doBetter deBugging
A Custom Data Pipeline and Analytics Engine for Generating Insights from Online Debugging Interviews

Amit Gupta, Hannah Walsh
Changwook Shim, David Wu
Corresponding Author: *akgupta@wharton.upenn.edu*
Team-15

Advisor: Dr. Sangeeta Vohra

Abstract

doBetter deBugging is a fully self-contained web platform for asynchronous candidate evaluation in technical software engineering recruiting. The status quo process for recruiting is a drain on employee's time and largely disregards the highly relevant, on-job skill of debugging. As a product, our implementation offers increased efficiency in the recruiting process and interactive visualizations summarizing an entire debugging interview that can be interpreted in just minutes. Furthermore, we have softly validated product-market fit through a variety of stakeholder surveys and product tests. This design project shows promise as a tool in industry and lends itself to further development due to a lack of technical debt and business-conscious design.

1 Motivation and Functionality

1.1 Background

Having taken part in over fifty software engineering interviews cumulatively, we have been involved firsthand in the scheduling nightmare that is the algorithms phone screen. Interviewers have to take time out of their day to exchange emails with recruiters to set up phone calls. Those recruiters then reach out to us, the candidates, to find times that work. After a time is agreed upon, the interviewers spend an hour or more talking through an algorithms question with us candidates. Later still, they write a qualitative report on the interview which goes to a separate group of people called a hiring committee. And only after this are the candidates told whether or not we can move on to the next round of interviews. What you end up with is a process full of friction and lots of full-time employees spending inordinate amounts of time talking to candidates.

Furthermore, in our experience, most software engineering interviews are conducted using algorithms questions. One Facebook interviewer said that debugging is "one of the more important skills of a SWE [software engineer]", and as students and former software engineering interns, we agreed.

Given this motivating context, we believe there is an opportunity for a framework that helps the programmer and the recruiter understand debugging as a process from a fundamental perspective—one that quantifies actions taken by an individual in a concrete and meaningful way. Our project aims to fill this void. doBetter deBugging is a general platform deeply rooted in the fundamental facts of code which seeks to bring a meaningfully automated and quickly comprehensible analysis to debugging interviews.

1.2 High-Level Product Description

doBetter deBugging is a fully self-contained web platform for asynchronous candidate evaluation in technical software engineering recruiting. The product is targeted toward the technical software engineering interview where a candidate is given a programming task to solve. Typically, an interviewer is on the phone while the candidate completes the task. doBetter deBugging monitors the interaction so full-time employees no longer have to. For a narrated video demo of the product, please visit this link.

From a more detailed system architecture perspective, the product is composed of 3 main components: the interaction interface, the data pipeline, and the visualization library. The

interaction interface (Appendix A1) provides a full programming environment for the candidate equipped with standard debugging tools like dynamic testing and console outputs. The data pipeline tracks the candidate's edits with a custom server and connects the candidate's interactions to our analytics engine. The visualization library interactively summarizes the candidate's performance in a variety of time series, static, and overall metrics (Appendices A2-5).

The system allows the interviewer to more efficiently engage with the candidate's performance in an asynchronous manner. Furthermore, by focusing on debugging performance rather than algorithms trivia, the task is more directly representative of a software engineering (SWE) job.

1.3 Value Proposition to Stakeholders

doBetter deBugging is the technical interview platform for recruiters that understands how candidates interact with code. By replacing the traditional algorithms interview with a debugging exercise, our platform tests for a skill which we believe is much more relevant to employee effectiveness.

Our platform goes deeper than the traditional questions like "How long did it take?" and "Did it work once submitted?". The analytics engine dives way beyond that to quantify the whole interaction in terms of test cases and interpretable metrics that tell the whole story in a few interactive graphs and charts. These analyses are rooted in fact and transparency; the platform allows the interviewer to quickly follow any data point right down to the exact state and second of the candidate's exercise that created it.

While listening to an interview happen live allows similar detailed observation, doBetter deBugging allows the interviewer to quickly parse through the idle time and reach the relevant moments. Our platform short circuits the painful scheduling required to conduct these phone screens. By allowing the employee interviewer to evaluate the candidate asynchronously, doBetter deBugging saves countless hours of valuable company resources.

There are two main stakeholders in the recruiting pipeline: the candidate and the company. For this analysis, we take the candidates to be students as this is the group with which we have exceptional familiarity and context. For the company, we use the full time engineers who interview as a proxy as they have the most direct control over the interview process and are the ones eventually managing new hires.

Students have long lamented the algorithms interview and often spend so much time preparing for interviews that they do not even realize what the actual job is like: "Debugging code seems more representative of day-to-day work in SWE rather than solving Leetcode-style [algorithmsss questions." says one Turing Scholar from UT Austin.

One Facebook interviewer feels that debugging is "one of the more important skills" for a Software Engineer and responds to a doBetter deBugging demo with the following statement: "My current interviewing process doesn't do much testing...and is therefore prone to human error. This tool takes that error away and also sheds light on the candidates [sic] ability to think through a bug."

A Google Product Manager points out another benefit in "that more companies should have the early stages of the interview funnel/pipeline be asynchronous and require no effort from humans on their side." By removing the friction of scheduling a mutual time for a phone interview and the hours spent by employees listening to early stage candidates code, our product accelerates the early candidate funnel and saves countless employee hours, according to another Google employee.

doBetter deBugging is the coding challenge platform for university students that understands how they interact with buggy code. Our platform goes deeper than the traditional questions like "How long did it take?" and "Did it work once submitted?". By focusing on fundamental facts, our analytics target the niche between the uninterpretable analytics of cutting-edge research and handholding from a skilled observer. doBetter deBugging does not claim to rate performances as good or bad but instead summarizes facts on the interaction in a visually compelling, succinct report.

For the student programmer, we offer a way to understand interactions with buggy code, providing concrete metrics around what is a generally nebulous process. Our application

provides the accompaniments of any traditional programming platform, so users experience a natural programming flow. From this, though, our application goes further and generates a quantitative analysis around steps you took to locate the bug, test cases you wrote to make the issue concrete, and how you shifted your focus through time. This fundamental analysis, as compared to currently researched black box machine learning, offers the programmer a way to understand the quantification beyond just a generic score.

For recruiters, we offer a way to screen candidates that is representative of the job. Our platform allows employees to be uninvolved in the code screen process, saving countless hours of valuable company resources. Current state of the art recruiting tools simply output duration, the final code, and the raw number of test cases passed. Our debugging library and analytics dive way beyond that to quantify the whole interaction in terms of test cases and interpretable metrics that tell the whole story in a few graphs and charts. Given the immense costs of recruiting quality talent, even a slight improvement in signal-to-noise ratios in interviewing presents a worthwhile investment for corporations. By bringing in better debuggers, hours spent debugging (and thus costs) will go down.

2 Related Work

Ko and Myers (2004) show that bugs occur when a programmer experiences a cognitive breakdown or when they have misconceptions regarding language constructs [4]. In fact, Ahmadzadeh et al. (2005) claim most bugs are a result of something that is missing in the code while Simon and Hanks (2007) argue that the source is misinformation about what existing code actually does [1, 6]. In either case, the programmer has cognitively disconnected from the facts. A psychological gap exists between the scientific source of bugs and how programmers tend to approach debugging. By repeatedly identifying ways in which programmers approach debugging as a process, we build on existing debugging research in a practical and tangible application.

2.1 Competition

From a business perspective, we focus on related work in the candidate evaluation industry. To understand this, we take three industry leaders in slightly different verticals as a lens for the landscape.

CoderPad (https://coderpad.io/ - "CoderPad helps you hire better candidates faster, with an intuitive live programming environment") is an online coding interview software used by many firms (e.g. Facebook and Citadel). The system requires an employee to conduct the interview. They also log every keystroke made during the interview and act as a tool for interviewers rather than an analysis platform. Every keystroke made during the interview, though, is not a data source from which programmers or interviews can meaningfully infer information.

HackerRank (https://www.hackerrank.com/ - "Practice coding, prepare for interviews, and get hired") is an online coding interview software used by many firms (e.g. Amazon and Two Sigma) that also aggressively advertises training problems to students. The system allows you to create an account, track your progress on training questions, and even suggests practice modules focused on certain skills. This training component differentiates it as a go-to spot for interview preparation (other companies in this vertical include LeetCode). Their main feedback metric is the percentage of test cases passed, and they provide companies with your completed code. For the developer, they offer by far the most intuitive testing environment, exposing custom input and console output.

Triplebyte (https://triplebyte.com/ - "Get offers from top tech companies") is a relatively new player in the industry focused on the prescreening of applicants through a common certification program used by a few firms (e.g. Uber and Robinhood). They bring an innovative value proposition to the recruiter with a model that displaces employee hours spent interviewing. They replace algorithmic interviews with screening questions more relevant to job performance like devOps and application management (we went through Triplebyte's certification process to gain insight). They then certify applicants as meeting a baseline skill level and forward resumes directly to

companies. Triplebyte has a monetization model in the vertical most similar to what we are targeting.

