Understanding the Relevance, Efficiency and Efficacy of Timed Coding Assessments in the Software Engineering Industry

by

Paola A. Leon Alarcon

B.S. Computer Engineering
University of Massachusetts Boston (2018)

Submitted to the System Design and Management Program
in partial fulfillment of the requirements for the degree of

Master of Science in Engineering and Management

at the

Massachusetts Institute of Technology

February 2024

Authored by: Paola A. Leon Alarcon
System Design and Management Program
January 19, 2024

Certified by: Donna H. Rhodes
Thesis Supervisor
Principal Research Scientist, Sociotechnical Systems Research Center

Accepted by: Joan S. Rubin
Executive Director
System Design and Management Program

This page is intentionally left blank

Understanding the Relevance, Efficiency and Efficacy of Timed Coding Assessments in the Software Engineering Industry

by

Paola A. Leon Alarcon

Submitted to the System Design and Management Program
on January 19, 2024, in partial fulfillment of the
requirements for the degree of
Master of Science in Engineering and Management

Abstract

Software engineering has grown as a market in the past decades to become one of the most profitable and widespread globally. Given the demand for software products, software engineers and their skills have also become highly sought-after. Even though there is great demand for software engineers, companies are looking to hire the best possible talent, forcing them to implement assessment mechanisms to evaluate candidates and their technical proficiency. Consequently, interviewing processes for software positions have become highly competitive and rigorous for prospective candidates. The volume of applicants has also forced companies to implement mechanisms that allow for screening candidates at an efficient cost. As a consequence, timed coding assessments and other technical interviewing methods have been raised as an alternative to screening candidates.

A survey was disseminated where participants were asked about their experience with timed coding assessments. Twelve volunteers willing to participate in a semi-structured interviewed were recruited from those surveyed with the goal of understanding their experiences in more depth. It was found that timed coding assessments can be an effective filtering tool to narrow the pool of candidates but did not show consistent relevancy with respect to the job duties and responsibilities software engineers might need to carry out if offered a position. Furthermore, preparation for these types of examinations was found to be fundamental for their clearance and further advancement into the interviewing stages, showing that qualification came secondary to preparation.

Thesis Supervisor: Donna H. Rhodes
Title: Principal Research Scientist, Sociotechnical Systems Research Center

Acknowledgments

This research was possible thanks to all the participants who were willing to share their experience in the software engineering industry with me. Special appreciation goes to the twelve wonderful participants who gave me an hour of their time to go over their stories, frustrations and breakthroughs in the software engineering space.

I would like to acknowledge my advisor, Donna Rhodes, for supporting this thesis research which has a very personal meaning to me. Dr. Rhodes was very patient in discussing ideas, providing feedback and guidance, and working with me through a very hectic schedule to make this work worthy of publication.

A very large special thanks to Joan Rubin for her support, not only with this work but for my entire wonderful experience during the System Design and Management program at MIT. It has been a life-changing experience to interact with three different SDM cohorts and absorb all the knowledge through classes and social events.

Finally, to my parents and husband Andres, for their unconditional support and for always believing in my potential, even when I doubted it the most.

Contents

Abstract **3**

List of Figures **8**

List of Tables **10**

1 Introduction **11**

1.1 Research Motivation 12

1.1.1 Personal Motivation 13

1.2 Research Questions and Approach 15

1.3 Research Scope and Limitations 17

1.4 Thesis Outline 18

2 Literature Review **21**

2.1 Origins of Software Engineering as a Discipline 21

2.1.1 The Origin of Computers 21

2.1.2 The Term "Software Engineer" 22

2.2 Development of the Software Engineering Industry 23

2.2.1 The Software Crisis 23

2.2.2 The Rise of the Internet 25

2.2.3 Modern Industry 26

2.3 Demographics 27

2.3.1 Education 28

2.3.2 Career Path 32

2.3.3 Common Skills Required . . . 35
2.4 Demand for Software Engineering . . . 36
2.4.1 Market Valuation . . . 36
2.4.2 Employment Reports . . . 37
2.5 Compensation and Pay . . . 39
2.6 Interviewing for a Software Engineering Position . . . 39
2.6.1 Technical Assessments . . . 40
2.6.2 Hiring Costs . . . 43
2.6.3 Preparation . . . 44
2.6.4 Pitfalls and Challenges . . . 45

3 Research Methodology and Data Collection 48
3.1 Survey . . . 49
3.1.1 Survey Content . . . 49
3.1.2 Dissemination . . . 51
3.1.3 Survey Data Analysis . . . 51
3.2 Semi-Structured Interviews . . . 52
3.2.1 Consent and Disclosure . . . 52
3.2.2 Semi-Structured Interview Content . . . 53
3.2.3 Process . . . 54
3.2.4 Interview Data Analysis . . . 55

4 Results 57
4.1 Survey . . . 57
4.1.1 Demographics . . . 58
4.1.2 Timed Coding Assessments . . . 59
4.1.3 Correlation of Interviews with Job Duties . . . 63
4.1.4 Metrics of Success . . . 64
4.2 Semi-Structured Interviews . . . 68
4.2.1 Demographics . . . 68
4.2.2 Experience with Timed Coding Assessments . . . 70

4.2.3 Experience with Other Interviewing Techniques 74
4.2.4 Correlation of Interviews with Job Duties 74
4.2.5 Managerial Perspective 77
4.2.6 Sentiment Analysis 81

5 Conclusions and Recommendations **82**
5.1 Conclusions 82
5.2 Observations 86
5.3 Recommendations 87
5.4 Future Work 88

References **90**

A Survey **94**

B Semi-Structured Interview Script **107**

List of Figures

1.1 Thesis outline 19

2.1 Levels of formal education for software developers, worldwide, as of 2023 [35]. 28
2.2 Revenue of the software market [40]. 37
2.3 Employment forecast for software engineering, applications [31, 30]. . 38
2.4 Average salaries of software developers worldwide, as of 2023, by role (in 1,000 U.S. dollars) [34]. 40

4.1 Job title of survey respondents [n=39] 58
4.2 Highest degree of education of survey respondents [n=39] 59
4.3 Years of experience of survey respondents [n=39] 60
4.4 Frequency survey respondents [n=39] encountered timed coding assessments 61
4.5 Industries of survey respondents [n=39] 62
4.6 Reported methods used to prepare for timed coding assessments [n=39] 63
4.7 Reported time used to prepare for timed coding assessments [n=39] . 64
4.8 Opinions on the efficiency of timed coding assessments from candidates [n=39] and managers [n=20] 65
4.9 Opinions on the effectiveness of timed coding assessments from candidates [n=39] and managers [n=20] 66
4.10 Survey respondents' [n=39] opinions on the correlation between interview assessment and job duties 67

4.11 Participants' [n=12] reported alternative methods for finding full-time employment 69
4.12 Participants' [n=12] exposure to timed coding assessments 71
4.13 Interview styles experienced by participants [n=12] 75
4.14 Correlation between interviews and job duties according to participants [n=12] 76
4.15 Preferred methods for assessing candidates according to participants [n=12] 78
4.16 Traits of a successful software engineer according to participants [n=12] 80

List of Tables

4.1 Opinions on skills evaluated in timed coding assessments 65
4.2 Manager opinions on skills necessary for a successful candidate 66
4.3 Education and years of experience 68
4.4 Sentiment Analysis 81

Chapter 1

Introduction

Software engineering has grown as a market in the past decades to become one of the most profitable and widespread globally. Given the demand for software products, software engineers and their skills have also become highly sought-after. Even though there is great demand for software engineers, companies are looking to hire the best possible talent, forcing them to implement assessment mechanisms to evaluate candidates and their technical proficiency. Consequently, interviewing processes for software positions have become highly competitive and rigorous for prospective candidates. The volume of applicants has also forced companies to implement mechanisms that allow for screening candidates at an efficient cost.

