engineer skill strongly affects team output, then larger teams may choose an easier language even if the difficult language is a better fit because low-skill engineers are more profitable to hire. I show evidence for the model in two data sets. First, using data from the question-and-answer platform *Stack Overflow,* interest in C++, a difficult language, predicts better answers about other languages and interest in transferable skills like engineering practices, mathematics, and statistics. Second, looking at a near-universe of U.S. job postings between 2010 and 2024 from *Lightcast,* C++ has more relative adoption in technical problem domains such as securities trading, biomedical engineering, artificial intelligence, and financial technology, than mainstream languages like Java or C#, and niche languages like Scala, F#, and Clojure.

Date: September 10, 2024.
*Princeton University; e-mail: gnyeki@princeton.edu.

"...if a company chooses to write its software in a comparatively esoteric language, they'll be able to hire better programmers, because they'll attract only those who cared enough to learn it."

—Paul Graham, "The Python Paradox"

1. Introduction

While worker ability plays a crucial role in economic development as firm production becomes more complex (Kremer, 1993), frictions in the education system and the labor market result in misallocation. Such misallocation of talent has large consequences for economic growth (Murphy et al., 1991; Hsieh et al., 2019). Mirroring these frictions, firms report making significant efforts to select high-ability job applicants, especially for jobs where monitoring individual performance is harder (Huang and Cappelli, 2010). In this environment, high-ability workers and firms would both benefit from lower-cost screening technologies.

This paper studies screening in the labor market for software engineers. Software engineers perform complex cognitive work, and they typically do so in teams, where their individual performance is difficult to measure. As a result, higher-paying U.S. software firms are known for conducting over a half dozen rounds of job interviews, with "behavioral" as well as high-pressure "coding" interviews that aim to assess applicants' knowledge of algorithms and problem solving skills. While the costs of these search procedures are high, in this setting a firm's technology choice, namely the programming language that it chooses, can serve as an alternative or complementary screening technology.

This idea that language choice acts as a screening technology is motivated by anecdotal evidence in the software industry. Early internet entrepreneur and later venture capitalist Paul Graham (2004) famously described the "Python paradox": the apparent superiority of engineers who learned Python, an unpopular language at the time. While Python has been widely adopted in industry and education since then, it was niche in 2004 and Google, a pioneering start-up, was one of its early adopters. Although the existence of the Python paradox has been a matter of debate, similar views have been voiced by others over the years. To formalize this idea, I propose a signaling model of labor-market screening in which latent engineer ability enters the production function multiplicatively, as in Kremer's (1993) O-ring theory. Firms can choose between a mainstream and a niche programming language, the former of which is easier to use for engineers. I show numerically that a pooling equilibrium is always possible, but a separating equilibrium is only possible if the teams that would choose the niche language are smaller.[1] Moreover, if engineers' latent ability affects productivity more strongly, then a smaller team may choose the niche language even if it is a worse fit for the team's problem domain than the mainstream language.

I assess evidence for the screening role of programming languages in two data sources. First, I use data from the question-and-answer site *Stack Overflow,* and I look at (a) whether knowledge of niche languages predicts higher-quality answers in mainstream languages for the same individual, and (b) whether knowledge of niche languages predicts activity on associated question-and-answer sites that specialize in software engineering, mathematics, and statistics and machine learning. Second, I use *Lightcast's* near-universe of

[1] More precisely, the parameter region in which a separating equilibrium is possible shrinks with team size. Thus if teams are larger, then all teams choose the mainstream language in a pooling equilibrium.

U.S. job postings between 2010 and 2024, and look at whether firms post more jobs for niche languages in more technical problem domains than in web services (a less technical domain). Specifically, I estimate a measure that I call *relative adoption:* the difference in the niche language's adoption rate in a technical domain and in web services, compared with a mainstream language's adoption rate difference in the same pair of domains.

I find that the knowledge of an additional programming language almost always predicts higher-quality answers in unrelated mainstream languages, and more engagement on Stack Overflow's associated sites that cover transferable technical skills. However, the knowledge of a niche language such as Scala, Clojure, or F# *instead of* the knowledge of a mainstream language such as Java or C# does not predict higher-quality answers, nor does it in general predict more engagement on associated sites. The exception is C++, a mainstream language with great cognitive complexity.[2] Knowledge of C++ predicts both higher-quality answers in unrelated mainstream languages and interest in transferable technical skills.

The job postings data mirrors the special status of C++. While none of the niche languages studied in this paper see a greater relative adoption in more technical problem domains, C++ sees much more use in these domains. Relative adoption estimates show that its use is skewed more towards securities trading, biomedical engineering, and artificial intelligence, than the use of Java or C# is.

This paper contributes to the literature on labor-market screening for high-skill workers. The idea that education can serve as a signal of latent ability has been well-known in economics (Spence, 1973, 2002). Many studies since then have explored optimal payment schemes that induce high-type workers to self-select (e.g., Delfgaauw and Dur, 2007). By contrast, I consider an endogenous aspect of the production technology, the programming language used by a software firm, that attracts high-type workers. Studies of high-skill workers have shown that "talent" and resources are misallocated. Exposure to peer inventors (Akcigit et al., 2017; Bell et al., 2018) and peer mathematicians (Agarwal and Gaule, 2020), and access to resources, are likely to be critical in high-skill workers achieving high productivity.

The paper continues as follows. Section 2 describes computer scientists' views on programming languages and compares some of the major languages in use today. Section 3 introduces a signaling model of language choice. Section 4 presents results on predictors of engineer skill in the Stack Overflow data, and Section 5 on job postings by problem domain in the Lightcast data. Section 6 concludes.

2. Background

This section provides an overview of the roles that computer scientists attribute to programming languages, and describes some of the languages that are in common use today.

2.1. Mainstream and niche languages. A programming language serves multiple purposes. Perhaps the most obvious one is that it encodes algorithms unambiguously so that computers can execute them. However, it is also a "tool of thought" (Iverson, 1980), a precise notation that enables engineers to reason about the correctness and efficiency of algorithms and to communicate algorithms to each other. Every programming language makes a trade-off between the efficiency of executed code, the ease of writing, understanding, and

[2]The complexity of C++, a language that has existed and has been evolving since the 1980s, can be attributed to its strong backward compatibility guarantees. The first version of C++ predates nearly all of the other languages discussed in this paper, including Scala, Clojure, F#, Java, C#, Python, and JavaScript.

extending code, and the ability to ensure the correctness of algorithms by finding or fully preventing certain kinds of mistakes (Horowitz, 1983).[3]

By the early 1970s, navigating this complicated trade-off had already given rise to dozens if not hundreds of programming languages (Landin, 1966; Sammet, 1972). Since then, many more experimental languages have been created as testbeds of novel concepts, both by academics and by engineers working in industry. Ultimately, those concepts that proved to be practical in experimental languages have slowly made their way into the mainstream.