2.2 Market Research

Asynchronous candidate evaluation presents a market opportunity in that candidate pipelines continue to grow wider with the explosion of Computer Science graduates and employees are continuing to spend more time interviewing candidates. For a sense of scale, Google itself receives 2 million applications every year [5]. In fact, the Bureau of Labor Statistics puts 10 year projected growth for software developer employment at 21% (far above national averages). However, the cost of this time to companies is dwarfed by the amount of time software engineers lose to debugging every year.

doBetter deBugging provides a solution on both of these vectors – one which has eliminated the bulk of the technical risk and has soft validation of product market fit from key stakeholders. By being fully self-contained, the product is amenable to any company with a software engineer hiring need from small startups to Fortune 100 tech firms. By screening early on for debugging talent, companies can increase their productivity. As evidenced by the sheer size of the industry and the problem, even an incremental improvement in the signal-to-noise ratios of interviewing yields large gains for the firms. Every year, a new recruiting technology enters the market, but few to none offer the continuing value that doBetter deBugging does through on job performance improvements.

Willingness to spend in this area is evidenced from a recruiting perspective by the wide variety of available software solutions and emerging solutions gaining traction. HackerRank, for example, charges $3,000 a year for a single interviewing account which is limited to 30 candidate screens a month. No offering we currently see in the market, though, has a natural programming experience for the user and creates digestible insights for performance understanding. This is our unique value proposition.

Current platforms either facilitate employee-led interviews or attempt to displace the employee-led interview. In industry, there has been a shift to more automated recruiting tools as the number of applications explodes. The driving factor here is that the employee hours spent interviewing candidates are neither cost effective nor scalable. Nearly all resume screens for large technology companies are automated, which further demonstrates a willingness to automate early steps in the candidate pipeline. Overall, this industry consists of an existing market.

The industry is open to iteratively improved solutions. To understand the willingness to try new tools in the candidate pipeline, we return to the earlier number: $156B in yearly wages lost on debugging time. An improvement from 75% to just 70% of SWE-hours spent debugging, then, amounts to a $10.4B gain in productivity. This improvement is the hook of our marketing pitch.

Where we believe we are uniquely positioned to enter this industry is through our understanding of the student population. Recruiting software is one of the initial points of contact in the recruiting relationship between a candidate and a company. Attrition here is a particularly harmful loss to the candidate pipeline for companies as they never get to evaluate candidates who do not complete such code screens. Fewer applicants means fewer quality applicants and less healthy candidate pipelines. As such, we believe that demand in this industry is driven by student preference for platforms. One of the major determinants of student platform preference is transparency. HackerRank exposes a practice environment which is immensely popular with students. Triplebyte offers 3 attempts at the same quiz. These practice environments grow the user bases of these platforms. As with any two-sided marketplace, user base is at the heart of efficient performance.

As such, we turn to understand student demand for debugging understanding at the university level. For students to be willing to be assessed on this metric, they will first need to believe that this is relevant to industry and then be taught techniques. The first is evident from experience and the nature of industrial software engineering, so we turn to the second. See The full survey can be found in Appendix E2.

Societal Impact for more detail on this.

doBetter deBugging is based on a true dual value proposition, offering understanding to students and improving signals for recruiters with cost effective candidate pipelines.

3 Technical Approach

doBetter deBugging is a webapp on which evaluators and candidates can conduct debugging software engineering interviews asynchronously. It is built on a React JS frontend with a Django backend and SQLite database. It consists of the four components listed below.

The interaction interface draws from the buggy code base to display 15-30 line buggy code examples to the user. When the user makes any edits, the data pipeline notes these changes and stores them in the database for later analysis. After the user finishes making edits and submits the exercise, the visualization library displays insights and analysis on the interactions.

3.1 Buggy Code Base

The buggy code base is a collection of 15 complete buggy code examples. Each example is no more than 30 lines long and includes a problem statement that tells the user what the code should be implementing and expected output in various input scenarios. Early examples are all presented in Python. All examples come with a few public tests that will be displayed to the user and a number of private tests that we will use for analysis. No more examples are needed for testing the functionality and proof of concept of the platform, but we have a list of over 100 bugs that we can implement if needed. These are prepared in a structured way and compatible with our databasing.

We built this component so there was no onus on the evaluator to supply the code they want their candidate to work on. In general, though, companies like to use their own interview questions which will often be geared toward the field of the company. While doBetter deBugging is built around debugging, it can also handle general coding problems. The buggy code base exists so evaluators have the option to select a specific type of bug if they do not wish to supply their own.

3.2 Interaction Interface

This component displays two text editors to the user, one that contains a code example from the buggy code base and another that contains the associated public test cases. The user is able to make edits to both the code and the test cases. While editing, they can choose to run their updated test cases (or just one test to see more detailed output) on their updated code. The output of the test run is displayed next to the second text editor.

In addition to the programming components, we also offer the user the ability to opt in to our speech-to-text feature. To do this, we display an alert within the Interaction Interface that asks them if they would like to opt in to voice recording. If they do, we display a full text transcript of what they said during the interview in the Visualization Library. A fully functional version can be seen in this video.

A challenge we ran into building this component was effectively mimicking a standard development environment. In addition to allowing users to write their own unit tests, we wanted them to be able to debug with console output from those tests (which is something other asynchronous platforms often do not provide). To do this, we had to store the candidate's code as text, send that text to the Django backend, load the text as a Python module and run the test cases on it. Since the output of the test runs was sent to the console, we had to import Python's OS module to redirect it to a string we could send back to frontend. This took us a while to work out, but, based on the evaluations discussed in Interaction Interface, it made doBetter deBugging feel like a standard programming environment.

3.3 Data Pipeline

When the candidate makes any edit within the Interaction Interface, whether it is to the test cases or the buggy code example, the updated version of the code is saved to our database under the unique ID for this user's session and a timestamp. Each of these code snapshots will be fetched later using this session ID for analysis.

Analysis is done using different code snapshot listeners, where a listener is a function that takes in a snapshot and outputs information associated with it, like whether or not it statically checks or how many test cases it passes. We iterate through all snapshots one by one and run all our listeners on them. We compile all the data gleaned from this into both an analysis of what and how the user did over time and an individual analysis on each code snapshot. We send all this information back to frontend in a JSON format that is compatible with

the graphing components we use in the Visualization Library.

In addition to the data we collect from candidate edits, we also send the file containing their voice recording (assuming they opted in) to the data pipeline. Never storing this file itself anywhere, we translate it to text using Google Cloud's Speech-to-Text API. After getting the raw text back, we go in and add timestamps and punctuation to it so we can effectively display it to the evaluator later.

3.4 Visualization Library

This component displays the information we processed using the Data Pipeline for the evaluator to see. It consists of four different tabs. The Time Series Analysis tab can display graphs that provide timestamped data points on six different metrics: Hidden Tests, User Tests, Print Information, Test Quality, Test Correctness, and Cheat Checking. Clicking any of the buttons within the tab will change the metric the evaluator is looking it.

Clicking a data point on a graph will take an evaluator to the Snapshot Analysis tab, where they can see the code the candidate was working on at that moment in time. This tab allows evaluators to flip back and forth between candidate's code snapshots, filter out snapshots that do not statically check, and, for each snapshot, we provide a static analysis detailing how many test cases this snapshot passed.

The third tab is the Speech Transcript. Here we display the text we got from the Google Cloud API. Each word in said text is formatted to be a button, so when the evaluator clicks on it, they can see the code snapshot the candidate was working on when they said that word. There is a button that allows the evaluator to view the same snapshot in the Snapshot Analysis tab as well.

The fourth and final tab of the Visualization Library is Individual Insights. Here we display a pie chart for the evaluator detailing how long the candidate spent thinking (or sitting idle), running code, writing tests, and writing print statements. Based on whatever the candidate spent the most time doing, we highlight a small paragraph explaining what that may or may not imply about the candidate's debugging style.

A fully functional version can be seen in this video.

3.5 Cost and Revenue Models

doBetter deBugging at its core is a SaaS (Software-as-a-Service) play. Simple competition-based pricing allows a quick way to benchmark revenues. HackerRank, for example, charges $3,000 a year for a single interviewing account which is limited to 30 candidate screens a month. CoderPad which offers decidedly less functionality still charges $750 a month if you interview up to 120 candidates (https://coderpad.io/pricing). We see no need to charge less than our competition as we hold no competitive advantage from a costing perspective to undercut them with. Instead we are proposing a value add for no increased cost. More specifically, in our vertical, tiered pricing makes the most sense and is what you see across the B2B SaaS sector. Monthly estimates are as follows:

- 1-week free trial
- $50 Up to 5 interviews
- $250 Up to 25 interviews
- $1,000 Up to 100 interviews
- $2,500+ for a custom contract

Variable costs are essentially negligible on a per use basis due to the lightweight interface and availability of flexible compute. Marking down 5% variable costs on revenue offers a conservative cover for maintenance and other server needs. Find a full breakdown in Appendix B.

Customer acquisition costs will be the greatest concern as we rollout. We have relatively little information around what it will take to close deals. As such, offering free trials to small companies will be a low friction way to gain early customers. Ideally, doBetter deBugging entrenches itself with growing companies and allows those customers to grow our revenue as they grow. Our projections suggest the first two months of revenue from each new customer will go to covering acquisition costs. From there, the stickiness of the product will have to prove itself but will provide nearly pure profit.

Before all that though, our cost model sees a few constraints. While the current demo is compelling it lacks some key functionality which would be necessary to begin selling. First, the system needs a security upgrade. Any system which has users inputting code needs to be locked down from code injection. Furthermore, privacy concerns for the users will need to be addressed through encryption and other standard measures. Optimizations for different screen sizes are also missing. User

License and other legal agreements will need to be put into place, as well. Lastly, the platform will need to have a user account management system built for both interviewers and candidates. While these steps seem scary, industry-standard solutions exist for each of these and can be implemented through outsourcing for under $30k and within 12 weeks, see Appendix C for details. Securing 3 customers in the 1-100 employee range at $250 each in monthly recurring revenue (MRR) and 3 slightly larger companies at $1,000 MRR each gives just over 8 months to break even on the additional investment required. See Appendix D for more details.