Many methods for assessing candidates have appeared, such as timed coding assessments, take-home exams, live coding assessments, and verbal interviews. Each of these methods differs in form, but they all look to evaluate if a candidate can technically excel in software engineering duties if offered a position at the organization.

Timed coding assessments tend to be widespread across larger organizations, where reviewing each candidate with a phone screen or interview is virtually impossible. As a type of asynchronous assessment, meaning there is no examiner or representative of the company present, timed coding assessments consist of a series of problems the candidate needs to solve using computer science fundamentals. They are usually time-constricted and equipped with test cases to help candidates assess their solutions. The candidate can expect to move to the next interviewing round if

all test cases pass.

A great emphasis is given to technical skills during the interview process, but soft skills seem to be an afterthought in academia and some interview settings. Candidates are expected to endure rigorous preparation. As a consequence, online communities have emerged to provide resources for candidates. Practicing platforms, books, courses, and forums have been popularized to help software engineering candidates prepare best for the interview process at notorious firms.

A research study was conducted to understand timed coding assessments better, their relationship with day-to-day job responsibilities, and managerial and candidate opinions surrounding this screening method. A survey was composed where participants could answer simple questions about their experience with timed coding assessments and important demographic data points such as education, years of experience, and industry within software engineering. Those interested in discussing their in-depth experiences indicated they would be willing to be interviewed. Thirty-nine participants answered the survey, and twelve agreed to be interviewed. Using a semi-structured interview script, the twelve interviewees were asked about their job history, how they were assessed for technical roles, and their overall experience when interviewing for software engineering positions.

1.1 Research Motivation

Interviewing for software engineering has gained popularity in the last decades, for better or for worse. The rise of the market and the profitability of the technology sector, potentiated by the development of algorithms and logic, has attracted professionals inside and outside of the tech industry, given the high salaries and prestige of companies.

An important observation of this phenomenon has been the popularization of platforms that prepare candidates for interviews. A business opportunity emerged, allowing platforms to capitalize on the participation of candidates by offering preparation plans catered to each individual's personal goals. Noticing the rise of this

emerging market that has exploited the difficulty and intricacy of technical interviews was a compelling motivation to understand the interviewing practice in more depth, and determine if it can actually help to recruit successful candidates.

Markets originate from consumer needs, and software engineering candidates' shared frustration gave space for these communities to emerge. Other markets have originated due to consumer needs but also due to financial incentives: it can be the case that processes and products do not contribute to a need and do not actually solve a problem, but the environment and market originate due to demand. Consequently, processes can fail to improve, following the established trends that have allowed some to capitalize on inefficiency. Are technical interviews, and asynchronous assessments in particular, actually necessary? Or is the market and demand too high to encourage organizations to use a method that might be outdated and might not contribute to recruiting successful talent?

1.1.1 Personal Motivation

My personal motivation for this research stemmed from my experience interviewing for software engineering positions. As a young professional and fresh out of school, I noticed how the curriculum I had at university did not seem enough to clear software engineering interviews. That is when I understood I needed to prepare, outside academic activities, to find a software engineering position that would allow me to grow professionally. My undergraduate studies and experience at school did not seem to be enough. This experience did not change as I gained experience in the field: every career movement had a lot of friction. It required great dedication and practicing time to prepare just for the interviewing process.

My first job as a professional engineer did not require an asynchronous evaluation to assess my expertise in software engineering. My evaluators at the time thought it was more effective to assess my skills with questions, inquiring about past projects I had worked on that were relevant to them, and focusing extensively on my communication abilities. I was offered a full-time position at this organization and spent five years working with them. Each year, I was praised for my excellent work ethic and

ability to make a change, implement successful software processes, and develop state-of-the-art software technology. A coding assessment may or may not have predicted this success.

I discussed my experience with fellow software engineers and work colleagues to find they felt the same way: somehow, we had excellent job performance, but could not clear some of these assessments if we wanted to pursue different opportunities. Other engineering colleagues seemed to have challenges navigating the job market but could rely better on their experience and projects during interviews. My friends in biotechnology and mechanical engineering positions seemed to have less friction moving between jobs. They were not requested to demonstrate their technical skills on the spot like myself and other software professionals. Why is software engineering different?

Further inspiration for this research came when it was time to expand my engineering team. I was set to find a successful software engineer to join our organization, and thoroughly meditated on which assessment method I would use to determine if they were technically proficient. However, my judgment would be biased on what I thought would be an excellent method to assess my technical skills. Finding an objective assessment method seemed a difficult task. Regardless, I hired a software engineer for our team that provided measurable results. Through a verbal assessment and placing importance on the credibility of their degree and previous experience, I felt I had made the right decision. Still, the method was neither formal nor structured. Perhaps I was lucky to find a candidate with good communication and technical skills, and the interview method had nothing to do with successfully evaluating this candidate.

Just as we like to engage in software development through different means, engineers want to be assessed through different methods, perhaps with those that showcase our strengths more than our weaknesses. Not all of us are good in the same structured evaluation methods. Some prefer more time and access to resources, some like being challenged and time-pressured to complete a problem, and others prefer the conversational component a live coding assessment might bring into the mix. We can achieve a diverse technical workforce only through different perspectives and ex-

periences. Understanding other fellow engineer's opinions, not only those in my close circle, was very motivating. Understanding if I was the only software developer to see the issues in hiring motivated me to look for further information and reasoning into why the process for interviewing engineers came to be this.

Finally, while synthesizing the results from the survey and interviews, I engaged in a hiring process, as a candidate, for a software developer position at a large technology firm. I knew I had to prepare to pass this interview, and even though I was not successful at clearing specific components, a timed coding assessment, the hiring manager saw the potential in my profile and adapted the interview process to fit my needs better. Through a live coding assessment, I could showcase my technical proficiency better, ultimately allowing me to proceed further in the interviewing process and eventually get the position. In my experience, a hiring manager had never taken the time to adapt a process, giving credibility to my previous job experience.

From the people I interviewed, I noticed similar experiences, where the process is highly driven by organization representatives, either for better or for worse. I realized the process needs improvement, and even though we have made significant progress, much remains to be sorted out. Hopefully, the research questions posed by this thesis can help contribute a piece of the puzzle in this systemic issue, where components affect each other on a bigger scale.

