CASE: E-768(A)
DATE: 12/1/21

BYTEBOARD: REINVENTING THE TECHNICAL INTERVIEW (A)

The incubator gave teams six months to get a proof of concept in the hands of customers. We were halfway through the program and had yet to write a single line of code. We needed to get some amount of data to validate that we were building something that had legs.

—Nikke Hardson-Hurley, Cofounder and Chief Product Officer, Byteboard[1]

INTRODUCTION

Everyone else filed out of the conference room at Google's San Francisco office, leaving the Byteboard team alone to debrief. They had just completed a midway check-in with investors and were feeling increased pressure to launch a product and begin collecting feedback.[2] With three months left in the incubator program, the clock was ticking.

Three months earlier, in May 2018, Byteboard cofounders Nikke Hardson-Hurley and Sargun Kaur joined Area 120,[3] Google's in-house incubator program, to explore ways to improve the technical hiring process for software engineers. As Google employees with software engineering backgrounds, Hardson-Hurley and Kaur had first-hand experience with the stress and difficulty of technical interviews. They had also spent months validating the problem from the employer's perspective, identifying opportunities to improve the effectiveness, efficiency, and equity of the

[1] Interview with Nikke Hardson-Hurley and Sargun Kaur, September 21, 2021. Subsequent quotations are from the author's interviews unless otherwise noted.

[2] Investors in this case refer to Area 120 partners, who manage Google's incubator funds and fulfill an investor-type role with start-ups in the incubator program

[3] Area 120, Google's in-house incubator, was founded in 2016 as a space where Googlers could spend 100 percent of their time incubating start-up ideas that might have value to Google's business.

Dominic Mirabile and Lecturer in Management Russell Siegelman prepared this case solely as the basis for class discussion. This case was made possible with the generous support of the Hunter Family Fund. Stanford GSB cases are not intended to serve as endorsements, sources of primary data, or illustrations of either effective or ineffective handling of an administrative situation. This case was reviewed and approved before publication by a company designate.

interview process. After researching standard practices, asking candidates about their interview experiences, and conducting need-finding sessions with software engineers and recruiters, Hardson-Hurley and Kaur were convinced that there was a big opportunity for improvement.

The incubator program was designed to help teams validate their core value hypotheses using resources from across Google. During the midway check-in, investors raised concerns about the team's progress. Other teams had already built prototypes and collected feedback on their value hypothesis. Hardson-Hurley and Kaur had yet to write a single line of code and investors felt that they should be focusing their efforts on developing and launching a high-quality product. Some investors suggested various features that would make their interviewing platform much better than current solutions. The team considered their investors' advice and wondered exactly how to proceed. Should they launch a full-feature product or first consider a minimum viable product (MVP)?[4] If they chose the latter, which features were necessary and sufficient to test their idea?

BACKGROUND

Nikke Hardson-Hurley and Sargun Kaur met in December 2017 at a Google Startup Weekend, a hackathon where Googlers could come together to try and solve interesting problems. Kaur earned her bachelor's degree in computer science from the University of California, Berkeley, and began her career at Yahoo before joining Google in 2014. Hardson-Hurley came from a family of educators and studied symbolic systems, a major comprising computer science, psychology, linguistics, and philosophy, at Stanford University before joining Google as an associate product manager on the Google Shopping team.

The two instantly connected over similar interests and career aspirations. To build on the Startup Weekend experience, they began meeting regularly to explore various ideas for social enterprises. They focused on four categories: civics, health, education, and diversity in tech. When they started to explore the problem of diversity in tech, Hardson-Hurley and Kaur investigated why companies struggled to improve diversity, especially given concerted efforts by large companies like Google. They began by examining current-state interviewing practices to understand how high-quality candidates with diverse backgrounds navigated the hiring processes.

Reflecting on their own journeys, Hardson-Hurley and Kaur immediately bonded over their terrible experiences with technical interviews. In addition, both recalled instances of referring excellent engineering candidates from backgrounds that are historically underrepresented in tech, specifically BIPOC, women, and non-binary individuals, and being surprised when the candidates did not pass the technical screening interviews. Perhaps there were structural issues in the interview process that led to decreased diversity among candidates, offerees, or both.