Ending with a brief statement explaining doBetter deBugging as a company instead of just a product, we turn to a data play. While the current value proposition of the product is compelling, as we collect data on an increasing number of candidate interactions and interviewer evaluations (pass or reject), we build an immensely valuable dataset to further automate this process. doBetter deBugging is the time-efficient, data-driven approach to recruiting that focuses on the relevant skills.

4 Evaluation

4.1 Buggy Code Base

The effectiveness of the buggy code base relies on how extensible it is, how well it integrates with the interaction interface and data pipeline, and how helpful it is to evaluators. We designed the code base under a fully general yet well-defined structure. This has allowed us to add to the code base easily and will allow us to fetch a specific kind of bug for a user.

With respect to integration, each buggy code example is based on a predetermined template. This has made it very straightforward to load an arbitrary example into the interaction interface for any given session. Furthermore, since all examples are structured identically due to the template, our code listeners have been generalized to work on all snapshots. Although small, the code base provides enough example code for users to interact with the system in a meaningful way. In the event we run many repeat tests with the same users, we have a list of over 100 additional bugs. By creating a structured template and standard implementation for these examples, writing and testing a new code example takes under 30 minutes.

To evaluate how useful our buggy code base is to evaluators, we would need to perform a long-term study to see how different companies use it (if at all) and then ask them to evaluate its completeness and usefulness. Our plan is explained in depth in Long-Term Studies.

4.2 Interaction Interface

To evaluate the interaction interface, we turned to those who would be using it: students. Since the interaction interface is entirely used by the candidate, we sent a survey out to 30-40 different computer science majors at various top universities who have interviewed for software engineering positions. As students who have had been through the interview process ourselves, we identified our three main pain points in the standard interviewing process: limited development environments, complicated interfaces, and irrelevant questions. Since we are trying to offer an alternative to the current process, we asked students to evaluate numerically on each of those metrics. We also asked for them to evaluate our system overall and provide qualitative feedback, since our pain points may not be everyone's pain points.

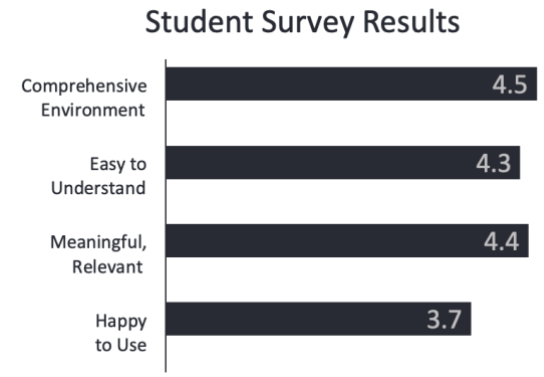


Figure 1: Quantitative results from the student survey (metrics rated out of 5).

As seen in Figure 1 above, the students and graduates who responded to our survey found our system to be comprehensive (average rating 4.4/5), easy to understand (4.3/5), and highly relevant (4.4/5).

The feedback above indicates that we mostly alleviated the pain points we had identified.

However, as implied by the overall score, there was still room for improvement, so we turned to the qualitative feedback to see what we needed to address (fortunately, we had sent his survey out in February, so we had time to address concerns).

In response to our question "what functionality seems to be lacking or missing?", a few students expressed concerns about cheating ("I personally prefer in person interviews because cheating is a major concern.") and about not being able to explain their thought process to evaluators ("I think it is important to hear the thought process one goes through, even if they don't get the right answer".) To address both of these concerns, we added both the Speech Transcript and the Cheat Checker to the Visualization Library. The Speech Transcript gives an evaluator the ability to read a candidate's thought process (which they would have to voice out loud in a traditional interview as well. The Cheat Checker shows all copies and pastes the candidate did in in their interview, and highlights pastes that come from foreign sites to minimize cheating.

We have not yet followed up on these additions with those students, but we believe the positive feedback we received both in Figure 1 and in some qualitative responses indicate that potential candidates like the product. ("I think this is a great idea and I really liked the snapshot analysis feature, good job!", "I think some environments make it confusing or even impossible to run my own test cases, and it looks like the environment in this demo really focuses on making that an easier experience.", "It'd make a good addition to a repertoire of test [sic] to run on candidates, makes it feel more holistic.")

The full survey can be found in Appendix E1.

4.3 Data Pipeline

For the data pipeline to be effective, it must be faster than the current industry standard. One of our major design goals was improved process efficiency. As such, we ran repeated simulations on the end-to-end time it takes to conduct and evaluate an interview through our platform. This gives an estimate of how much time evaluators would save by switching to our system.

Said simulations consisted of each member of our team pretending to be a candidate and completing a debugging exercise on doBetter deBugging. After completing the exercise, we each set a timer to see how long the data from the pipeline took to process, and how long it took us to comprehensively review all the tabs in the Visualization Library. We each did this twice, opting in to the Speech to Text feature the first time and opting out the second time. Below in Figures 2 and 3 are the results.

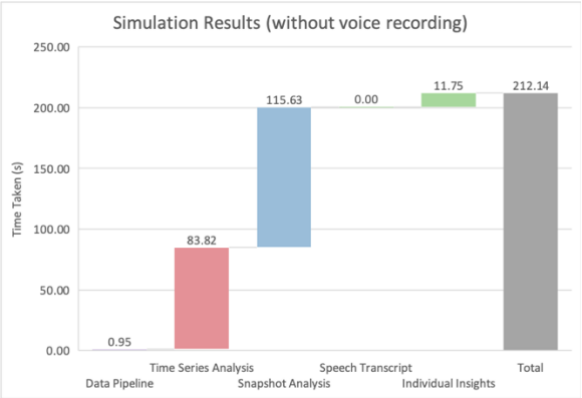


Figure 2: Simulations done after opting out of the voice recording feature.

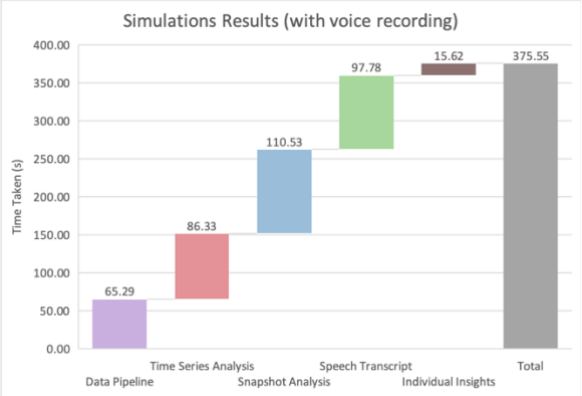


Figure 3: Simulation done after opting in to the voice recording feature.

Assuming that a software engineering phone screen takes one hour, our system performs much faster on average. This does not even include gains from not having to schedule interviews! Note that since we as developers are familiar with the platform, we may have moved through the Visualization Library quickly, but even if you were to add an extra 5 or 10 minutes to the process, doBetter deBugging is much faster under the assumptions we made.

The numbers in Figures 2 and 3 are averages from the eight simulations we ran, please see Appendix F for the raw data.

4.4 Visualization Library

The function of the visualization library is to display a meaningful summary of a user's interactions and our analysis in a digestible, yet comprehensive format. So, to evaluate it, we turned to the people who would be using it: interviewers. We created another survey like the one used for the Interaction Interface but this time, we sent it to stakeholders at both Google and Facebook. We asked them to rate doBetter deBugging on four metrics: completeness of data, digestibility, relevance, and how pleased they would be if their company began to use it. When directly asked how happy individuals would be if their company began to use our product for candidate evaluation, professionals responded with an average rating of 4 out of 5. The full results from the survey are displayed below in Figure 4.

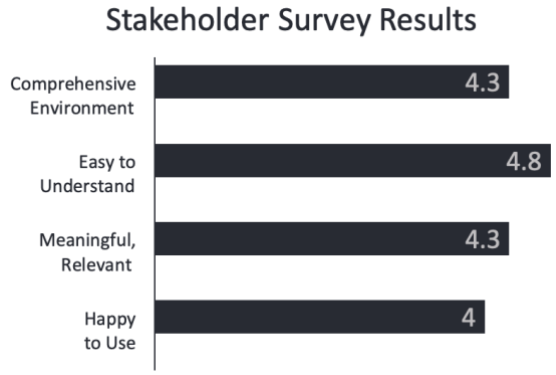


Figure 4: Quantitative results from the stakeholder survey (metrics rated out of 5).

Qualitative feedback was very positive as well: "from a product perspective, allowing me to skip to just relevant changes in the code based on whether they got a test right or wrong was awesome!" and "sheds light on the candidate's ability to think through a bug (which IMO [in my opinion] is one of the more important skills in a SWE (Software Engineer).".

Like the student survey, one stakeholder was concerned about a candidate not being able to voice their thought process ("The interviewee could want to add insight with a voiceover, to explain why they're doing a certain thing."), so the addition of the Speech Transcript also addressed stakeholder concerns.

The full survey can be found in Appendix E2.

5 Societal Impact

In addition to the feedback we requested on the system functionality itself, we also asked for people to voice their ethical concerns. Many of our classmates noted that there is already a great deal of bias (both conscious and unconscious) and cheating that happens within the traditional software engineering interview process. To truly be considered an ethical product, we cannot ignore those existing issues. If we want to offer a *better* alternative, it must be better in all ways – so we attempted to tackle these issues as well.