Mainstream languages differ from experimental ones in that they prioritize familiarity.[4] Familiarity is especially important for larger organizations with long-running projects, where new and often early-career engineers must be able to understand and contribute to existing code bases (Pike, 2012).[5] The long-term maintainability that a familiar language supports is so important that focus on it has been described as one of the characteristics that distinguish "software engineering" from "computer programming" (Winters et al., 2020).

The most widely familiar languages are those that engineers see in their early education. Over the past couple decades, Java and Python have become by far the most widely known languages, and the majority of computer science graduates can be expected to have a working proficiency of both. A 2014 survey of computer science programs at 39 U.S. universities found that 27 used Python and 22 used Java in its introductory courses (Guo, 2014). Almost a third of the surveyed departments also used C or C++, two other mainstream general-purpose programming languages.

Jane Street, a proprietary trading firm, has been a prominent user of OCaml, and its head of technology has co-authored a book on the language. The book describes OCaml's strengths compared with mainstream languages, and writes that when mainstream languages adopt such features, they usually do so in a limited form:

> *"Despite their importance, these ideas have made only limited inroads into mainstream languages, and when they do arrive there, like first-class functions in C# or parametric polymorphism in Java, it's typically in a limited and awkward form. The only languages that completely embody these ideas are statically typed, functional programming languages like OCaml, F#, Haskell, Scala, Rust, and Standard ML." (Madhavapeddy and Minsky, 2022)*

2.2. Platforms that host multiple languages. Java and C# are notable in that they represent not only two mainstream languages but also their associated "runtimes." These runtimes serve two purposes: they enable portability across operating systems and computer architectures, and they hide certain technical details, e.g., memory management, from software engineers. However, since both runtimes are relatively

[3] Horowitz (1983, p. 34) writes: "Programmers must design and implement a software system. They must document it for themselves and for others. They must produce a product which is free of errors and which performs efficiently. […] Thus the programmer's tasks put demands on a language which are at the same time diverse and mutually antagonistic. This makes the design of a programming language an exciting challenge and a field where only a select few succeed."

[4] Hoare (1974) emphasizes the importance of complete understanding: "A necessary condition […] is the utmost simplicity in the design of the language. […] A programmer who fully understands his language can tackle more complex tasks, and complete them more quickly and more satisfactorily, than if he did not. In fact, a programmer's need for an understanding of his language is so great that it is almost impossible to persuade him to change to a new one."

[5] Pike (2012), one of the designers of Google's Go programming language, writes: "Programmers working at Google are early in their careers and are most familiar with procedural languages, particularly from the C family. The need to get programmers productive quickly in a new language means that the language cannot be too radical."

language-agnostic, they can also accommodate alternative programming languages. This has opened up the possibility for software modules that were written in different languages, but for the same runtime, to interoperate with each other.

Several niche languages were born and have gained popularity due to interoperability with the existing ecosystem of one of the two mainstream languages. Today, Java's Java Virtual Machine hosts Scala, Clojure, and Kotlin, and C#'s Common Language Runtime[6] hosts F#, among other languages.

Scala was originally developed as an academic research and teaching language, aiming to address the shortcomings of Java and C# (Odersky et al., 2006). Within the decade after its first public release in 2004, it saw adoption among technology and finance firms (Odersky and Rompf, 2014).[7] Clojure, taking its inspiration from a well-known niche language called Common Lisp, was released in 2007 as an alternative to C++, Java, and C# in enterprise software. The language was subsequently adopted mainly for web services, data analysis, and financial technology applications (Hickey, 2020).

F# began as an attempt at Microsoft Research to adapt OCaml, a niche language mentioned earlier, to C#'s runtime. Its first major version was released in 2005. F# saw early interest among finance firms, and due to the inherent characteristics of the language and Microsoft's change in business strategy, cloud computing became an area of focus around 2012 (Syme, 2020).

In a 2019 Stack Overflow survey cited by Hickey (2020), engineers who used Clojure or F# were among the most experienced and highest earning respondents.

2.3. Screening technologies. As proposed by Paul Graham (2004), niche programming languages may serve as valuable screening technology in the labor market. The trade-offs between mainstream and niche languages have been the subject of much discussion in the software industry. On one hand, high-skill engineers may be disproportionately more likely to apply for jobs that use niche languages. On the other hand, niche languages reduce the firm's applicant pool, making it harder to recruit engineers. While there are proponents of both views, at the least one team at Facebook which uses a niche language called Haskell has reported that this trade-off has been favorable to them in hiring (Dreimanis, 2023).

3. Theoretical Framework

This section proposes a signaling model of programming language choice. The key predictions of this model will be tested in Sections 4 and 5.

3.1. Engineer ability. Engineer i has latent ability $\theta_i \in \Theta := (0, 1)$. She takes a job at a firm that works with programming language $\ell \in \mathcal{L}$ and pays a wage $w(\ell)$. This yields her a utility $u(w(\ell), \theta_i) := w(\ell) - \gamma_\ell/\theta_i$. She chooses among her possible employers so as to maximize this utility.

The parameter $\gamma_\ell > 0$ reflects the objective difficulty of learning and using language ℓ. The ratio γ_ℓ/θ_i is the subjective difficulty from the perspective of engineer i. An engineer with higher latent ability finds the same language less difficult, i.e., $u(w(\ell), \theta_i) > u(w(\ell), \theta_j)$ for all $\theta_i > \theta_j$.

[6] C#'s runtime was originally called the Common Language Runtime. It has been renamed to ".NET runtime" since.

[7] As an example, most of Twitter's code base has been written in Scala. Although the firm initially used Ruby, a popular language for web development, Ruby's slow execution, high memory use, and lack of static typing made it difficult for Twitter's web services to scale as its user base grew and its code base became more complex (Venners, 2009).

3.2. Firm profit. Each firm in the economy operates in a specific problem domain. The representative firm in domain $\delta \in \Delta$ chooses a language ℓ. Its realized profit is

$$\Pi(\ell, \delta) := p(\delta) y(\ell, \delta) - N(\delta) w(\ell)$$

where $y(\ell, \delta)$ is the firm's output, $p(\delta)$ is the market price of the firm's output, $N(\delta)$ is the number of engineers hired by firms in domain δ, and $w(\ell)$ is the wage that it pays its engineers if it chooses language ℓ.