1.2 Research Questions and Approach

When composing the survey and semi-structured interview script, the goal was to allow participants to speak freely about their experiences, both with interviewing and on their jobs. Some research questions were vacillated before the dissemination of the survey, but it was not until interviewing volunteers that essential research questions were raised, including the following:

- Are technical interviews, specifically timed coding assessments, correlated to the day-to-day responsibilities of the position they are being evaluated for?

- What outlook do software engineering candidates have on technical interviews?
- Do candidates and managers have different perspectives on the efficiency and effectiveness of technical interviews, specifically timed coding assessments?

These questions are comprehensive enough, allowing partial or complete answers using this thesis's data collection and analysis methods.

There is specific interest in understanding if participants believed that timed coding assessments and overall technical interviews were fair and representative of the job duties encountered later on once they accepted their job offers. In other words, are the assessments overly complicated or simplified for candidates compared to their job duties? Simple questions were asked to assess the correlation of assessments with job duties, allowing participants to rank their views using five varying degrees of agreement and disagreement in the survey. During the interview, participants shared more details about this statement, outlining the relevant and non-relevant parts of their interview process to their job duties encountered later on.

Another important aspect was to question participants on their opinions about success in the software engineering field. Although very subjective, the results can be used to analyze industry trends and company needs that drive software engineering hiring. Furthermore, these skills can be compared to what timed coding assessments and technical interviews aim to measure, to understand if hiring managers and companies are using the most efficient and effective tools to drive their hiring processes. By answering questions during the interview portion and ranking abilities in the survey, participants showed what they considered essential skills for a successful software engineer and to what level these are assessed during timed coding assessments, if any.

Finally, the last question aims to understand if there is a bias between managers and candidates when qualifying if timed coding assessments are efficient and effective in hiring qualified software engineers. In other words, since candidates must endure a rigorous preparation process and clear a series of tests, their opinions might differ from those of managers. Managers also might hold strong views about assessments due to their position, where diminishing risk and ensuring a good hire are fundamental.

For this question, participants in the survey were asked to report if they had hiring experience, allowing them to unlock an extra series of questions in case they did. The answers to this portion of the survey were compared to those reported by the participants without hiring experience on the survey, with the aim of outlining the differences, if any.

1.3 Research Scope and Limitations

It is essential to understand the research scope and limitations of this thesis. The scope of this thesis was initially to study timed coding assessments only, without venturing into other assessment methods. However, as the interviews progressed, it was inevitable to notice much information about technical interviews, including other popular methods. Consequently, this research also ventures into those methods, their efficiency, and efficacy, but focuses primarily on asynchronous assessments.

The research was carried out with the hope of obtaining answers from all across the globe, allowing participants to report where their work experience had taken place. Upon analyzing the survey results for this question, almost 90% of participants expressed working in North America. The dissemination channels used for the survey might have contributed to this issue, something that could be improved in further research. Consequently, it is fundamental to notice that the answers from this research will be heavily influenced by North America's customs and working culture. A more comprehensive research that includes professionals from all continents in a more balanced ratio might yield different results.

Another limitation of the research is the sample pool size. Thirty-nine participants filled out the survey, and twelve agreed to be interviewed. Better dissemination strategies could have improved the geographical distribution of candidates and the sample size. Even though the number of participants sufficed to draw conclusions and observations, it could be assumed a more extensive sample pool could be more representative of the population.

The profile of the candidates interviewed poses another limitation to the research.

Having more variety on this aspect could have provided further insights into what drives the opinions of candidates and managers about timed coding assessments and other interviewing methods. Considering a more varied population in terms of the highest degree obtained, years of experience, and successful outcomes with timed coding assessments could provide more specific results, helping to expand the research questions.

A challenge experienced during the survey's conception was selecting the appropriate language for the outlined concepts. The software engineering industry does not have a standard and therefore does not name consistently the types of methods used during interviews. Consequently, definitions were provided to the survey and interview participants to avoid ambiguity in the terms used during the research process.

Finally, bias in the data analysis could be considered the last limitation of this research. Qualitative and sentiment studies were conducted for semi-structured interview scripts, but minimizing biases during this process was fundamental. However, the tool used for sentiment analysis abstracts the specifics of how speech is categorized, making it hard to quantify any algorithmic bias. Similarly, these natural language processing tools have difficulty handling sarcasm and detecting context in the conversation, diminishing the confidence level of speech categorization.

1.4 Thesis Outline

This research is structured across five chapters, as outlined in Figure 1.1. Chapter 1 outlines the introduction, motivation, research approach, and limitations.

In Chapter 2, fundamental events and concepts surrounding the birth of software engineering as a mature industry are discussed, as well as demographics of the software engineering industry, educational paths and compensation for software engineers, and descriptions of the rigorous interviewing process engineers need to endure to secure a position in the field.