They wondered if they could create a more equitable technical interview process that might improve diversity in tech. As two women of color, working at one of Silicon Valley's leading tech

[4] The concept of a "minimum viable product" (MVP) was popularized by entrepreneur and author Eric Ries. An MVP is a product that has only enough features for essential functionality and is launched among a targeted set of "early adopter" consumers to validate demand and source customer feedback. That feedback is incorporated into product development in a rapid, iterative cycle.

companies, they thought, "Maybe this is something we could work on and maybe we're the right people to try and solve this problem."

The Market

In addition to a lack of diversity, the software engineering job market posed other challenges for recruiting organizations. First, the demand for software developers outstripped the supply. The number of software developer jobs in the United States was expected to grow by 21 percent from 2018 to 2028, an increase that was unlikely to be absorbed by new graduates in computer science.[5] This war for talent created fierce competition for software engineers, prompting recruiting organizations to interview more candidates for each role and to spend more on candidates during the interview process.

Second, software engineering jobs were highly skilled roles that required screening by a technical interviewer, usually a software engineer working at the company. For technology companies, a software engineer's time was extremely valuable, which made it difficult for recruiting organizations to coordinate schedules—and often lengthened the interview process.

Together, these issues demanded that recruiting organizations spend significant time and resources for each new hire. The market size of the U.S. information technology staffing market was $31 billion in 2018, which only included the outsourced recruiting market.[6] For companies that recruited and interviewed in-house, it was common for a large technology company to spend up to $25,000 per on-site candidate.[7] With more than 400,000 software engineers hired in the United States in 2017, companies were committing billions of dollars internally or externally to secure software engineering talent.

PROBLEM DEFINITION

Energized by the idea of creating a more equitable technical interview and convinced that organizations were spending considerable resources on interviewing, Hardson-Hurley and Kaur set out to validate the problem beyond their own experiences.

Crafting the Initial Problem Statement

Hardson-Hurley and Kaur began by creating a survey on Google Forms and posting across Facebook groups, Slack channels, class list serves, and LinkedIn. The simple survey asked participants about their candidate experience with software engineering interviews. They collected 37 responses detailing whether participants felt the interviews effectively assessed their ability, whether they understood why they were hired or not hired, if they felt discriminated against due

[5] U.S. Department of Labor Bureau of Labor Statistics, "Computer and Information Technology Software Developers," July 2018, https://www.bls.gov/ooh/computer-and-information-technology/software-developers.htm#tab-1 (October 25, 2021).

[6] "IT Staffing Growth Assessment: 2018 Update," Staffing Industry Analysts, February 2018, https://www2.staffingindustry.com/site/Research/Research-Reports/Americas/IT-Staffing-Growth-Assessment (October 25, 2021).

[7] Source: Byteboard analysis.

to their identity, and which skills were tested in the interview (see **Exhibit 1** for survey results). A troubling 73 percent of respondents felt that the interview did not effectively assess whether they could perform the job, and more than 27 percent answered "Yes" or "Maybe" to questions about whether they experienced discrimination during the interview process.

This data, even from a small sample size, was enough evidence for the team to keep digging. Hardson-Hurley and Kaur began researching and compiling dozens of articles, blog posts, and interview guides that described the challenges faced during the software engineering recruiting process. The team realized that many people felt their interview did not sufficiently evaluate their ability to perform the job, and that this deficiency in the process seemed central to the current lack of diversity in tech.

When reviewing the survey responses for skills tested in the interview, many participants mentioned "whiteboard coding." This was a standard interview format in which candidates were often given a theoretical computer science question and asked to code on a whiteboard with no other resources. This theme aligned with the experiences of Hardson-Hurley and Kaur and helped crystallize one of the key problems with technical interviewing. Kaur recalled:

> For my first software engineer interview process, I had to stand in front of a whiteboard and code. They asked an obscure bitwise operations question. It was something I wouldn't use in my day-to-day work—something I would ordinarily just look up if I really needed to. But because I had not memorized this specific concept, I couldn't even understand what they were asking. Standing at the whiteboard, with nothing but an Expo marker, I completely bombed the technical interview and didn't end up getting the job.