The firm's output can be decomposed in terms of the latent abilities of the engineers that it hires as

$$y(\ell, \delta) := \varphi_\ell(\delta) \left[N(\delta) \left((1 - \alpha(\delta)) + \alpha(\delta) \prod_i \theta_i \right) \right]^{\beta(\delta)}.$$

In this formula, $\varphi_\ell(\delta) > 0$ is language ℓ's "fitness" in domain δ,[8] $\alpha(\delta) \in [0, 1]$ determines how reliant the firm is on high-ability engineers, and $\beta(\delta) \in (0, 1)$ is the labor share. The product term $\prod_i \theta_i$ reflects complementarities in engineer skill, as in Kremer's (1993) O-ring theory of the effect of worker skill on output quality and economic development.

The firm observes the output price $p(\delta)$, the language fitness $\varphi_\ell(\delta)$, and the market wage $w(\ell)$, but it does not observe the ability of its engineers. Thus it maximizes *expected* profit:

$$\mathbb{E}(\Pi(\ell, \delta) \mid \ell) = p(\delta) \varphi_\ell(\delta) \mathbb{E} \left\{ \left[N(\delta) \left((1 - \alpha(\delta)) + \alpha(\delta) \prod_i \theta_i \right) \right]^{\beta(\delta)} \,\middle|\, \ell \right\} - N(\delta) w(\ell).$$

In the remainder of the paper, I suppress the dependence of model parameters on δ to reduce the notational burden.

The role of α can be understood by considering the firm's optimal choice at the extreme values that this parameter can take. At its minimum, $\alpha = 0$, which implies that engineer skill is irrelevant to the firm. In this case, the expectation operator can be dropped, and the firm chooses the language that maximizes the difference $\varphi_\ell - w(\ell) N^{1-\beta}/p$ between language fitness and a normalized form of market wage. At its maximum, $\alpha = 1$, the firm is completely reliant on the ability of its engineers. Consequently, it might choose a language ℓ that has a lower value for $\varphi_\ell - w(\ell) N^{1-\beta}/p$ than other languages. Although such a choice may seem suboptimal to an outside observer who assumes that $\alpha \ll 1$, it would in fact be optimal for the firm because the language serves as a screening technology to recruit high-skill engineers.

The firm's optimal language choice fundamentally depends on the domain within which it operates. Languages can be ranked by their fitness parameter φ_ℓ. Depending on the firm's domain, the fitness parameters, and thus the ranking of languages, can change. For example, due to its ecosystem and it being designed for statistical analysis, R is a much better fit for data science applications than C++. At the same time, C++ is a much better fit than R for applications that require speed and memory efficiency.

[8] "Fitness" reflects the effect that a language has on firm productivity via (i) its library ecosystem and (ii) its inherent characteristics. Examples of (i), such as R's CRAN, Python's PyPI, Stata's SSC, and TeX's CTAN are well-known to economists. In the software industry, the recognition that code reuse improves productivity drove the 1990s popularization of both object-oriented programming and package ecosystems such as the Perl language's CPAN. Two examples of (ii) are automatic memory management which was introduced into mainstream software development by Java in 1996 (originally invented for the Lisp language in 1959), and strong static typing which has made it easier to manage large code bases, e.g., in web development when TypeScript was released in 2012.

3.3. Mechanisms. While the output, $y(\ell, \delta)$, takes a reduced-form approach to modeling the mechanisms that drive the firm's language choice, it is worth enumerating the mechanisms at play. From the firm's perspective, the engineer's latent ability is salient because it affects its output via three mechanisms:

1. Defect rate: engineers with higher latent ability can write program code that is more comprehensible to other engineers and more likely to be correct.

2. Problem solving: engineers with higher latent ability are able to solve problems more quickly and are able to find better solutions.

3. Task prioritization: engineers with higher latent ability are able to set priorities that better align with the firm's strategic goal of delivering output and staying competitive.

3.4. Equilibrium. The firm takes as given the language fitness and market wage of each language, $(\varphi_\ell(\delta), w(\ell))_\ell$. It also takes as given the number of engineers it hires, N, and the skill-reliance parameter, α. The only variable that it chooses is the language ℓ with which it maximizes its expected profit.

Engineer i takes as given the market price for the firm's output, $p(\delta)$, and the market wage for each language, $(w(\ell))_\ell$. She chooses the firm at which to accept a job which determines the language ℓ that she uses.

The labor market is assumed to be perfectly competitive. Thus, the market wage is equal to engineers' expected marginal revenue product of labor.

There are two types of possible equilibria: separating equilibria in which engineers sort into languages by latent ability and pooling equilibria in which they don't.

To illustrate firm and worker behavior in a separating equilibrium, I make two simplifying assumptions. First, there are two ability levels, high and low, so that the ability space is $\Theta := \{\theta_H, \theta_L\}$ where $0 < \theta_L < \theta_H < 1$. Second, there are two languages that firms and engineers can choose from, mainstream and niche, with the corresponding language space $\mathcal{L} := \{\ell_M, \ell_N\}$.

3.4.1. The engineer's choice. Recall that engineer i's utility is given by the function $u(w(\ell), \theta_i) := w(\ell) - \gamma_\ell/\theta_i$ where γ_ℓ/θ_i is the subjective difficulty of language ℓ. She decides whether to take a job that uses the niche language by evaluating the inequality $u(w(\ell_N), \theta_i) > u(w(\ell_M), \theta_i)$. In a separating equilibrium, where the niche language signals high ability, high-skill engineers prefer the niche language and low-skill engineers prefer the mainstream one:

$$\begin{aligned} u(w(\ell_M), \theta_H) = w(\ell_M) - \frac{\gamma_{\ell_M}}{\theta_H} < w(\ell_N) - \frac{\gamma_{\ell_N}}{\theta_H} = u(w(\ell_N), \theta_H), \\ u(w(\ell_M), \theta_L) = w(\ell_M) - \frac{\gamma_{\ell_M}}{\theta_L} \geq w(\ell_N) - \frac{\gamma_{\ell_N}}{\theta_L} = u(w(\ell_N), \theta_L).. \end{aligned}$$

These inequalities imply

$$\begin{aligned} \frac{\gamma_{\ell_N} - \gamma_{\ell_M}}{\theta_H} < w(\ell_N) - w(\ell_M) \text{ and} \\ \frac{\gamma_{\ell_N} - \gamma_{\ell_M}}{\theta_L} \geq w(\ell_N) - w(\ell_M), \end{aligned}$$

and if the niche language pays a higher wage than the mainstream language, then

$$0 < \theta_L \leq \frac{\gamma_{\ell_N} - \gamma_{\ell_M}}{w(\ell_N) - w(\ell_M)} < \theta_H.$$

3.4.2. The firm's choice. The firm chooses the niche language if and only if its expected profit is greater with the niche than with the mainstream language, $\mathbb{E}(\Pi(\ell_N) \mid \ell_N) > \mathbb{E}(\Pi(\ell_M) \mid \ell_M)$. In a separating equilibrium, all high-ability engineers choose the niche language and all low-ability engineers choose the mainstream language, so the expectation operator can be dropped.