Chapter 3 outlines the research methodology and data collection methods used for

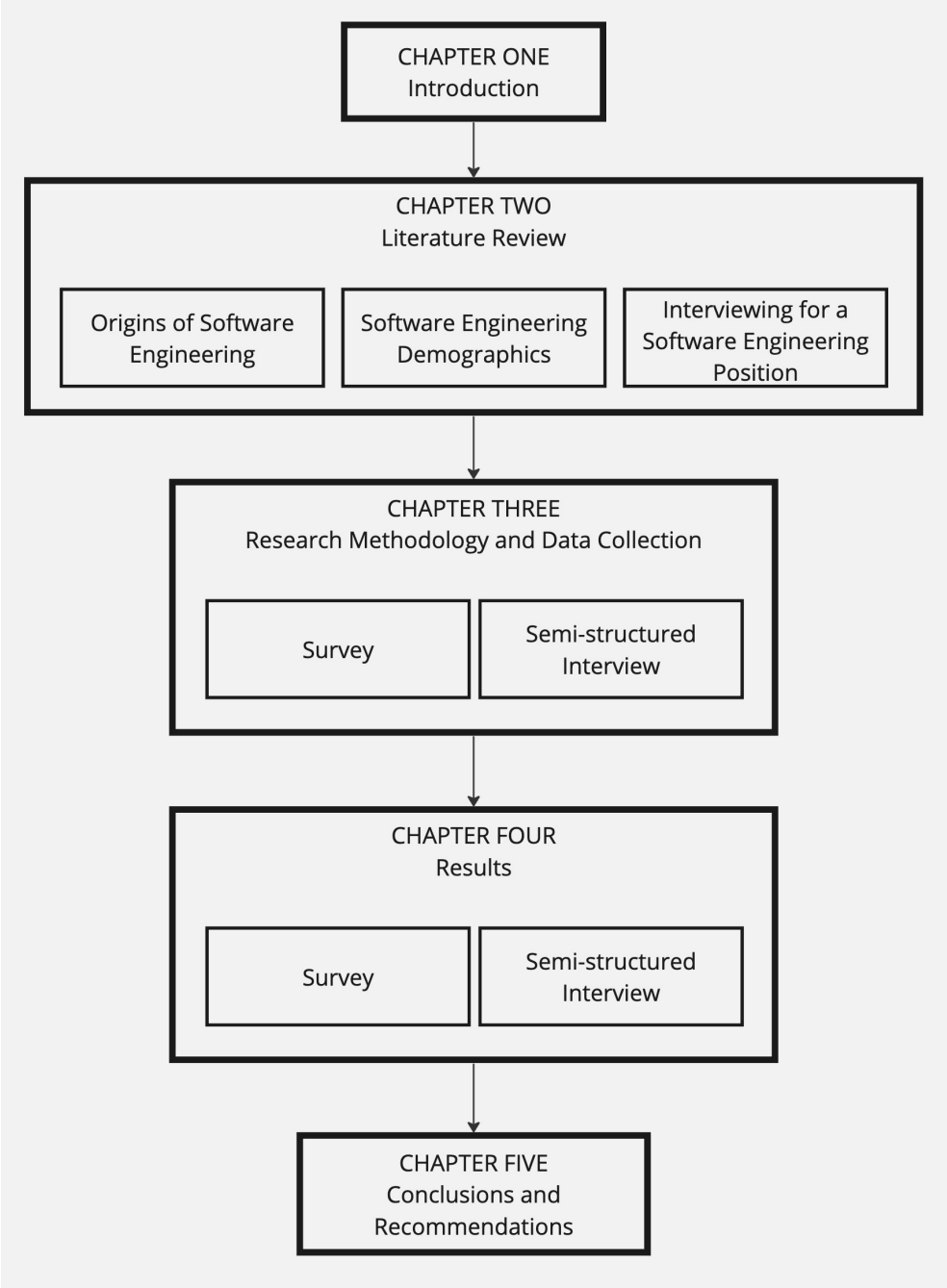


Figure 1.1: Thesis outline

this research, which include a survey and a semi-structured interview. The conception and dissemination processes for each data collection tool are described in detail, as well as the analysis methods used for this investigation's quantitative and qualitative components.

Chapter 4 synthesizes the results from the survey and interview using the methods

described in Chapter 3.

Finally, Chapter 5 summarizes critical research findings and answers the research questions outlined earlier in Chapter 1, provides recommendations that could contribute to resolving the issues outlined by participants of the research, and describes future work that could contribute to improving the hiring process of software engineers across different industries and countries.

Chapter 2

Literature Review

Research related to technical interviewing and timed coding assessments can be contextualized with information about the origin of software and facts related to the industry in general. The following chapter also outlines previous work done in recruiting to understand, among many things, the cognitive load of software engineering interviews and the relevancy of academic curricula.

2.1 Origins of Software Engineering as a Discipline

The emergence of software engineering as a discipline and profession is essential to understand and contextualize. Software engineering is a relatively recent practice that has allowed to, among many things, automate processes, allowing humankind to reach goals that were thought unimaginable, such as space exploration, improvement in communication methods due to the invention of The Internet, and the usage of data as an essential tool for decision making.

2.1.1 The Origin of Computers

A conceptualized idea for the invention of the first computer first originated in the 1820s, given the latent need of automating calculations [6]. These ideas matured, allowing the creation of difference and analytical engines. Still, it was through the

creation of the Turing Machine, a universal computing device, in 1936, that an infinite range of particular applications for the device became available [27, 6]. Turing's ideas were refined and iterated through the greater part of the 20th century, allowing John Von Neumann to pioneer the classic architecture users know today, where programs can be stored in the computer. [6].

The computer's use would depend on the expectations and experience of those who would burden the responsibility of its use or of the design for others to use [27].

2.1.2 The Term "Software Engineer"

A set of conferences hosted during the late 1960s by the North Atlantic Treaty Organization (NATO) kick-started the discussion of the term "Software Engineer", where the term was discussed as a sub-discipline of computer science [27, 36]. The term was also popularized by Margaret Hamilton, an engineer responsible for the Apollo Mission's guidance computer implementation[16]. The Apollo mission landed astronauts on the moon, signifying a monumental step in the space race during the 1960s.

Engineering disciplines are based on theoretical foundations and principles, so software engineering was chosen as a provocative term that implied the manufacturing of software to be based on said principles [27]. However, multiple questions arose, including if software, holding an immaterial principle, could even be manufactured [27].

Even with an unclear definition, software engineering was deemed necessary in a young upcoming industry [27]. The design of mechanical calculators and development of mathematical logic became fundamental in the creation of what is known today as the modern computer and deemed an important task; hence, naming the discipline appropriately to represent the importance of this work was necessary [27, 16]. The convergence of calculators and logic showed that the computer industry emerged from the combination of mathematics and electrical engineering, two industries that by that time had already been well defined [27].

2.2 Development of the Software Engineering Industry

With the commercialization of computers, hardware progressed immensely: better processors, memory, and peripherals became available [27]. By 1968, companies were already producing software, given that data processing divisions were given more importance [27]. Selecting and training engineers was challenging, given no standard and no general specifications for the software engineer role [27].

Schools of thought on how to approach programming arose during the early times of the industry, given a plethora of new challenges. Programming started to become an experimental science or engineering even more, yielding trends such as structured programming, extreme programming, and agile software development, just to name a few doctrines [27].

2.2.1 The Software Crisis

The software crisis occurred when problems were pointed out in software development. Projects were constantly running over budget and schedule; some projects even caused property damage [27]. Managing software and its accompanying methodologies became more of an issue as time progressed [43].

Ambitious projects were undertaken with little preparation on how to face potential budget overruns. One example was the IBM OS/360 project, where IBM attempted to design a simple operating system that would run on all of their mainframes in the first half of the 1960s [10]. The staff struggled greatly with this massive project, forcing IBM to hire more developers to assist. The product was delivered, but late and over budget[10].

Lack of calendar time was responsible for the failure of software projects, partially due to poor estimating skills, equating effort with progress, poor schedule monitoring, and scope creep [7]. Although a great motivator, optimism could and can be another cause that fuels these issues. Assuming that more engineers equated to more progress

was also a fallacy since many software tasks cannot be distributed and, therefore, do not impact the schedule for delivery favorably.

Property damage and accidents generated by poor software system implementations also caught the attention of the software engineering community during the software crisis. The Therac-25 accidents took place between 1985 and 1987, when six people overdosed during radiotherapy sessions, causing severe injuries and deaths [26]. These accidents were attributed to not following appropriate procedures in the creation of the system and the implementation of software [26]. The intricate interaction of software with other systems needed to be revised.

During this time, the ideas of Frederick Brooks, in his paper titled "No Silver Bullet," were echoed by many in the community. Brooks argued no single magical solution could solve the software crisis [8]. Some suggestions included the exploitation of the market to avoid implementing what could be bought; in other words, not reinventing the wheel, and taking advantage of prototyping as a means to iterate over software, allowing its organic growth and development as new needs were identified [8].