With respect to bias, doBetter deBugging has the potential to completely anonymize the interview process. By converting speech to text with an opt in feature, an evaluator can interact with a candidate without knowing their name, gender, race, or even what their voice sounds like. In fact, since we use Google Cloud to make this possible, doBetter deBugging can interview candidates in almost any language as well, since the speech to text feature supports over 120 different languages. After getting the text, we can use Google's translation API to translate it into whatever language the evaluator wishes to read in. We have personally tested the feature with English, Chinese, Korean and Spanish, and all worked seamlessly. As long as the evaluator goes into the interview without knowing anything about the candidate, the interview itself will provide no information about them. By completely anonymizing the interview, we can actually eliminate both conscious and unconscious bias from the interview process. Candidates can only be judged on their ability to solve the problem, giving qualified minorities the fair chance they deserve.

To tackle the issue of cheating, we created the Cheat Checker feature to track copies and pastes done by candidates. While not a perfect indication as to whether or not someone cheated during an evaluation, it can highlight red flags to keep evaluators aware.

Lastly, in addition to tackling current societal problems, we believe that doBetter deBugging's emphasis on debugging may also bring about social good for both universities and tech firms. Penn CIS faculty Dr. Arvind Bhusnurmath, when asked about the presence of debugging in education, said that the CIS curriculum at Penn offers no courses that teach students how to debug.

He believes that we should teach students about debugging techniques and the debugging process while they are still at college, as it is a non-trivial skill that is a very important part of a computer science student's future career. Professor Steve Zdancewic of CIS 120 agreed that the presence of debugging in the introductory CIS curriculum is lacking. Through a product that encourages companies to evaluate students on their debugging ability (but supports all coding exercises of all types), doBetter deBugging may be able to help emphasize the relevance of debugging in both university and industry.

6 Discussion, Lessons, and Future Work

6.1 Discussion and Lessons

In designing, implementing, and building this system, we went through the full life-cycle of development, including background research, product design, and testing. doBetter deBugging was inspired by a real-world friction. Because of this concrete motivation, the process of choosing what features to implement and how to allocate time was fairly natural. By considering the users' needs throughout the design and implementation process, we arrived at a fully functional product which can provide value to all of our stakeholders.

Along the way, we often were faced with a choice between a quick workaround and a longer design process to build a system the right way. Time and time again we chose the latter to avoid technical debt and build an understanding of how our system might scale. As we continued to encounter challenges like running arbitrary user code, connecting spoken words to a specific snapshot of code, dynamically generating graphs, or even presenting the work to stakeholders, this principled and generalizable design approach proved rewarding. The system and plans were often updated and modified based on feedback, but the overall framework and goals remained constant.

We started by generating a large number of potential use cases for our product before focusing on the one we were uniquely positioned to tackle. In other words, not every technical challenge is worth solving. We solved many which were core to the outcomes and functionality we desired, but we also chose not to solve others (like security) as they were not unique to our project and existing work can be leveraged to address those concerns. Such an exercise allows for careful selection of the correct application to focus on in the earlier stages of a project. Now, we speak toward longer term plans.

6.2 Long-Term Studies

A lot of this project is designed on real-world applications and improvement over existing processes. In order to validate product-market fit as well as technical effectiveness, we require a longer term study. This study is composed of two key phases. The first is customer acquisition. Acquiring customer provides an early litmus test for the quality of our product and the reality of the issues we propose solving. Paying customers putting money behind the strong verbal support we've already seen and customer conversion metrics will inform which parts of our project were successfully and meaningfully implemented. The Cost and Revenue Models section has spoken more to this.

The second phase is tracking customer retention and tangible impact of candidate pipelines – both in terms of efficiency with candidate volumes and quality of hiring signal. The latter is most critical as we will have to assess the on-job performance of candidates hired with our focus on debugging and through our asynchronous interview insights. Key performance metrics to track will certainly include the following: time spent per candidate interviewed, total time spent per candidate hired, average time to promotion of candidate, as well as the firm's own performance reviews. This will take the form of constant customer relations with the hiring managers and on-job managers who will be asked to provide regular feedback on their candidates. These individuals have the closest real-world connection to the problem we have attempted to address in this project and, thus, represent the critical testing ground for the larger impact and evaluation of our project.

6.3 Takeaways

Building a system for users presents a unique technical challenge in that there is a tradeoff between perceived feature completeness and speed of development. As such, we spent a fair bit of time at the outset of the project designing a scalable and robust framework upon which to build the project.

This has led to heavily modularized code and development flows which allow for feature and listener addition with ease. Each feature or analysis tool added is isolated from others and can fail safely. The work completed and demoed above lays the foundation upon which we can continue to innovate as the project develops. The user experience seamlessly enables debugging while tracking the data we need to automatically provide the mocked visualizations.

The code snapshot listeners tell a unique and digestible story about how a user interacted with a snippet of buggy code. From this story, users and recruiters alike are able to draw their own conclusions about the user's debugging ability. The stories the listeners tell and the conclusions individuals can draw form a richer and more informative analysis than state of the art online programming competitors.

Students use the story that our analysis tells to better prepare for both interviews and industry itself. Recruiters also use this story to better evaluate candidates without spending employee hours on code screens. Our project is built on a unique understanding of the student population, transparency as a core value, and justifiable fundamental insights in the billion-dollar industry of tech talent acquisition.

References

Ahmadzadeh, Marzieh, et al. "The Impact of Improving Debugging Skill on Programming Ability." *Innovation in Teaching and Learning in Information and Computer Sciences*, vol. 6, no. 4, 15 Dec. 2005, pp. 72–87., doi:10.11120/ital.2007.06040072.

Assaraf, Ariel. "This Is What Your Developers Are Doing 75% of the Time." *Coralogix*, 20 Feb. 2019, coralogix.com/log-analytics-blog/this-is-what-your-developers-are-doing-75-of-the-time-and-this-is-the-cost-you-pay/.

Britton, Tom, et al. "Reversible Debugging Software." University of Cambridge. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.370.9611&rep=rep1&type=pdf.

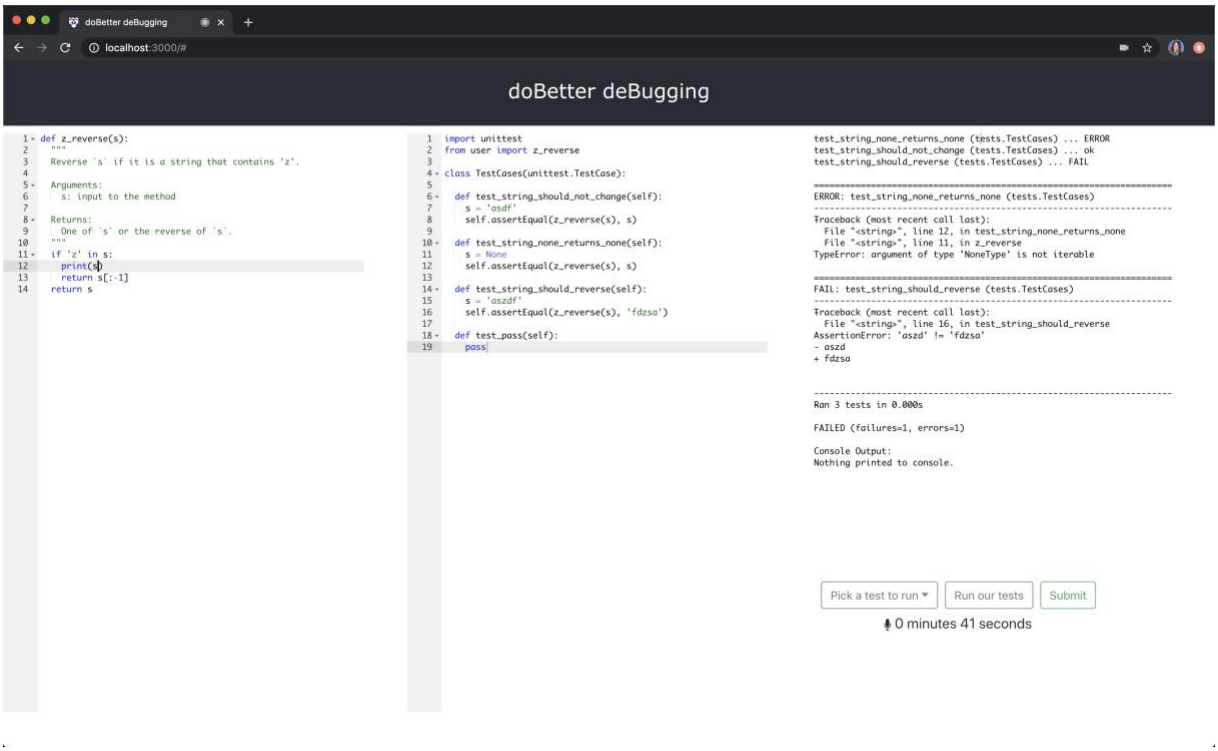
Ko, Andrew J., and Brad A. Myers. "A Framework and Methodology for Studying the Causes of Software Errors in Programming Systems." *Journal of Visual Languages & Computing*, vol. 16, no. 1-2, 1 Aug. 2004, pp. 41–84., doi:10.1016/j.jvlc.2004.08.003.

Shain, Susan. "2 Million People Apply to Work at Google Each Year. Here's Why." *Business Insider*, 14 Sept. 2014, businessinsider.com/2-million-people-apply-to-work-at-google-each-year-heres-why-2014-9.