The labor market is characterized by perfect competition. Thus, the market wage is equal to engineers' individual marginal revenue product of labor.

4. Niche Languages and Engineer Skills

Within the theoretical framework of Section 3, engineer i is said to have ability θ_i which determines her productivity, and which firms thus care about. However, rather than directly observing θ_i, firms only observe i's knowledge of programming languages. In essence, they are interested in the difference

$$\Pr(\theta_i = \theta_H \mid x_{i\ell} = 1) - \Pr(\theta_i = \theta_H \mid x_{i\ell} = 0) \tag{1}$$

where $x_{i\ell} = 1$ if and only if i has worked with language ℓ.

For the purpose of screening workers, firms do not need to identify the causal effect of $x_{i\ell}$ on θ_i, merely its usefulness as a predictor. Thus the goal of this section is to assess how experience with niche languages predicts skills in other areas of software engineering.

4.1. Specification. Let i index individuals who answered questions about a mainstream language, and let j index such questions. On Stack Overflow, both questions and answers are voted on by users and assigned (possibly negative) scores based on those votes. As empirical counterparts to (1), one can think of

$$\begin{aligned} &\mathbb{E}\left(\text{CppAnswerScore}_{ij} \;\middle|\; \text{KnowsScala}_i = 1, \text{KnowsJava}_i = 1\right) - \\ &\quad - \mathbb{E}\left(\text{CppAnswerScore}_{ij} \;\middle|\; \text{KnowsScala}_i = 0, \text{KnowsJava}_i = 1\right) \end{aligned} \tag{2}$$

which captures how knowledge of Scala (a niche language that is interoperable with Java, itself a mainstream language) predicts i's ability at C++ (a mainstream language that is unrelated to Java), and

$$\begin{aligned} &\mathbb{E}\left(\text{JavaAnswerScore}_{ij} \mid \text{KnowsScala}_i = 1, \text{KnowsCpp}_i = 1\right) - \\ &\quad - \mathbb{E}\left(\text{JavaAnswerScore}_{ij} \mid \text{KnowsScala}_i = 0, \text{KnowsCpp}_i = 1\right) \end{aligned} \tag{3}$$

which captures how knowledge of Scala, in addition to knowledge of C++, predicts i's ability at Java.

To operationalize this, I denote the score assigned to question j by QuestionScore_j and the score assigned

to individual i's answer by AnswerScore_{ij}. The main specification that I fit to the data is

$$\begin{aligned}\text{AnswerScore}_{ij} = \beta_0 + \beta_N \text{Niche}_i + \beta_M \text{Mainstream}_i + \\ + \beta_{NM} \text{Niche}_i \text{Mainstream}_i + \mu_{y(j)} + \sigma_{\text{QuestionScore}_j} + \varepsilon_{ij}\end{aligned} \tag{4}$$

where AnswerScore_{ij} is the score of an answer about a mainstream language (C++, Java, or C#), Niche_i is a binary indicator for whether individual i has also posted at least one answer about a niche language (e.g., Scala, F#), Mainstream_i is a binary indicator for whether i has also posted at least one answer about a mainstream language that is different from the language that question j is about (e.g., Python), $\mu_{y(j)}$ is a fixed effect for the year in which question j was asked, $\sigma_{\text{QuestionScore}_j}$ is a fixed effect for the question's score, and ε_{ij} is an exogenous residual term. The fixed effects account for two factors that affect AnswerScore_{ij}, independently of the quality of the answer itself: the age and the popularity of the question that is being answered. Questions that are older or more popular receive more views. Thus answers to such questions have a chance to accumulate more votes.

I test two null hypotheses in equation (4). As an example, let's take Scala as the niche language, Java as the mainstream language, and C++ as the language that the answer is about. Then the two null hypotheses and their interpretation in the context of the theoretical framework are:

H_A : $\beta_N = \beta_M$. Knowing Scala but not Java predicts the same C++ ability as knowing Java but not Scala. This means that Scala and Java have the same signaling value to firms.

H_B : $\beta_N + \beta_{NM} = 0$. Knowing *both* Scala and Java predicts the same C++ ability as knowing only Java. This means that Scala has no signaling value beyond that which Java has.

In addition to skills in specific programming languages, firms could also value an engineer's broader interest in software engineering and other intellectual topics. Stack Overflow is associated with a dozens of sibling sites that specialize in other topics, such as software engineering, mathematics, and statistical theory, under the umbrella *Stack Exchange*. I assess whether knowledge of niche languages predicts such interests by fitting

$$\mathbf{1}_{\{\text{Reputation}_i > 200\}} = \beta_0 + \beta_N \text{Niche}_i + \beta_M \text{Mainstream}_i + \beta_{NM} \text{Niche}_i \text{Mainstream}_i + \mu_{y_0(i)} + \varepsilon_i \tag{5}$$

where $\mu_{y_0(i)}$ is a fixed effect for the first year in which individual i posted an answer on Stack Overflow and ε_i is an exogenous residual term. The outcome variable is binary and defined in terms of Reputation_i which is i's "reputation" on a specific Stack Exchange site. Reputation is determined by i's total score on answers and comments on that site. I test the same null hypotheses for equation (5) as I do for (4).

4.2. Results. The p-values in Table 1 for the null hypothesis that $\beta_N = \beta_M$ show that knowledge of Scala and F#, in Panels A and B, do not predict higher-quality answers about unrelated mainstream languages when compared with Java or C#. At the same time, Panel C shows that knowledge of C++ predicts higher-quality answers for Python when compared with knowledge of Java.

However, the coefficient estimates in Table 2 and the p-values for the same null hypothesis show that knowledge of C++ generally predicts more interest in transferable technical skills such as software engineering practices, mathematics, and statistics and machine learning.

5. The Language Choice of Firms

The theoretical framework of Section 3 predicts that difficult languages should see more adoption in skill-intensive problem domains than they do in non-skill-intensive ones. The goal of this section is to assess whether this prediction holds in U.S. job postings data between 2010 and 2024.