Software can seem innocent and straightforward, but the probability of missing schedules, blowing budgets, and producing flawed products is relatively high [8]. Aiming to reduce the costs of software, mimicking the behavior of hardware costs was the desire back then, but the progress was noticed to be significantly slower, unnerving the community [8]. Avoiding solving the same problem twice was a critical idea discussed, but to achieve this, established disciplines such as configuration management, coding standards, and naming conventions were necessary [43].

The rapid evolution of software and technology made it particularly difficult to catch up, and critics of the software crisis vacillated the idea of waiting until the industry was more stable to develop disciplines and best practices that addressed the problem [43]. Improving the rate of progress in software required examining the difficulties and interlocking concepts used in the discipline, such as data sets, relationships among data items, algorithms, and invocation of functions [8]. Quality of software was measured in customer satisfaction, minimization of error rates, performance and

extendability, and reusability of other product characteristics [43].

2.2.2 The Rise of the Internet

The rise of the Internet had a significant impact on the software engineering industry [42]. New businesses for company processes related to product development, distribution, and support were created as a consequence [42]. The transition from mainframe to minicomputers, personal computers, and finally local networking granted client-server applications to exist [42, 6].

Prior to the Internet, software could be categorized into three main slots: enterprise, consumer, and shareware [42]. Enterprise products are those leased to organizations that aid with their productivity and maintenance services, while end-users use consumer products and shareware and run on local machines without making connections to other machines or external networks [42].

Traditional desktop applications vastly differ from client applications, and the languages and tools required for their development differ too [42]. New classes of applications have appeared, including hosted applications, originally termed Software as a Service (SaaS), requiring no installation from the user's end [42]. Email, personal productivity, office automation, customer relationship management, software development, online communities, e-commerce, telephony and conference, and multiplayer games are some of the sub-industries within software that originated after the arrival of the Internet [42].

The Internet eased access to collaboration, facilitating distributed organizations with tools for that purpose, making information accessible to billions of people [42]. Free and Open Source Software (FOSS) was another originated class, which included primarily non-commercial software for a broad range of applications and mainly focused on software development and system infrastructure management [42].

The release of products, hosting web applications, and selling products at lesser costs were some of the impacts the Internet brought to the industry, since, before its creation, software was usually delivered in physical form via tape, floppy disks, or compact disks [42]. How organizations interact with consumers also changed,

including marketing strategies [42].

Advances in mobile and cloud computing gave room for more innovation in products, making the industry grow [42]. Innovative ideas no longer required significant capital investments to provide services or careful thinking about over or under-provisioning, as cloud computing was based on hosting computer facilities and storage utilities with a third-party provider [17]. The introduction of smartphones and the Internet of Things (IoT) also redefined the meaning of a host in the Internet, not as any computer but as any device with networking capabilities [32]. Applications available in mobile devices were an entirely new category, giving room to ecosystems and platforms to originate and allowing people to sell their customized applications [42].

2.2.3 Modern Industry

Software remains a relatively soft concept. The definition has not changed much since the NATO conferences at the end of the 60s and the beginning of the 70s. The definition remains as "the disciplined application of engineering, scientific and mathematical principles and method to the economical production of quality software" [27, 22].

Brooks' publication continues to be relevant even in the modern industry today, where many of the same challenges continue to be faced [8]. Although the industry has progressed and become more productive, some of the challenges inherent to software development remain: specifications, design, and testing of a conceptual construct [8]. In addition, software properties such as complexity, conformity, changeability, and invisibility fuel these challenges [8].

The development of high-level languages, time-sharing, and unified program environments helped with some of these challenges, but there was and is still more to do [8]. The modern industry catapulted additional solutions, such as the evolution of previously defined high-level languages, object-oriented programming, artificial intelligence, expert systems automatic programming, graphical programming, program verification, environments and tools, workstations, buy vs. build, requirement refin-

ing and rapid prototyping, incremental development-grow, and great designers [8]. Back during the software crisis and even today, no silver bullet is available to deal with the intricacies and challenges of software development.

With the growth of the industry, code bases have followed suit, becoming more complex and increasing the workload on software engineers [32]. Being intangible, software does not obey a physical law, making it rely on good practices rather than fundamental theories [32].

As the discipline has evolved, the software industry has emphasized the importance of human factors, including communication and teamwork, to deliver quality products successfully [32].

2.3 Demographics

Software engineering remains male-dominated, where the median age is 43 years old, where 61% of developers are younger than 35, according to a 2021 demographics survey [32, 33]. Other exciting areas highlight that 11.5% of developers surveyed identify as something other than heterosexual and 1% of professionals identify as transgender[33]. According to the same 2021 survey, there also continues to be evidence that minorities are underrepresented since 58% of those surveyed identified as white or of European descent [33].

Technology workers are often stereotyped as being socially awkward or having difficulty communicating with others, and software engineers are not the exception [28, 32]. Many technology workers with atypical cognitive profiles are heavily affected by such stereotypes [28]. Many people with Autism Spectrum Disorders (ASDs) have an interest and affinity for technology but can often face challenges finding employment or even discrimination in the workplace [28]. During 2021, ten percent of surveyed developers admitted dealing with anxiety, and another 11% self-identified as having concentration and memory disorders, and having ASDs [33]. However, companies are trying to hire more diverse workforces, not only as a matter of social justice but to take advantage of affinities between the profile of individuals that are

neurodivergent and the job requirements of the technology industry [28].

2.3.1 Education

Academia has not been able to keep up with the rapid development of the software industry, given the fundamental need to adopt new technologies and the evolution of best practices [32]. The increased importance of proper software knowledge on how to build it has resulted in a new need for software engineers to be educated on how to design, build, test, and maintain software products [36]. Online resources have become very valuable for those looking to learn programming skills. In 2021, almost 60% of developers acknowledge they learned to code using online resources [24, 33]. Regardless, formal education still prevails, showing 41% of developers in the U.S. have a bachelor's degree and 21% have a master's degree, as shown in Figure 2.1 [24].