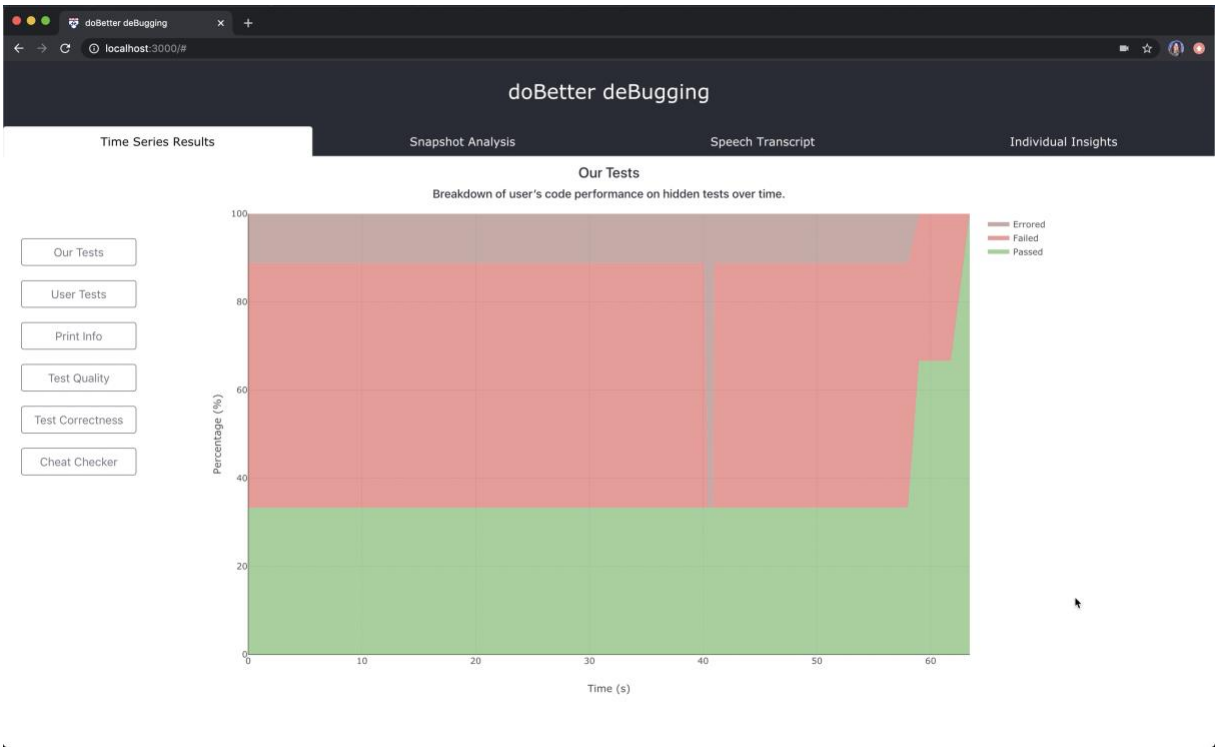
Simon, Beth, and Brian Hanks. "First Year Students' Impressions of Pair Programming in CS1." *Proceedings of the Third International Workshop on Computing Education Research - ICER '07*, 15 Sep. 2007, doi:10.1145/1288580.1288591.