5.1. Specification. Let $\text{Postings}_{cm\ell y}$ be the number of job postings made by firm c in metropolitan area m mentioning language ℓ in year y. I use ordinary least squares to fit the specification

$$\ln \text{Postings}_{cm\ell y} = \alpha_\ell + \sum_{d \in \text{Domains}} \beta_{\ell d} \mathbf{1}_{\{\text{Domain}(c)=d\}} + \lambda_c + \mu_m + \theta_y + \varepsilon_{cm\ell y} \tag{6}$$

where $\text{Domains} := \{\text{Trading}, \text{Fintech}, \text{Biomed}, \text{Pharma}, \text{Manufacturing}, \text{SAAS}\}$.[9] Equation (6) excludes observations in years in which the company made zero postings for a given language. Thus I also use negative binomial regressions to fit

$$\mathbb{E}\left(\text{Postings}_{cm\ell y}\right) = \exp\left(\alpha_\ell + \sum_{d \in \text{Domains}} \beta_{\ell d} \mathbf{1}_{\{\text{Domain}(c)=d\}} + \lambda_c + \mu_m + \theta_y\right). \tag{7}$$

The null hypothesis that I test is constructed as follows. First, I look at the difference between a language ℓ's adoption in domain d and its adoption in SAAS: $\beta_{\ell d} - \beta_{\ell,\text{SAAS}}$. Then I compare this difference to the analogous difference for another language ℓ', by subtracting one from the other: $(\beta_{\ell d} - \beta_{\ell,\text{SAAS}}) - (\beta_{\ell' d} - \beta_{\ell',\text{SAAS}})$. I call this double difference a *relative adoption* rate of language ℓ, and denote it by $\Delta(\ell, \ell', d)$.

5.2. Results. Table 3 shows estimates of the relative adoption measure, $\Delta(\ell, \ell', d)$, for C++, a difficult mainstream language, and Scala and F#, two niche languages that are interoperable with the mainstream languages Java and C#, respectively. Panel A shows that C++ is more used in securities trading, biomedical engineering, and AI applications, than Java or C#. This result is robust to estimation method and whether the complete set of fixed effects is included in the specification.

Panel B shows that none of the three niche languages see more relative adoption in these problem domains. Rather, Scala is *less* used in securities trading and biomedical engineering than Java, while there is no statistically significant difference in adoption for AI applications. Similarly, Clojure and F# are less used in securities trading and AI than Java and C#, and have no statistically significant difference at the 5-percent level in biomedical engineering.

These results parallel the results of Section 4: niche languages that, according to the Python paradox, should be favored for highly technical problem domains are in fact less widely used in these domains. Scala, Clojure, and F# all aim for pragmatism and a good "developer experience." At the same time, both Scala and F# incorporate niche but, by today, well-established advances in programming language theory. Their lack of adoption in technical domains could be explained either by the relative sparsity of their library ecosystems, the lack of a sufficiently large productivity advantage, or the lack of self-selection into their use by high-ability engineers. C++, due to its long lifespan and backward compatibility guarantees, has a worse

[9] The acronym SAAS stands for "software as a service," i.e., the delivery of software as a web service rather than a program that the user needs to install. This problem domain has become very popular in the 2010s.

developer experience and a much greater complexity than Scala, Clojure, or F#. Yet, its often lamented complexity may make it a credible signal of latent engineer ability.

6. Conclusion

References

Agarwal, R. and P. Gaule (2020): "Invisible Geniuses: Could the Knowledge Frontier Advance Faster?" *American Economic Review: Insights*, 2, 409–24.

Akcigit, U., J. Grigsby, and T. Nicholas (2017): "Immigration and the Rise of American Ingenuity," *American Economic Review: Papers & Proceedings*, 107, 327–31.

Bell, A., R. Chetty, X. Jaravel, N. Petkova, and J. Van Reenen (2018): "Who Becomes an Inventor in America? The Importance of Exposure to Innovation," *The Quarterly Journal of Economics*, 134, 647–713.

Delfgaauw, J. and R. Dur (2007): "Signaling and screening of workers' motivation," *Journal of Economic Behavior & Organization*, 62, 605–624.

Dreimanis, G. (2023): "Haskell in Production: Meta," https://web.archive.org/web/20231202092825/https://serokell.io/blog/haskell-in-production-meta.

Graham, P. (2004): "The Python Paradox," https://web.archive.org/web/20240515221423/https://paulgraham.com/pypar.html.

Guo, P. (2014): "Python Is Now the Most Popular Introductory Teaching Language at Top U.S. Universities," https://web.archive.org/web/20240616180637/https://cacm.acm.org/blogcacm/python-is-now-the-most-popular-introductory-teaching-language-at-top-u-s-universities/.

Hickey, R. (2020): "A history of Clojure," *Proc. ACM Program. Lang.*, 4.

Hoare, C. A. R. (1974): "Hints on Programming Language Design," in *Computer Systems Reliability*, ed. by C. Bunyan, vol. 20, 505–534.

Horowitz, E. (1983): *Fundamentals of Programming Languages*, Rockville, Md.: Computer Science Press.

Hsieh, C.-T., E. Hurst, C. I. Jones, and P. J. Klenow (2019): "The Allocation of Talent and U.S. Economic Growth," *Econometrica*, 87, 1439–1474.

Huang, F. and P. Cappelli (2010): "Applicant Screening and Performance-Related Outcomes," *American Economic Review: Papers & Proceedings*, 100, 214–18.

Iverson, K. E. (1980): "Notation as a Tool of Thought," *Commun. ACM*, 23, 444–465.

Kremer, M. (1993): "The O-Ring Theory of Economic Development," *The Quarterly Journal of Economics*, 108, 551–575.

Landin, P. J. (1966): "The Next 700 Programming Languages," *Commun. ACM*, 9, 157–166.

Madhavapeddy, A. and Y. Minsky (2022): *Real World OCaml: Functional Programming for the Masses*, Cambridge University Press, 2 ed.

Murphy, K. M., A. Shleifer, and R. W. Vishny (1991): "The Allocation of Talent: Implications for Growth," *The Quarterly Journal of Economics*, 106, 503–530.

Odersky, M., P. Altherr, V. Cremet, I. Dragos, G. Dubochet, B. Emir, S. McDirmid, S. Micheloud, N. Mihaylov, M. Schinz, E. Stenman, L. Spoon, and M. Zenger (2006): "An Overview of the Scala Programming Language," technical report.

Odersky, M. and T. Rompf (2014): "Unifying functional and object-oriented programming with Scala," *Commun. ACM*, 57, 76–86.