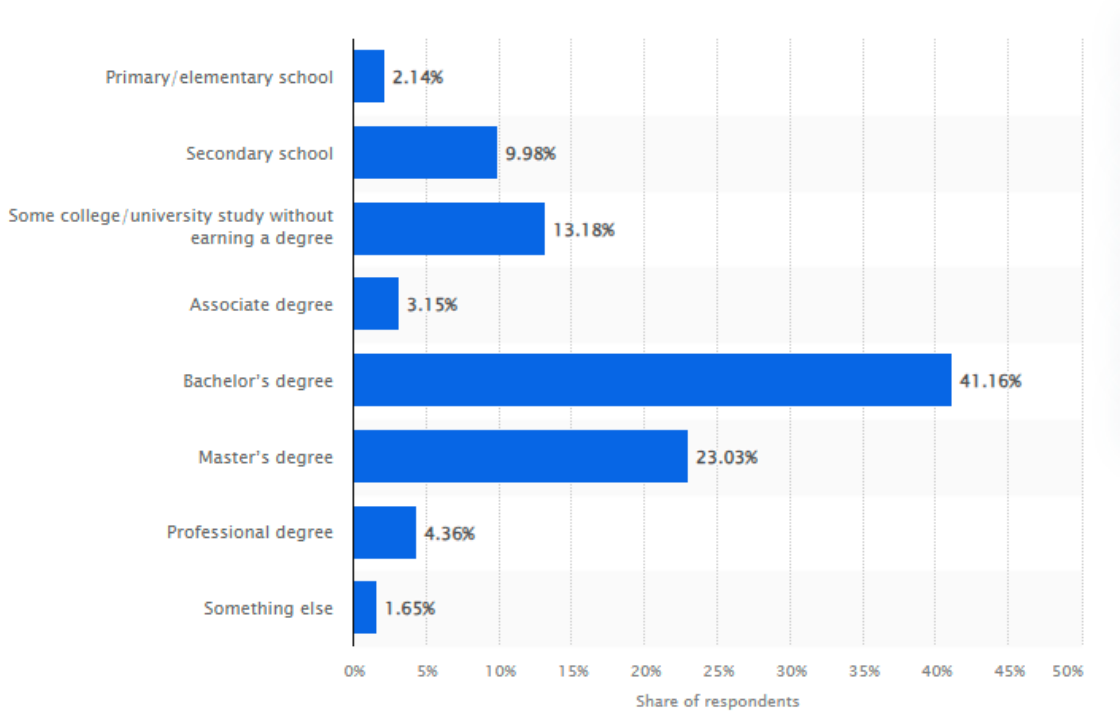


Figure 2.1: Levels of formal education for software developers, worldwide, as of 2023 [35].

University Educational Programs and Curriculum

The U.S. Bureau of Labor Statistics, an agency of the Department of Labor, is responsible for finding facts in the field of labor. They agree that formal education still prevails, stating that the typical entry-level education for software developers, quality assurance analysts, and testers is a bachelor's degree [31]. However, no work experience in a related occupation and on-the-job training is necessary, according to the same body [31].

Historically, software engineering has been discussed as a subfield of computer science [36]. Computer science focuses on automata theory, language design, operating systems, theorem proving, and software engineering, among other sub-disciplines [36]. The standard computer science curriculum courses include compilers, database systems, algorithms and data structures, and software engineering concepts [36].

The style of education varies across formal university programs, and these disparities originate from the different career goals and interests a student might have [36].

Just as electrical engineering is considered to be derived from physics, a similar analogy can apply to software engineering and computer science [36]. There is a basis for offering degrees and programs in each discipline, as engineers usually focus on building and scientists focus on the theoretical aspect and learning, emphasizing that the career path of choice should drive this decision [36].

Science focuses on what is accurate, how to confirm and refute models of the world, and how to extend that knowledge. In contrast, engineering focuses on what is true and its usefulness in their chosen specialty [36]. Applying the body of knowledge and how that can apply to a broader area allows building products that will function in realistic environments [36].

One of the challenges institutions face is creating realistic classroom experiences that will be translatable and applicable to industry [32]. Students are expected to complete their education, develop personal projects, contribute to free and open source, and acquire freelance jobs from customers to obtain more relevant experience

that qualifies them as well-rounded candidates [32].

The Software Engineering Body of Knowledge (SWEBOK) describes generally accepted knowledge about software engineering. The guide is imagined as a living and changing document that continues to change and update and has been created by leading authorities, reviewed by professionals, and disseminated for public review and critique [39]. It is considered the most authoritative, fundamental, and trusted definition of the software engineering profession [39].

SWEBOK aims to characterize the contents of the software engineering discipline, promote a consistent view of the practice worldwide, clarify and set the boundary between software engineering and other disciplines, provide a foundation for training materials and curriculum development, and provide a guideline for certification and licensing of professionals in the software engineering field [39].

Degrees aim to obtain accreditation from agencies to increase potential students' credibility and the quality and quantity of graduates who will be well prepared to excel in the industry after graduation [32]. Accreditation has become more critical in the U.S. and Canada, given that legislation might limit those who can practice engineering outside of software engineering [32]. The Accreditation Board for Engineering and Technology (ABET) has substantial requirements to bless programs, making ABET visits a significant event on college campuses [36]. They aim to raise the quality of educational programs, ensuring students are exposed to essential ideas and know how to use them [32].

These accreditation agencies, in combination with laws on higher education and regulations, make it difficult and slow to adapt curricula [32, 36]. Complaints about the curriculum not equipping students well enough are popular. Still, it could be considered a combination of the lack of adaptability from programs and the constant advances in technology and industry trends. Summer internships provide an excellent opportunity for students who might be able to minimize the gap between education and industry [36]. Course projects are not considered realistic since their scope is too minimized to apply in the field once students graduate [36].

Another observation is that software engineering can be showcased as a sociotech-

nical practice. Although the profession closely ties computer science and computer engineering, soft skills are widely acknowledged but rarely emphasized in university programs [32].

Turning the profession into a more social one would also benefit the eradication of the stereotype that benefits introverted men in the software industry [32]. Having dedicated classes allowing more human interaction and the involvement of teaching assistants or experienced programmers to showcase realistic team dynamics and operations could greatly benefit students [32].

Programming Boot Camps

Individuals with non-technical backgrounds or non-software engineer students can get up to speed with their software engineer counterparts through popular programming courses, often called boot camps. Fundamental programming knowledge is taught, including object-oriented programming, data structures, database basics, and data-centered programming [29]. As the demand for non-software engineers to learn software skills increases, the popularity of boot camps follows suit [29].

In addition, with college tuition on the rise, boot camps offer an efficient and affordable option to those looking to transition into more technical careers or pick up new skills [19]. Boot camps still require a significant commitment of time, money, and effort, and even though they are not a substitute for a college degree, they can open up plenty of opportunities in the tech field [19].

Expected benefits provided by boot camps are the reduced cost of tuition, assistance navigating the job market, lack of previous educational requirements, and versatility in the curriculum offered and skills taught [19]. According to the National Center for Education Statistics, the average cost of a programming bootcamp averaged $10,762 in 2022, compared to the average of $9,349 for a single semester in a traditional in-state school and $27,023 in out-of-state schools [19, 14]. The cost of formal education becomes even more expensive, considering at least four semesters are necessary to complete an associate's degree and eight semesters are necessary for a bachelor's degree.

Different boot camps also offer different educational styles that could adapt to the routine of students. Synchronous and asynchronous formats are available, where learning requires attending classes simultaneously as other students and learning is self-paced, allowing students to list and complete lectures at their own time, respectively [19].

2.3.2 Career Path

The U.S. Bureau of Labor Statistics categorizes the following occupations as similar to software development: computer and information research scientist, computer and information systems manager, computer hardware engineer, computer network architect, computer programmer, computer support specialist, computer systems analyst, data scientist, database administrator and architect, information security analyst, mathematician and statistician, post-secondary teacher, web developer and digital designer [31].