Hardson-Hurley and Kaur reflected on why the whiteboard coding interview was so daunting. Engineers were often asked to answer theoretical, rather than practical, questions in a setting that was drastically different from their typical work environment. This format contributed to many candidates feeling that employers were not getting a true sense of their ability from a whiteboard coding exercise. The team wondered, "What if we could flip the whiteboard interview on its head? What if we could design an interview that gave a better signal and more real data on a candidate? Let's make the whiteboard interview obsolete." With this aspiration, Byteboard, a more real-world, data-driven replacement for whiteboard coding interviews, was born.

Armed with evidence from initial online research and a small sample of survey data, Hardson-Hurley and Kaur now felt they had a working problem statement: whiteboard coding interviews did not provide an effective assessment of candidate ability. They felt they could design a better technical interview and applied to Google's in-house incubator, Area 120. If accepted, they could work on the idea full-time for at least six months and get access to Google resources and mentorship as they tested their hypothesis.

Problem Definition at Area 120

After describing the opportunity and the relevance to Google's business, Byteboard was accepted into the May 2018 cohort for the Area 120 incubator. Leveraging the customer development

framework and lean start-up principles, Hardson-Hurley and Kaur planned to use the program to test their value hypothesis by running experiments with customers.[8] Building on their initial problem statement around whiteboard coding interviews, the team first sought to deepen their understanding of the distinct pain points experienced by both employers and candidates throughout the interview process.

Problem #1: Ineffective assessment

Hardson-Hurley and Kaur started by focusing on the primary objective of an interview process: assessing whether candidates can perform the responsibilities of a software engineering role. From their survey data and interviews, they had a sense that the whiteboard coding format tested for theoretical knowledge, but they wanted to learn more about how the skills screened in the interview compared with the skills truly relevant for job performance.

Through additional surveys and talking to former candidates, they learned that technical interviews often screened for specific technical concepts that engineers might learn in a textbook or look up as needed, but not necessarily use in everyday work. They recalled that one engineer remarked, "Interviews test for a random set of concepts that you learned at some point. If you remember them, you get the job. If you don't [remember them], you don't get the job." This notion aligned with many of the interview preparation guides, which encouraged candidates to spend 100 or more hours memorizing common question types and concepts.

Furthermore, interviews failed to screen for non-technical skills such as big-picture project planning, interpersonal interactions, and creative problem-solving. Seeing that there was a large gap, the team set out to build a more comprehensive skills library for software engineers. When asking engineers and team managers, "What makes an effective software engineer?" Kaur recalled the data-gathering process:

> We set up calls throughout San Francisco and asked people to walk us through their day as an engineer. What do you do on the job? What skills do you practice every day? What do you value in your peers? What makes an effective engineering team? We wanted to hear from all these different perspectives. This broad sampling helped us synthesize what makes an effective engineer, which directly informed our thinking about how to improve the technical interview process by focusing on the skills that were used on-the-job.

Hardson-Hurley and Kaur saw that there was very little overlap between skills being tested and skills that made an effective engineer. Out of a compiled library of 33 skills demonstrated by effective software engineers, only 12 were being assessed in the current screening interview (see **Exhibit 2**).

[8] The Customer Development framework, the portion of the Lean Startup methodology aimed at understanding the problem, was developed by serial entrepreneur Steve Blank in the 1990s. Blank wrote a series of books outlining methodologies for product development that were heavily influential in what came to be known as the "lean start-up" movement.

The team felt this problem of an ineffective assessment was well-summarized by a common sports analogy, as Kaur explained:

> If you are assessing a recruit for an NBA team, picture asking them to walk up to a whiteboard and draw up plays. And that is how you would judge if you are going to recruit them on to the team or not. You're just having them draw plays and seeing if they draw them perfectly without any resources. You're not looking at how they play the game and you're not actually seeing them on the court. And that's how technical interviews are traditionally structured. That is what we wanted to change.

If Byteboard wanted to solve this problem, they needed to find a way to get candidates away from the whiteboard and evaluate them *on the court*.