Appendices

Appendix A1. Candidate View

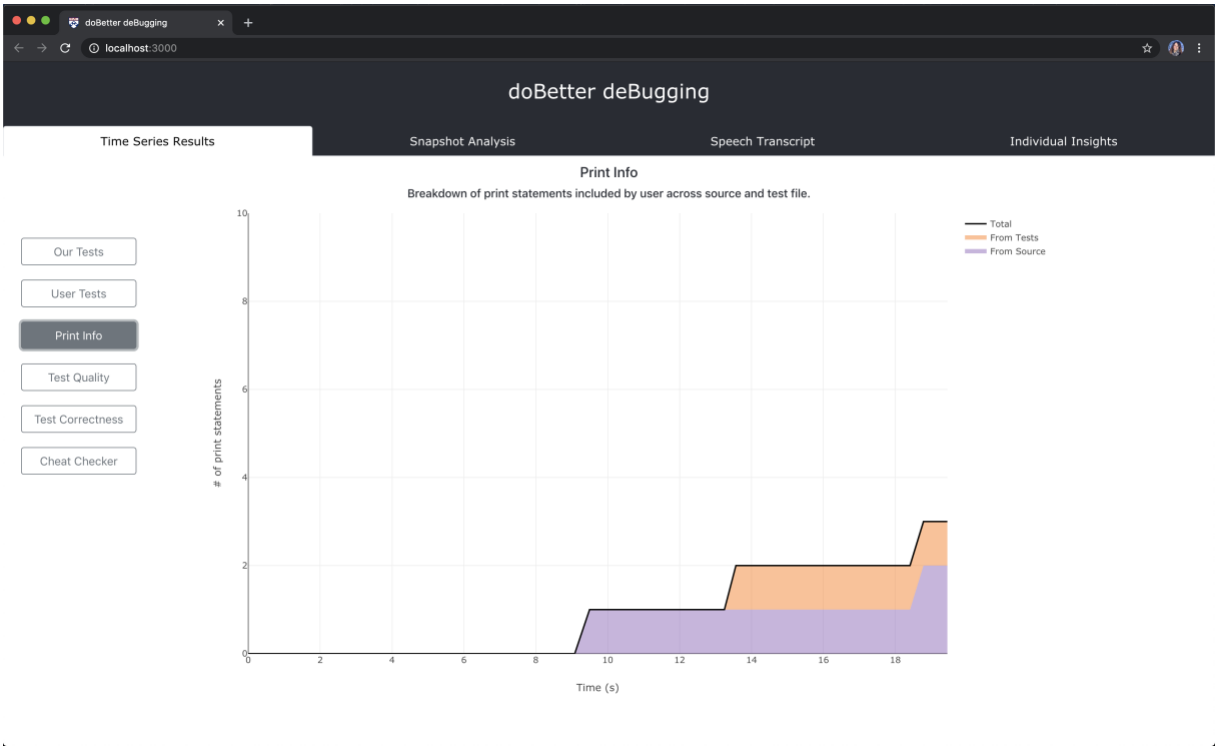


Appendix A2.1. Time Series: Our Tests



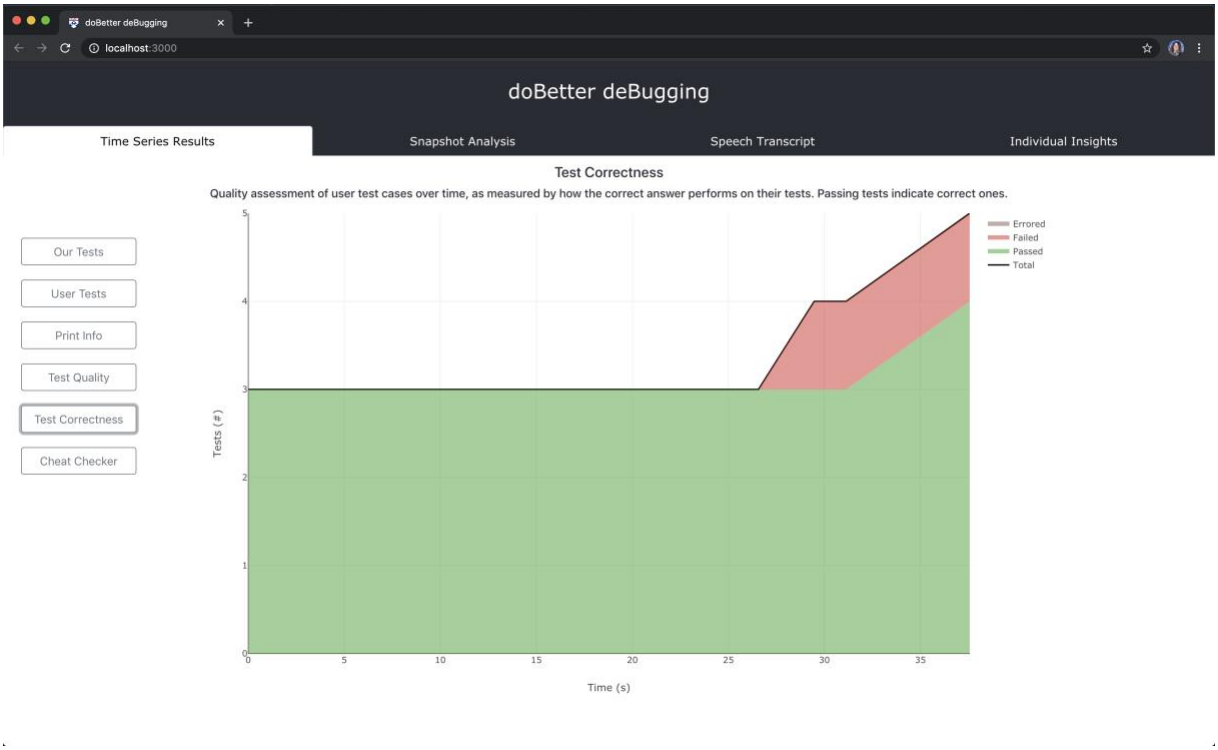
Appendix A2.2. Time Series: User Tests

Appendix A2.3. Time Series: Print Info

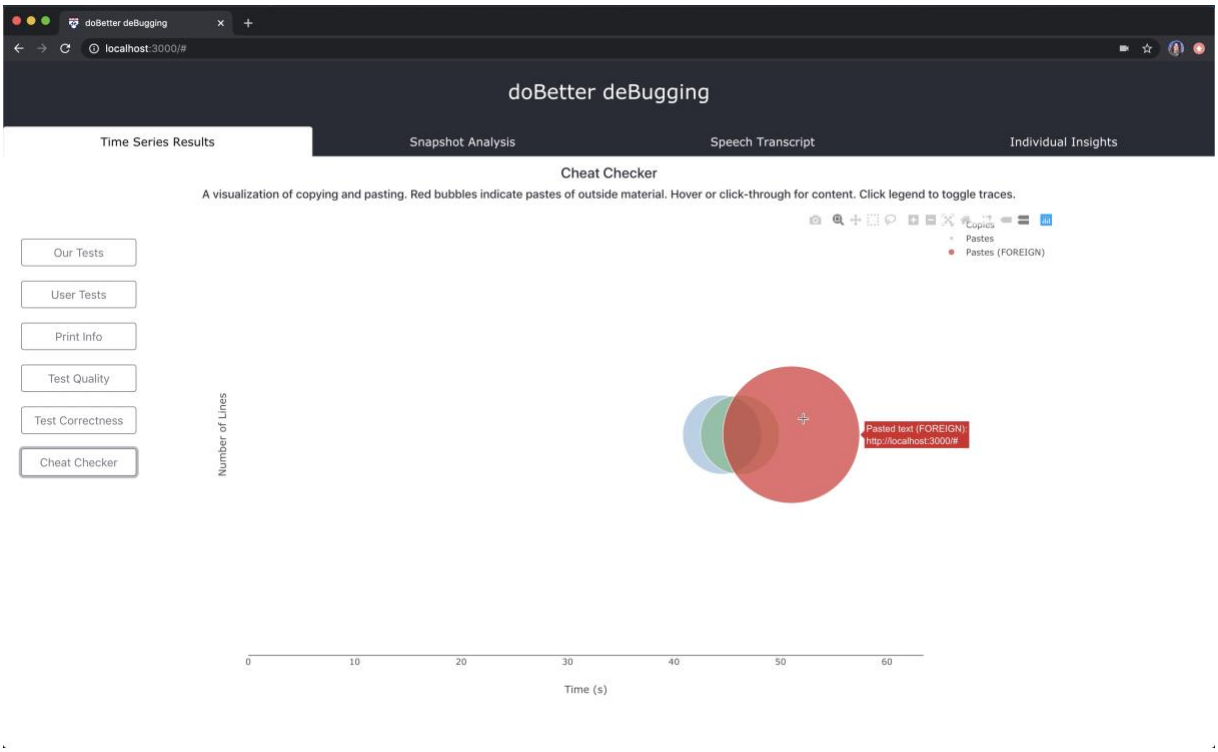


Appendix A2.4. Time Series: Test Quality

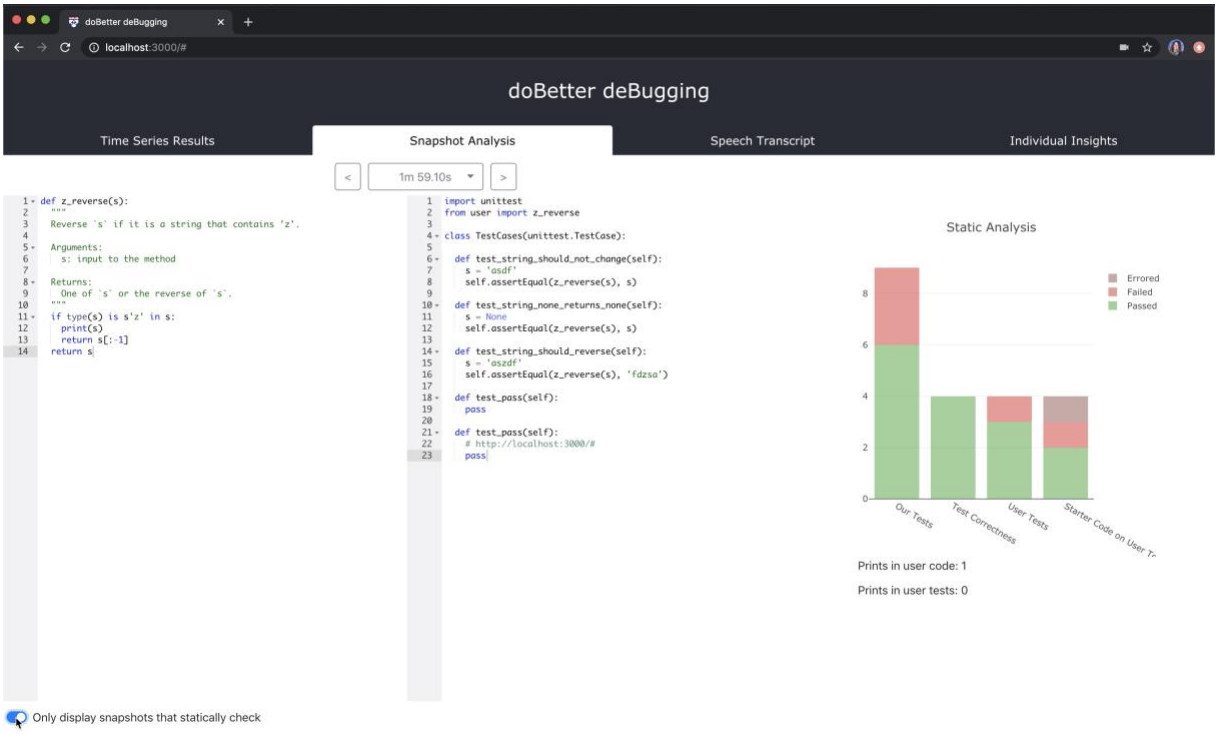
Appendix A2.5. Time Series: Test Correctness



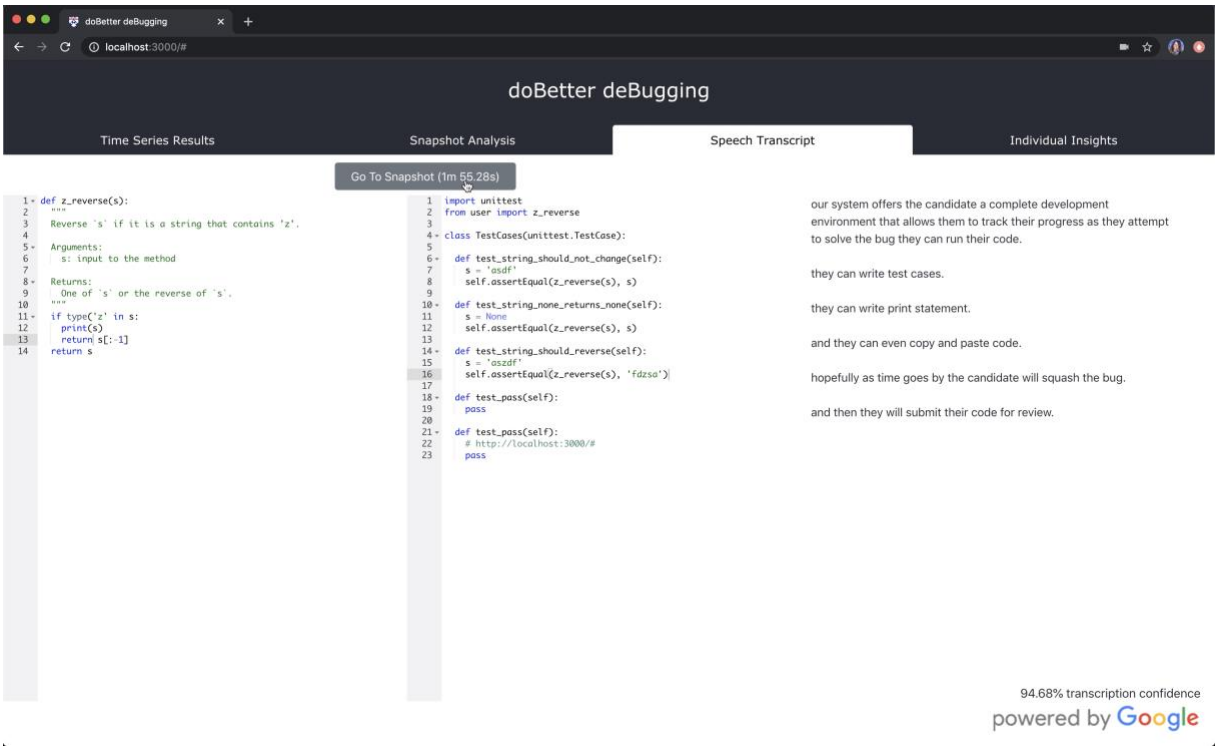
Appendix A2.6. Time Series: Cheat Checker (premium)