When it comes to software development, engineers have a broad range of options they could focus on, depending on their interests and the employer's needs. It is essential to notice that the skills needed for each subcategory vary greatly, as do the salary and compensation [42, 33]. Eighty-one percent of professional developers were employed in 2021, including independent contractors, freelancers, or self-employed individuals [33]. Common career paths within software engineering are described below, although many other options are available for those looking for software engineering careers.

Front-End

Front-end engineers specialize in the client side and user interface of applications. Their responsibilities include planning, designing, building, and implementing the user interface system of websites, software programs, or web-based applications [18, 41]. The primary goal is to provide a satisfactory user experience with minimized issues, errors, and downtime, as well as focus on the accessibility and browser com-

patibility aspects of the product [18, 41]. Front-end engineers serve as intermediaries and bridge the gap between the user and the logic of the program [18, 41].

Common skills for front-end developers include proficiency in programming languages such as JavaScript, CSS, and HTML and expertise in user interface design and frameworks such as React [41].

Back-End

Back-end engineers specialize in the server side of applications, often including programming logic and data storage. They are responsible for curating the server-side information structure, including but not limited to server scripts and application programming interfaces (API) to be utilized by front-end engineers and user experience designers [41].

The logical side of applications needs to be optimized for speed, stability, and security [41]. Back-end engineers must implement security structures, generate reusable code libraries, and generate storage solutions to deal with these properties efficiently [41].

Full-Stack

Front-end and back-end categories do not always have a clear and striking distinction, and employers often expect engineers to have some basic knowledge about the other side [41]. Enter the full stack developer, the jack of all trades, which allows better cross-functionality between teams [41].

Full stack developers tend to have more opportunities since they have fluency in both significant components of the application: user interface development and logic [41]. Companies starting up or facing budgetary constraints may opt for full-stack engineers to solve these issues.

DevOps

Another route software engineers tend to take is DevOps. These engineers are responsible for introducing processes, tools, and methodologies to balance needs throughout

the software development life cycle, from coding and deployment to maintenance and updates [38].

The role of a DevOps engineer becomes relevant after the code has been developed and needs to be released. Reducing the complexity and closing the gap between actions needed to change and application quickly and the tasks that maintain its reliability is expected from these professionals [38]. Safeguarding pipelines, allowing for checks and testing to prevent vulnerabilities in the delivery of software, is also fundamental [38].

Some essential skills for DevOps engineers include continuous integration/continuous delivery (CI/CD) expertise, which allows teams to build and test changes, add repositories, and deploy updates quickly and efficiently [38].

Data Engineer

Data engineers are responsible for designing and building systems for collecting, sorting, and analyzing data at scale [11]. In this data-driven era, organizations have the ability to collect massive amounts of data, and they need the correct talent to ensure the usability of this data [11].

Some typical responsibilities for data engineers include acquiring data sets that align with business needs, developing algorithms to transform data into valuable and actionable items, building, testing, and maintaining database pipeline architectures, and ensuring compliance with data governance and security policies [11].

Quality Assurance and Testing Engineer

The job of a quality assurance and testing engineer is self-explanatory in its title. Their responsibilities include performing tests on software applications to ensure they work correctly [23]. This step is essential in the software development process's planning and design stages, often called verification.

Testing ensures organizations can release usable and high-quality software by identifying bugs or issues related to the application's performance or the user interface [23]. A lot of effort in modern development goes into testing, and even carefully crafted

test cases and mathematical proofs can be faulty. Perfect program verification is unattainable, as it can only point to the fact that a program meets specifications, not that it is free of bugs or issues [8].

2.3.3 Common Skills Required

The differences between sub-disciplines in software engineering and required skills have been outlined. However, common skills are required among all of these professionals, and these are sought after by employers when considering hiring candidates, regardless of the niche in which they will engage with programming.

SWEBOK specifies fifteen key knowledge areas in software engineering: requirements, design, construction, testing, maintenance, configuration management, engineering management, engineering process, engineering models and methods, quality, professional practice, engineering economics, computing foundations, mathematical foundations, and engineering foundations [39].

Some of the most sought-after requirements from employers when hiring software engineers include a passion for continuous self-learning, impeccable communication skills, and proficiency with software engineering concepts. A majority of employers also request a bachelor's degree. However, as mentioned before, this continues to change, and many free and paid resources are available to the public to master software engineering basics [31].

The software engineering industry reacts to new platforms and trends requiring new skills. Students and professionals must remain vigilant and constantly learn new concepts and tools. Self-regulated learning skills allow people to set goals and pursue resources to reach these goals [32].

It is crucial for developers to have excellent communication skills that allow them to work efficiently on teams and orient each other. Still, this area is considered to be underdeveloped [32, 9]. As previously mentioned, Oguz argues that human factors continue to be prevalent in the software engineering industry, and this is evident by the constant need to translate requirements into actionable items [32]. Although technical skills are fundamental for a successful software engineer, decisive human factors

determine if a developer can work in a collaborative environment, making software engineering rely on technical competence and soft skills [9]. Programming is both a social and individual activity, and even though driven by logic and based on a mathematical foundation, collaboration between developers, managers, and stakeholders is fundamental to achieving impact and propelling innovation in the software field.

Lastly, candidates are expected to master software engineering concepts, regardless of the niche and industry they develop themselves in [32]. The SWEBOK, discussed in Section 2.3.1, becomes relevant again, where widely accepted knowledge is discussed and differentiated from educational curricula [39]. It is essential to highlight that software engineers are not expected to know everything related to these subcategories. Still, it is a good list to reference for those starting a career in software development.

2.4 Demand for Software Engineering

Software engineers have become among the most sought-after professionals worldwide [24]. Understanding the market and how it changes allows for a better comprehension of why software engineering is getting more expensive as the years pass [24].

2.4.1 Market Valuation

Software engineering has become one of the most popular and lucrative markets. The global software market, including productivity, enterprise, system infrastructure, and application development software, reached $565B in 2021 [24]. Enterprise software is the biggest category in the market with a projected volume of $271B [24]. Figure 2.2 illustrates the market valuation, highlighting all categories.

Digital transformation and cloud-based software have become significant drivers of investment. There is high demand for software that focuses on progress automation and data analytics to increase efficiency and gain more business insights, respectively [40].