Problem #2: Interview bias and stress

The persistent and growing demand for software engineers was drawing more and more candidates with diverse backgrounds into the industry. Online courses and diversity-focused organizations such as Black Girls Code worked to ensure a much broader population of candidates could build the skills and apply for software engineering roles. However, Hardson-Hurley and Kaur were concerned that unconscious bias was entering the interview process during the technical screening and impacting the rate of advancement to on-site interviews and, ultimately, job offers.

Unconscious bias in interview processes was a well-documented issue in diversity, equity, and inclusion (DEI) efforts, and many companies were looking for solutions to mitigate bias with a more equitable process. Hardson-Hurley and Kaur knew that as soon as a candidate interacted with an interviewer in person, a potential for interviewer bias was introduced. The interviewer might experience bias related to a candidate's gender, race, educational background, sexuality, or other identities—all of which were unrelated to a prospective employee's ability to perform the responsibilities of the job.

In addition, the Byteboard team found research studies showing that the performance anxiety associated with whiteboard coding reduced performance of candidates by more than a half, and disproportionately affected female candidates.[9] This dynamic contributed to a false-negative effect, in which high-quality candidates performed below their true skill level and were screened out early in the interview process.

Improving diversity in the tech sector was what drew Hardson-Hurley and Kaur to this problem in the first place. If they wanted to move the needle, they would have to address the issues of interview format and bias to ensure candidates were being evaluated based on their skills, rather than their identity or backgrounds.

[9] Mahnaz Behroozi, Shivani Shirolkar, Titus Barik, and Chris Parnin, "Does Stress Impact Technical Interview Performance?" *ESEC/FEC,* November 8, 2020, http://chrisparnin.me/pdf/stress_FSE_20.pdf (October 21, 2021).

Problem #3: Lengthy, inefficient process

The technical nature of software engineering roles and the fierce competition for talent meant that interview processes were designed to be extensive and rigorous. However, as Hardson-Hurley and Kaur mapped the process of several potential customers, they found it to be an inefficient, lengthy, and expensive undertaking.

For each software engineering role, companies began by spending 5 to 10 minutes per applicant to identify candidates who seemed qualified enough for an initial screening interview. For these applicants, software engineers at the company spent 1 hour conducting a technical screening interview over the phone. If candidates passed the screen, they were invited on-site for a full day of technical and behavioral interviews. If roughly 60 applicants yielded 10 candidates who moved on to the technical screen, 5 candidates who qualified for on-site interviews, and 1 offeree, companies spent at least 65 hours of engineering resources to extend a single offer.

For all the expense, recruiting organizations still only averaged a 20 to 24 percent on-site to offer rate,[10] meaning only one in four to one in five candidates brought in for on-site interviews received an offer.[11] Companies were spending valuable resources and still driving lackluster results.

The use of internal engineering resources also created two additional issues: process delays and interview inconsistency. Interviewing is not the primary job responsibility of a software engineer. Their time and energy are almost entirely allocated towards development. This mismatch led to drawn-out interviewing processes and high variance in interviewing skills across software engineers. Hardson-Hurley recalled realizing, "Just because you're a rockstar engineer doesn't necessarily mean you're a strong interviewer."

The more Hardson-Hurley and Kaur met with recruiting leads, the more they saw how desperate companies were to fill their open roles with top talent. They were spending tens of thousands of dollars per hire, utilizing staffing firms, iterating on the candidate experience, and working hard to reduce their time-to-hire. They were also generally open to trying new solutions. They viewed recruiting as an operationally intensive function that they had to continue to invest in due to its importance to the business. As designers, Hardson-Hurley and Kaur knew that there must be creative ways to smooth out the overall process using technology, outsourcing, or both.

CHOOSING A CORE VALUE HYPOTHESIS

The Byteboard team's problem definition process had identified three interrelated, but distinct, problems with the current software engineering interview model: ineffectiveness, inequity, and inefficiency. Given their uncertainty about which problem was most important for customers, they felt that it would be premature to invest in a lot of product development. They decided that their next step in the customer development process was to build an MVP that would validate their core value hypothesis. This hypothesis would articulate the value they would create that would compel customers to adopt their solution. Their MVP would consist of the minimum features needed to

[10] Benchmark based on research and interviews completed by the Byteboard team.

[11] On-site to offer rate is a metric for recruiting process efficiency, measuring the number of offers extended compared to the number of candidates brought in for later-stage on-site interviews.

prove that they created measurable value and that a critical mass of customers would consider their solution.