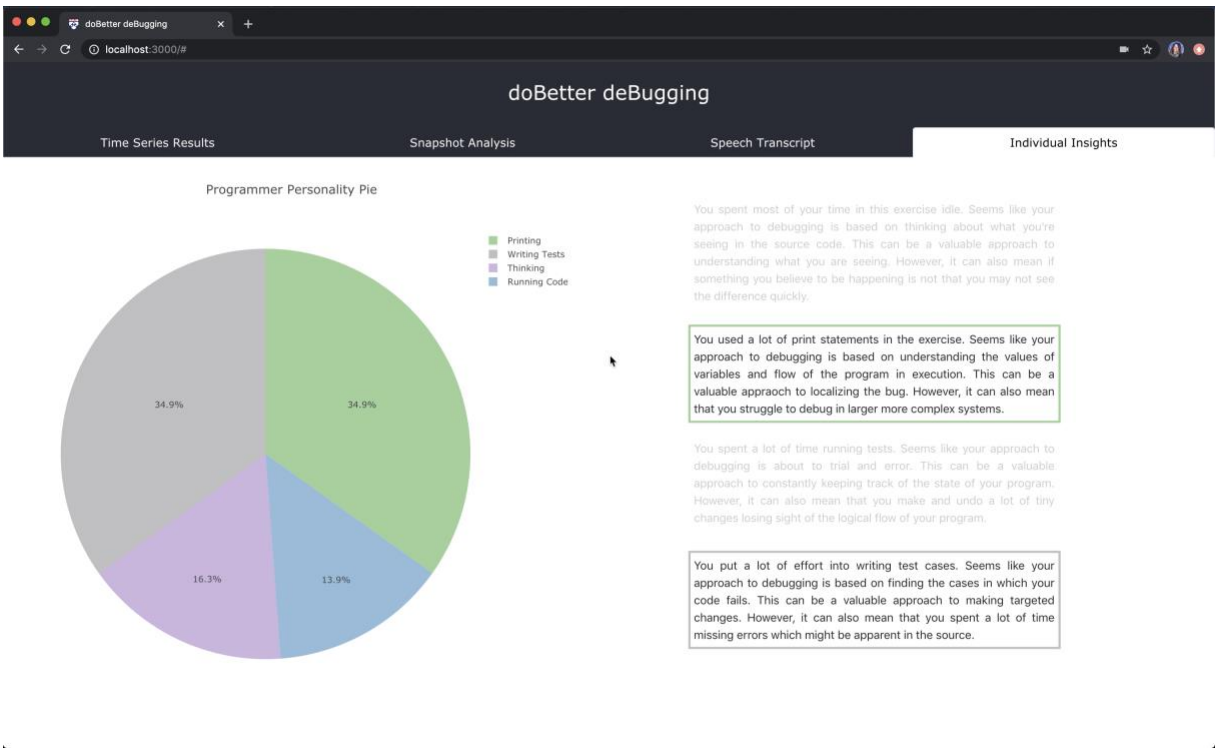
Appendix A3. Snapshot Analysis



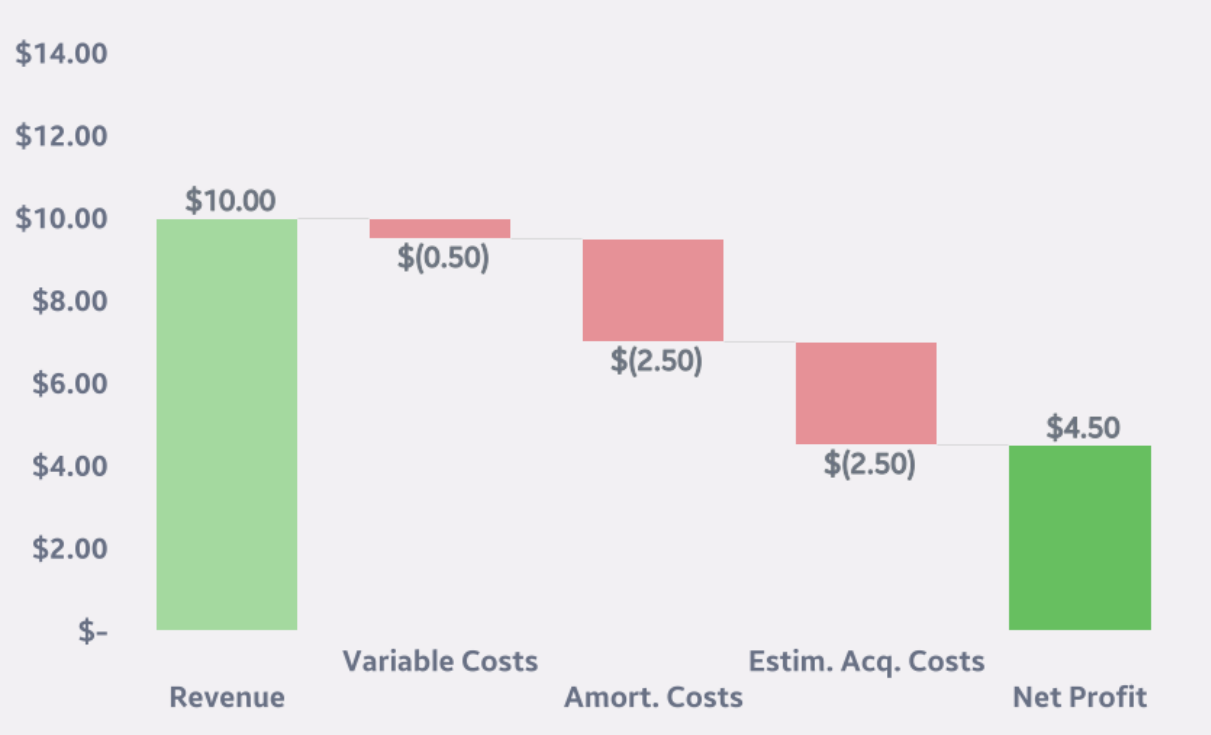
Appendix A4. Speech Transcript (premium)