Pike, R. (2012): "Go at Google: Language Design in the Service of Software Engineering," https://talks.golang.org/2012/splash.article.

Sammet, J. E. (1972): "Programming Languages: History and Future," *Commun. ACM*, 15, 601–610.

Spence, M. (1973): "Job Market Signaling," *The Quarterly Journal of Economics*, 87, 355–374.

——— (2002): "Signaling in Retrospect and the Informational Structure of Markets," *American Economic Review*, 92, 434–459.

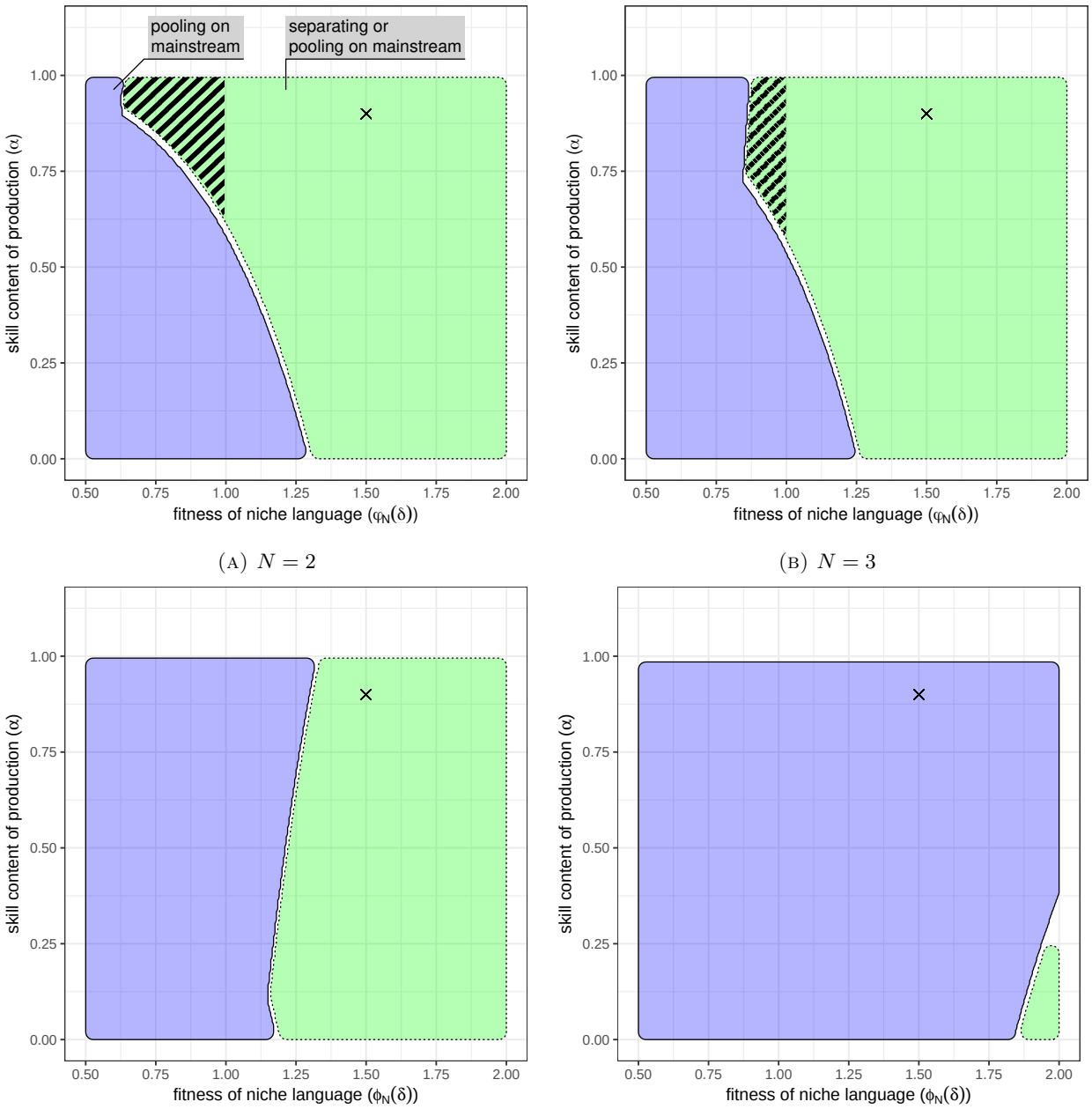
Syme, D. (2020): "The early history of F#," *Proc. ACM Program. Lang.*, 4.

Venners, B. (2009): "Twitter on Scala: A Conversation with Steve Jenson, Alex Payne, and Robey Pointer," https://web.archive.org/web/20240325081304/https://www.artima.com/articles/twitter-on-scala.

Winters, T., T. Manshreck, and H. Wright (2020): *Software Engineering at Google: Lessons Learned from Programming over Time*, O'Reilly Media.