Revisiting their three problems, Hardson-Hurley and Kaur wondered: Should they focus on the benefits of a more effective screening, a more equitable interview experience, or a more efficient process for employers? Once they chose their core value hypothesis, which features would allow them to test it?

A More Effective Screening Interview

If the team believed that their core value was a more effective assessment of job-relevant skills, their MVP should demonstrate a higher correlation between a high interview score and either an offer extension or strong job performance. Their feature set might include revised questions, a project-based interview, scoring algorithms to differentiate candidates, and reports to help recruiting managers make better decisions. The value created accrued to recruiting managers and engineering teams, both of whom would benefit from the improved assessment of candidates and overall talent quality.

A More Equitable Interview Experience

The team also considered the core value hypothesis of a more equitable interview process. If increasing the diversity of engineering teams was Byteboard's core value proposition, an MVP should drive an increase in the number of job offers to high-quality candidates from backgrounds historically underrepresented in tech. To correct any false-negative effect impacting underrepresented candidates, Byteboard could include features that anonymized candidates, increased standardization in the process, and controlled for unconscious bias. Additionally, they could iterate on the candidate experience to eliminate sources of stress that disproportionately affected underrepresented candidates. This type of value appealed to recruiting leaders and engineering teams who were dedicated to improving diversity, as well as candidates from diverse backgrounds searching for a more equitable interview process.

A More Efficient Process

Finally, Hardson-Hurley and Kaur could test a value hypothesis around a more efficient process. They would target recruiting managers who were desperate to free up engineering resources and reduce operational complexity. An MVP to prove greater efficiency would improve process metrics (e.g., time to offer, cost per candidate, person-hours spent in the interview process). This value proposition resonated most with recruiting managers—especially with human resources VPs who typically oversaw the budget for corporate recruiting—and all other parties committing time and energy to hiring (e.g., engineers involved in recruiting).

Just before their midway investor check-in, Hardson-Hurley and Kaur had received a promising signal when discussing a potential pilot with a hiring manager. The hiring manager was eager to have Byteboard execute more than 100 technical interviews in the pilot, and also asked that all of the grading and scoring of the interviews be done by Byteboard.

DECISION TIME

The choices of which hypothesis to test and what features to consider were weighing heavily on the founders. The team had three months remaining in their incubator stay and they felt that for their time to be successful, they needed to secure initial pilots and generate enough data to prove their core value hypothesis. Their investors were used to seeing Area 120 teams iteratively develop solutions with customers and suggested several interesting features that they could begin building immediately.

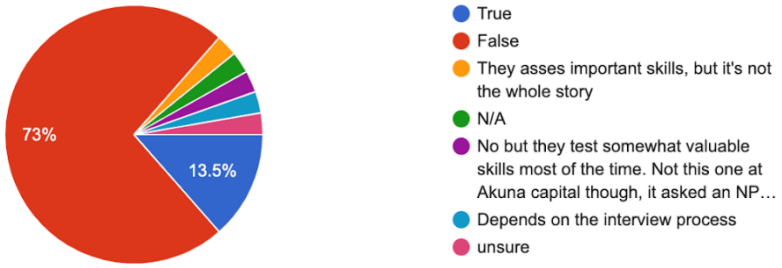
Hardson-Hurley and Kaur wanted more evidence before they invested a lot in product development, especially since they had time constraints and limited resources. They wanted to be sure they were building something that transformed how technical interviews were conducted, rather than just a set of fancy features.

The first stages of their customer development process had identified several problems that seemed important to engineers being interviewed, and to companies looking to hire them. What should they do next to move forward?

Exhibit 1
Initial Problem Definition Survey Results

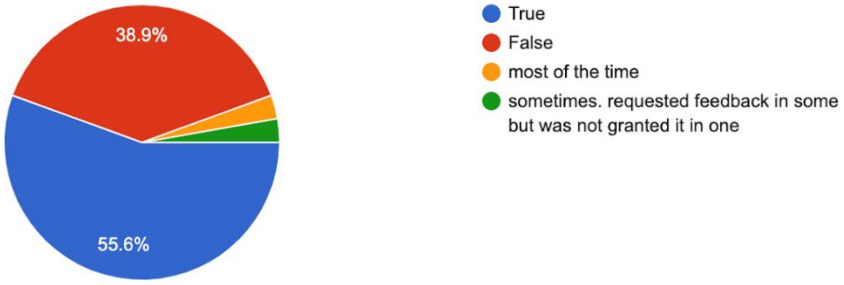
True or false: You feel as though SWE interviews assess how great you would be able to perform at the job.