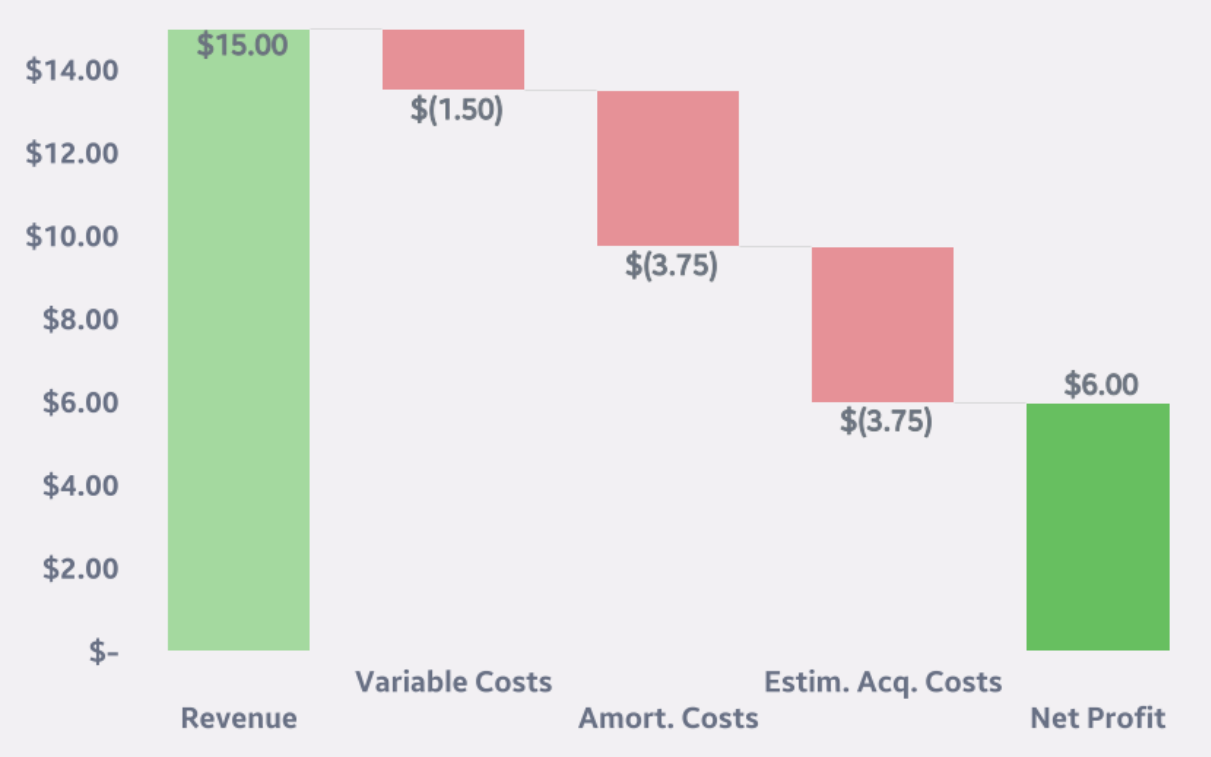
Appendix A5. Individual Insights



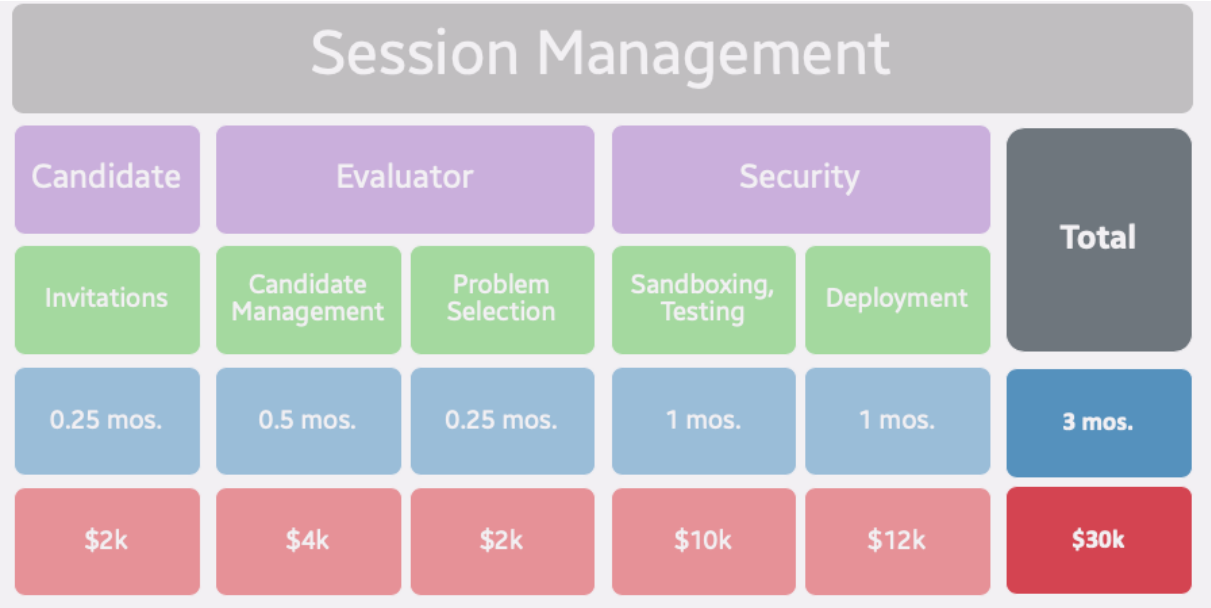
Appendix B1. Standard Unit Economics



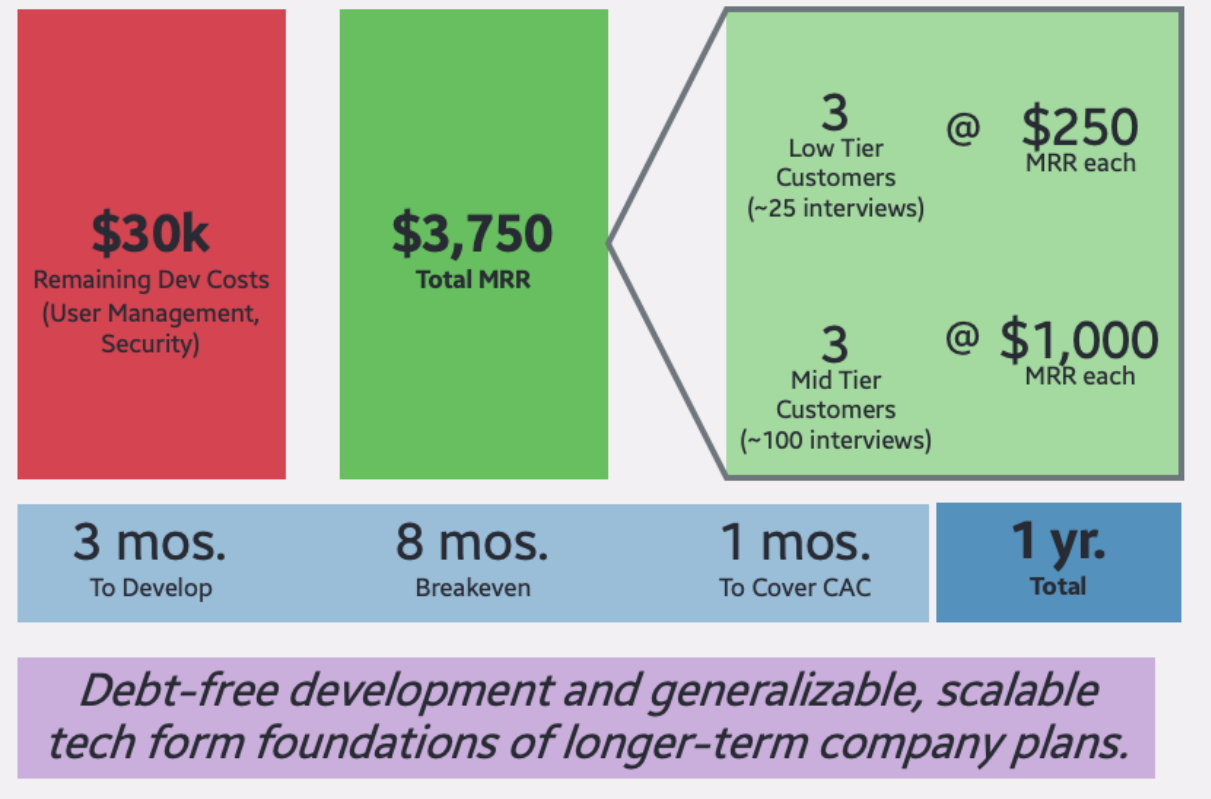
Appendix B2. Premium Unit Economics



Appendix C. Time and Cost to Develop Remaining MVP Features



Appendix D. Targeted Growth Strategy



Appendix E1. Full Candidate Survey

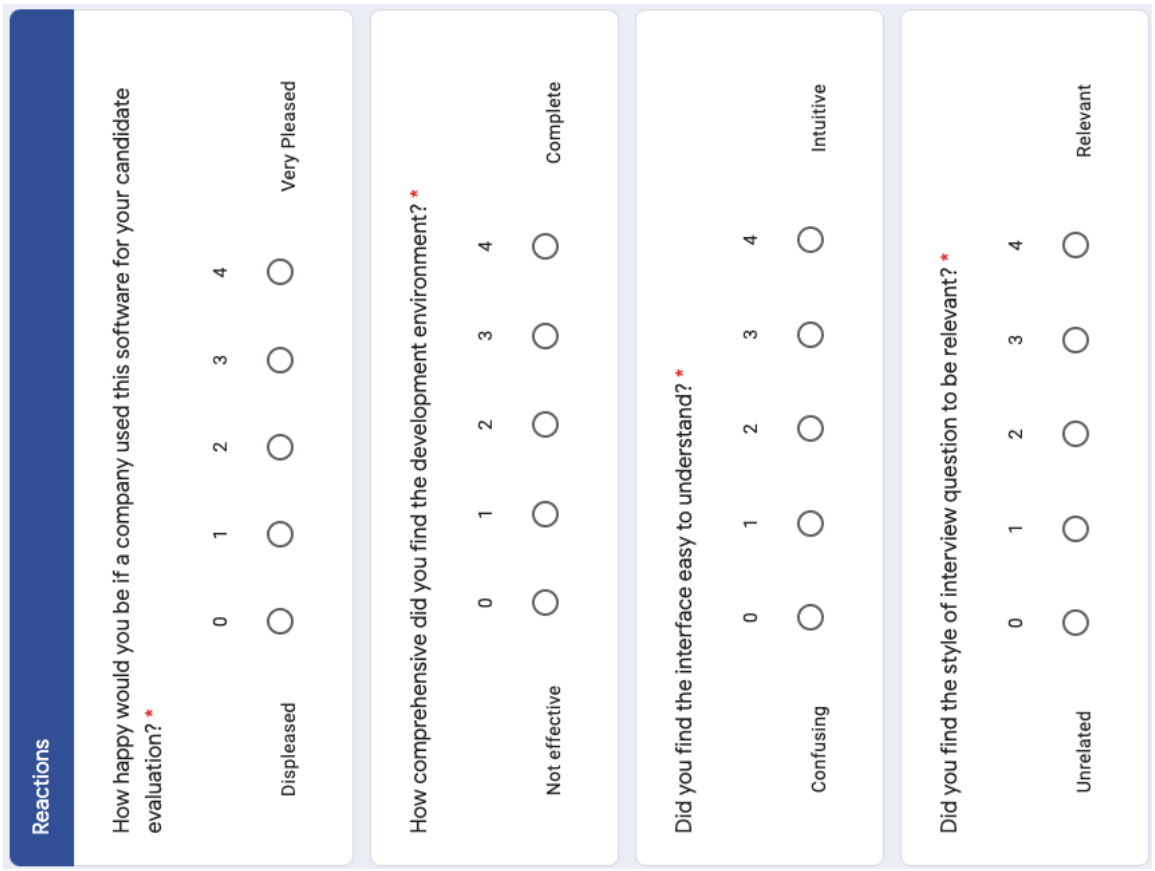
Feedback

What improvements does this demo make over your current interviewing process?
Why do you feel that way?
Your answer

What functionality seems to be lacking or missing?
Please include anything here that would make you not want to use such a product?
Your answer

Product aside, how do you feel about asynchronous candidate evaluation?
Your answer

Anything else you would like to say or let us know?
Really, anything at all!
Your answer



Reactions

How happy would you be if a company used this software for your candidate evaluation? *

	0	1	2	3	4	
Displeased	○	○	○	○	○	Very Pleased

How comprehensive did you find the development environment? *

	0	1	2	3	4	
Not effective	○	○	○	○	○	Complete

Did you find the interface easy to understand? *

	0	1	2	3	4	
Confusing	○	○	○	○	○	Intuitive

Did you find the style of interview question to be relevant? *

	0	1	2	3	4	
Unrelated	○	○	○	○	○	Relevant

Appendix E2. Full Evaluator Survey

Appendix F. Simulation Raw Data (times measured in seconds)

With Voice	**Simulation 1**	**Simulation 2**	**Simulation 3**	**Simulation 4**
Data Pipeline	162.03	84.13	0	16
Time Series	127.64	82.69	70	65
Snapshot Analysis	167.01	67.12	117	91
Speech Transcript	96.03	68.08	123	104
Individual Insights	19.63	30.83	6	6

Without Voice	**Simulation 5**	**Simulation 6**	**Simulation 7**	**Simulation 8**
Data Pipeline	2.3	1.48	0	0
Time Series	144.32	86.95	64	40
Snapshot Analysis	195.03	81.48	116	70
Speech Transcript	0	0	0	0
Individual Insights	10.02	24.99	6	6