A. Pooling Equilibria

FIGURE 1. Parameter regions with pooling and separating equilibria

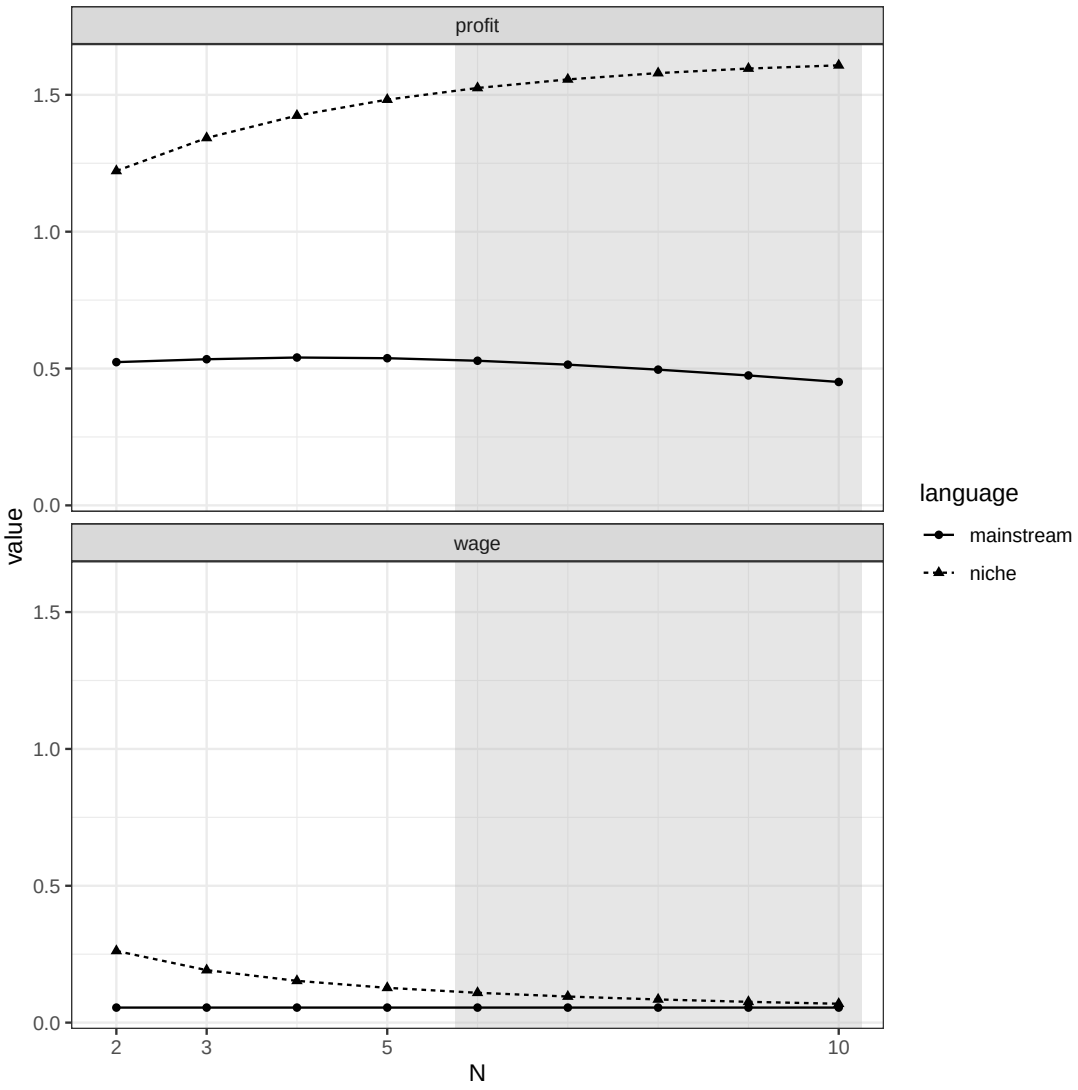


(A) $N = 2$ (B) $N = 3$

(C) $N = 5$ (D) $N = 10$

Notes: This figure shows that a separating equilibrium in which the niche language is used only exists if firms are smaller. Panel (a) shows equilibria for a firm size of $N = 2$, and panels (b)–(d) show equilibria for larger firm sizes. In the blue regions, only a pooling equilibrium exists, with all firms choosing the mainstream language. In the green regions, a separating equilibrium is also possible. In the shaded regions, a separating equilibrium is possible even though the niche language has lower fitness than the mainstream language. Profits and wages that correspond to the points marked by "×" are shown in Figure 2. The complete list of model parameters is shown in Table 4.

FIGURE 2. Profits and wages in candidate separating equilibria



Notes: This figure shows that in a candidate separating equilibrium, profit increases but the wage premium for high-skill engineers decreases with firm size. In the region shown in gray, no separating equilibrium exists because the wage premium is too low and high-skill engineers prefer the mainstream language. The profits and wages for $N = 2$, 3, 5, and 10 correspond to the points marked by "$\times$" in Figure 1, i.e., they are obtained under $\varphi_N(a) = 1.5$ and $\alpha = 0.9$. The complete list of model parameters is shown in Table 4.

TABLE 1. Answer scores and the knowledge of niche programming languages

Answer language	β_N	β_M	β_{NM}	Obs.	p-value for $\beta_N = \beta_M$	p-value for $\beta_N + \beta_{NM} = 0$
Panel A: Scala as niche language, Java as mainstream language						
C++	1.196**	1.722***	−0.696	1,268,013	0.313	0.012
	(0.5179)	(0.0861)	(0.5596)			
C#	1.746**	1.553***	0.001	2,463,895	0.779	0.026
	(0.6839)	(0.0848)	(1.0323)			
Python	0.677***	1.777***	0.346	2,859,907	0.000	0.000
	(0.1724)	(0.1464)	(0.3405)			
Panel B: F# as niche language, C# as mainstream language						
C++	1.711***	1.758***	−0.705	1,268,013	0.931	0.000
	(0.5371)	(0.1103)	(0.5928)			
Java	3.014***	1.730***	−1.167	2,927,074	0.132	0.032
	(0.8497)	(0.0869)	(1.2027)			
Python	3.316***	2.143***	−0.608	2,859,907	0.345	0.001
	(1.2327)	(0.1708)	(1.4620)			
Panel C: C++ as "niche" language, Java as mainstream language						
JavaScript	0.685***	0.958***	1.038***	3,840,883	0.025	0.000
	(0.1108)	(0.0613)	(0.1897)			
.NET	1.675***	1.442***	0.430	572,977	0.225	0.000
	(0.1809)	(0.1153)	(0.3282)			
Python	1.256***	0.930***	0.830***	2,859,907	0.038	0.000
	(0.1387)	(0.1082)	(0.2414)			
Panel D: C++ as "niche" language, C# as mainstream language						
JavaScript	0.984***	1.238***	0.650***	3,840,883	0.132	0.000
	(0.1603)	(0.0652)	(0.2268)			
.NET	0.724***	0.910***	1.958***	572,977	0.484	0.000
	(0.2798)	(0.1218)	(0.3856)			
Python	1.452***	1.307***	0.520*	2,859,907	0.448	0.000
	(0.1112)	(0.1677)	(0.2822)			

Notes: Rows correspond to estimates of equation (4). Standard errors are clustered by individual. Significance codes: * $p < 0.1$, ** $p < 0.05$, *** $p < 0.01$.