37 responses



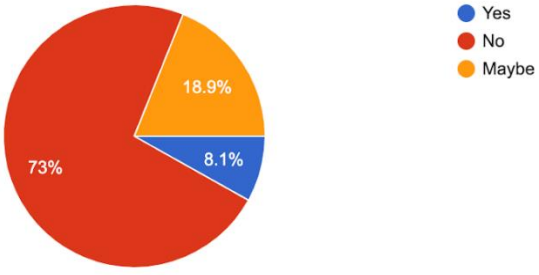
True or false: After your interviews you understood why you were hired or not hired.

36 responses



Have you ever felt like you were discriminated against during an interview process because of your identity?

37 responses



Source: Byteboard.

Exhibit 1 (continued)
Initial Problem Definition Survey Results

Survey Responses: What skills do you feel like are assessed in SWE interviews?
Ability to do SWE interviews. If the questions are well designed, assessing quick thinking as well (this is the best case)
Ability to problem solve and map solutions to code
Ability to quickly recognize familiar programming problems. Ability to communicate technical things. Ability to sell your experiences as being relevant.
Ability to regurgitate algorithms from a textbook
Ability to study for a particular kind of test (e.g. LSAT prep)
Ability to think, communicate, and problem solve
Algorithmic ability
Algorithms, data structures, mathematics
Algorithms, data structures, tech knowledge, and common design practices.
Algorithms, soft skills
Critical thinking under pressure
Data structure and manipulations, programming efficiency, and programming fluency
Data structures and algorithm knowledge
Data structures and algorithms
Depends a lot on the process. Some companies go for difficult algorithms puzzles, some go for high-level architectural design questions, some go for product sense and the ability to make judgment calls in poorly-specified problems, and some just test if you're able to throw a couple of loops and a print together. Even within a single company the process can be highly variable depending on who your interviewers are. My feeling is that the average company probably tests 70-80% for algorithms puzzles, though.
Dynamic Programming, Trees, Coding, some logic
Familiarity with undergrad level Data Structures (sometimes algorithms)
How well can you answer a particular suite of questions that everybody knows about
How well you can learn the strategies to do well in SWE interviews
Interviewing skills
Knowledge of logic, concepts, how to think, Communication, how you face new challenges
Mainly how you think, not so much about reading code which is an essential skill if you're working at big companies.
Memorization
On the spot problem solving, knowledge
Presentation skills/ability to think on your feet under stress
Problem solving
Problem solving
Problem solving and data structure knowledge
Problem solving, analytical thinking
Problem solving, talking through problems, clarifying problems
Quick thinking, decomposition, algorithm design
Time crunch solving of a problem with no resources. Thought process
Whiteboard coding
Whiteboarding, confidence, algorithms, design (in that order)

Source: Byteboard.

Exhibit 2
Skills Library

Skill Category	Skill
Growth Mindset and Grit	Ability & willingness to learn
	Passionate
	Proactive
	Persistent
	Productive
Big Picture Project Planning	Big picture view
	Setting reasonable goals
	Prioritize / manage time
	Product sense
Interpersonal Interaction	Communication
	Ask the right questions
	Collaborative
	Personable
	No ego
	Effective feedback
	Emotional intelligence
	Leadership
	Ability & willingness to help others
Problem Solving	Pattern matching & intuition
	Resourceful
	Problem decomposition
	Innovative problem solving
Code Composition and Comprehension	Writing clean code
	Be able to translate ideas to code
	Reading code
Systems Design	Trade offs analysis and reasoning
	Systems reasoning
	Distributed systems design
	Work within complex, large systems
Testing, Debugging, and Diligence	Debugging
	Identifying edge cases
	Testing
	Attention to detail

Source: Byteboard.