TABLE 2. Activity on Stack Exchange sites and the knowledge of niche programming languages

Site	β_N	β_M	β_{NM}	Obs.	p-value for $\beta_N = \beta_M$	p-value for $\beta_N + \beta_{NM} = 0$
Panel A: Scala as niche language, Java as mainstream language						
Software Engineering	0.006***	0.009***	0.047***	1,955,191	0.000	0.000
	(0.0009)	(0.0002)	(0.0022)			
Code Review	0.004***	0.004***	0.025***	1,955,191	0.642	0.000
	(0.0008)	(0.0001)	(0.0017)			
Mathematics	0.007***	0.003***	0.014***	1,955,191	0.000	0.000
	(0.0010)	(0.0002)	(0.0017)			
Cross Validated[a]	0.006***	−0.001***	0.007***	1,955,191	0.000	0.000
	(0.0009)	(0.0001)	(0.0014)			
Panel B: F# as niche language, C# as mainstream language						
Software Engineering	0.024***	0.012***	0.143***	1,955,191	0.029	0.000
	(0.0053)	(0.0002)	(0.0087)			
Code Review	0.023***	0.006***	0.057***	1,955,191	0.000	0.000
	(0.0048)	(0.0002)	(0.0069)			
Mathematics	0.040***	0.001***	0.005	1,955,191	0.000	0.000
	(0.0062)	(0.0002)	(0.0074)			
Cross Validated[a]	0.009**	−0.002***	0.010**	1,955,191	0.002	0.000
	(0.0035)	(0.0001)	(0.0043)			
Panel C: Scala as niche language, C++ as mainstream language						
Software Engineering	0.015***	0.020***	0.121***	1,955,191	0.000	0.000
	(0.0009)	(0.0004)	(0.0054)			
Code Review	0.008***	0.013***	0.066***	1,955,191	0.000	0.000
	(0.0007)	(0.0003)	(0.0042)			
Mathematics	0.007***	0.020***	0.049***	1,955,191	0.000	0.000
	(0.0007)	(0.0004)	(0.0040)			
Cross Validated[a]	0.006***	0.005***	0.022***	1,955,191	0.118	0.000
	(0.0006)	(0.0002)	(0.0028)			
Panel D: F# as niche language, C++ as mainstream language						
Software Engineering	0.049***	0.021***	0.199***	1,955,191	0.000	0.000
	(0.0047)	(0.0004)	(0.0116)			
Code Review	0.035***	0.014***	0.074***	1,955,191	0.000	0.000
	(0.0038)	(0.0003)	(0.0087)			
Mathematics	0.022***	0.021***	0.041***	1,955,191	0.688	0.000
	(0.0032)	(0.0004)	(0.0074)			
Cross Validated[a]	0.008***	0.005***	0.014***	1,955,191	0.201	0.000
	(0.0022)	(0.0002)	(0.0047)			

[a] The Stack Exchange site called *Cross Validated* specializes in statistics and machine learning.
Notes: Rows correspond to estimates of equation (5). Standard errors are clustered by individual. Significance codes: * $p < 0.1$, ** $p < 0.05$, *** $p < 0.01$.

TABLE 3. Language popularity across problem domains

	OLS (1)	OLS (2)	Neg. binom. (3)	Neg. binom. (4)
Panel A: Difficult mainstream language				
Δ(Cpp, Java, Trading)	0.121***	0.105***	0.186***	0.155***
	[0.000]	[0.000]	[0.000]	[0.004]
Δ(Cpp, Java, Biomed)	0.045	0.058	0.207***	0.221***
	[0.273]	[0.151]	[0.003]	[0.001]
Δ(Cpp, Java, AI)	0.151***	0.146***	0.274***	0.260***
	[0.000]	[0.000]	[0.000]	[0.000]
Δ(Cpp, C#, Trading)	0.158***	0.159***	0.306***	0.280***
	[0.000]	[0.000]	[0.000]	[0.000]
Δ(Cpp, C#, Biomed)	0.163***	0.171***	0.326***	0.333***
	[0.000]	[0.000]	[0.000]	[0.000]
Δ(Cpp, C#, AI)	0.177***	0.182***	0.350***	0.331***
	[0.000]	[0.000]	[0.000]	[0.000]
Panel B: Niche languages				
Δ(Scala, Java, Trading)	−0.086**	−0.114***	−0.187***	−0.213***
	[0.042]	[0.008]	[0.005]	[0.001]
Δ(Scala, Java, Biomed)	−0.224**	−0.239***	−0.298***	−0.334***
	[0.011]	[0.005]	[0.005]	[0.003]
Δ(Scala, Java, AI)	0.005	−0.005	−0.047	−0.034
	[0.863]	[0.839]	[0.371]	[0.517]
Δ(Clojure, Java, Trading)	−0.243***	−0.263***	−0.439***	−0.482***
	[0.006]	[0.001]	[0.005]	[0.001]
Δ(Clojure, Java, Biomed)	−0.580***	−0.548***	−0.595*	−0.666*
	[0.003]	[0.007]	[0.055]	[0.062]
Δ(Clojure, Java, AI)	−0.181**	−0.239***	−0.328**	−0.380**
	[0.021]	[0.003]	[0.022]	[0.011]
Δ(F#, C#, Trading)	−0.244*	−0.306**	−0.428**	−0.527***
	[0.078]	[0.028]	[0.029]	[0.002]
Δ(F#, C#, Biomed)	−0.150	−0.118	0.011	0.007
	[0.517]	[0.566]	[0.975]	[0.982]
Δ(F#, C#, AI)	−0.402***	−0.496***	−0.942***	−1.044***
	[0.000]	[0.000]	[0.000]	[0.000]
Observations	2,674,811	2,674,811	3,422,550	3,422,550
Firm FE	YES	YES	YES	YES
Year FE		YES		YES
MSA FE		YES		YES

Notes: These hypothesis tests use estimates of equations (6) and (7). $\Delta(\ell, \ell', d)$ is defined as $(\beta_{\ell d} - \beta_{\ell,\text{SAAS}}) - (\beta_{\ell' d} - \beta_{\ell',\text{SAAS}})$ and reflects whether language ℓ sees more relative adoption in domain d than language ℓ'. Square brackets show p-values based on standard errors two-way clustered by firm and MSA. Significance codes: * $p < 0.1$, ** $p < 0.05$, *** $p < 0.01$.

TABLE 4. Model parameters

Parameter	Value		Description
	$\delta = a$	$\delta = b$	
$p(\delta)$	1	1	Price of output good
$\beta(\delta)$	3/10	3/10	Labor share
$\varphi_M(\delta)$	1	1	Fitness of the mainstream language
$\varphi_N(\delta)$	*varies*	1	Fitness of the niche language
$\alpha(\delta)$	*varies*	1/4	Skill content of production
$N(\delta)$	*varies*	10	Firm size
γ_M	1		Difficulty of the mainstream language
γ_N	1.05		Difficulty of the niche language
λ_H	1/2		Population share of high-skill engineers
θ_H	9/10		Latent ability of high-skill engineers
θ_L	1/10		Latent ability of low-skill engineers

Notes: This table shows the parameter values that were used to generate Figures 1 and 2. The economy consists of two domains, a and b. The figures show how equilibria and equilibrium outcomes change with $\varphi_N(a)$, $\alpha(a)$, and $N(a)$, holding all other parameters fixed.