Software engineering was also affected by the COVID-19 pandemic, although not

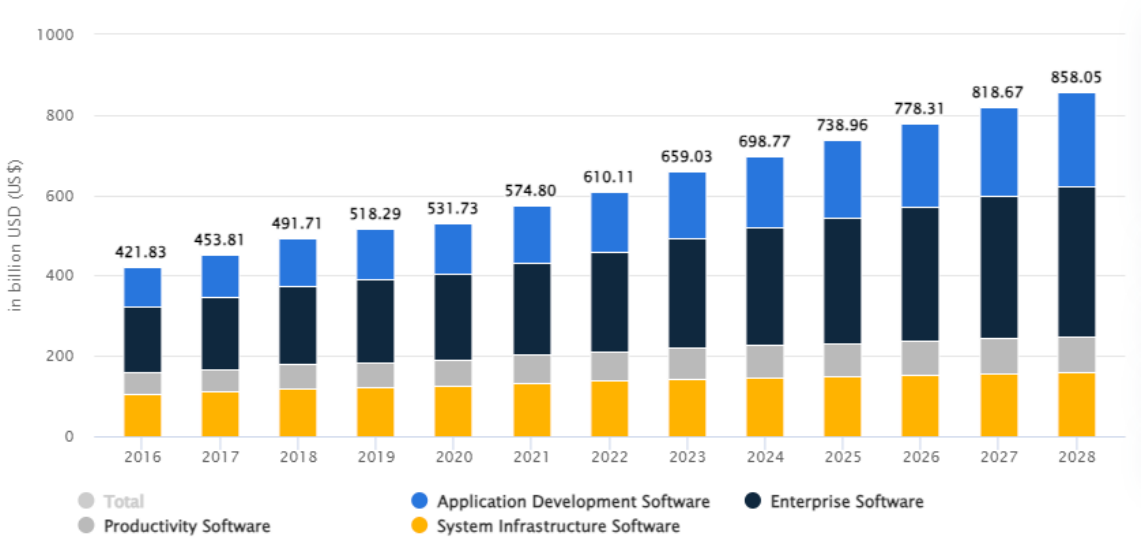


Figure 2.2: Revenue of the software market [40].

in the same proportion as other industries. The adverse effects of the pandemic were relatively short-lived, as the industry continued to grow [24]. The increase of people working remotely during this time led to a massive rise of collaboration in software [40].

Statista also shows that the U.S., China, United Kingdom, Germany, and Japan have the most prominent software markets, in that order [40]. The U.S. reported $338.20 billion in revenue in 2023 alone, by far the most significant market among the mentioned countries [40].

2.4.2 Employment Reports

It could be expected that with the popularity of software, the demand for software engineers will remain proportional, and therefore, the employment rates in this field will increase as well. The U.S. Bureau of Labor Statistics releases employment reports every ten years, and they estimate the job outlook for software developers, quality assurance analysts, and testers is expected to grow 25% by 2032 [31]. Still, this forecast could be inaccurate, given previous forecasts from the same organization being off by almost 50%. The agency forecasted that 752.9 thousand people would be employed in the field by 2022, but the actual employment reflected more than

1.5 million people in the software development specialty [31]. The forecast, off by a whopping 53%, could reflect that the estimates being suggested presently might not hold by 2032. Refer to Figure2.3.

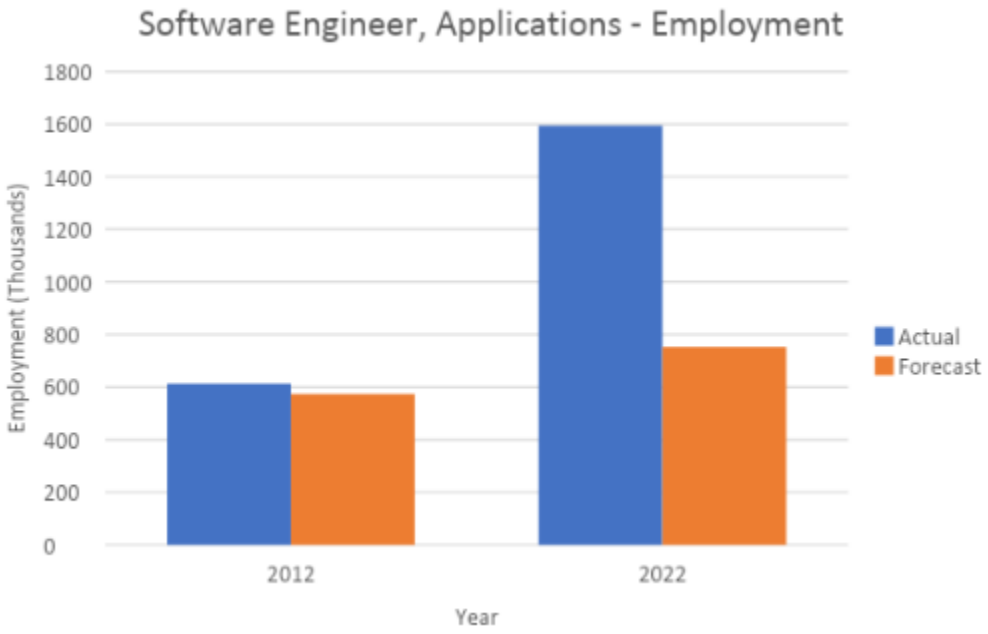


Figure 2.3: Employment forecast for software engineering, applications [31, 30].

Projections methods are heavily based on historical relationships in the data. Still, research is also conducted on factors that are expected to impact employment, particularly those that may not be reflected in historical data, such as new technologies and legislation [30]. Projections are always uncertain, and the exact impact of developments such as new technologies on the labor market ten years out is impossible to predict with precision. Regardless, a series of assumptions are made, in combination with considering the size and demographic composition of the labor force, aggregate economic growth, commodity final demand, input-output, industry output and employment, and occupational employment and openings [30]. Some factors that could be responsible for the inaccuracy of forecasts are mentioned in Section 2.2.2 and 2.2.3 are essential factors such as the availability of the Internet, popularization of smartphones and personal computers and the popularization of data in decision making. Innovation has also been responsible, including machine learning and AI.

2.5 Compensation and Pay

Compensation for software developers varies greatly depending on the job market and location. The average salary for a software engineer in the U.S. was $155,000 in 2021. Still, as mentioned in Section 2.3.2, compensation also varies on the job title, description position, and skills of the individual [24, 33]. Figure 2.4 illustrates the average salaries of professionals with respect to their sub-field in software engineering.

The U.S. Bureau of Labor Statistics also reported the median salary for software developers, quality assurance analysts, and testers to be $124,200 per year in 2022 [31]. This figure is consistent with the surveys cited before. It is vital to consider that compensation also varies significantly by state, where salaries are adjusted concerning the cost of living.

2.6 Interviewing for a Software Engineering Position

Having discussed the origin of software engineering as a discipline, the educational background of practicing professionals and potential career paths, the most popular skills required to become a successful developer, the demand for the profession and the lucrative compensation of employees, the interviewing process in software engineering can be contextualized and understood.

One or more stages are usually part of the interview life cycle. Usually, the process begins by screening the candidate, and depending on the performance, they may be invited to an on-site visit or a more in-depth screening [3]. This process can vary depending on the organization, but known tech companies such as Facebook, Google, and Microsoft are known to follow this strategy [3]. If all goes well, candidates can expect to receive an offer. During all the stages of the interview process, access to external resources is usually discouraged, so extensive preparation is expected from the candidates. Given the incredible popularity of the software engineering field, the process is highly competitive. Candidates must differentiate from the crowd by showcasing impeccable technical and communication skills.

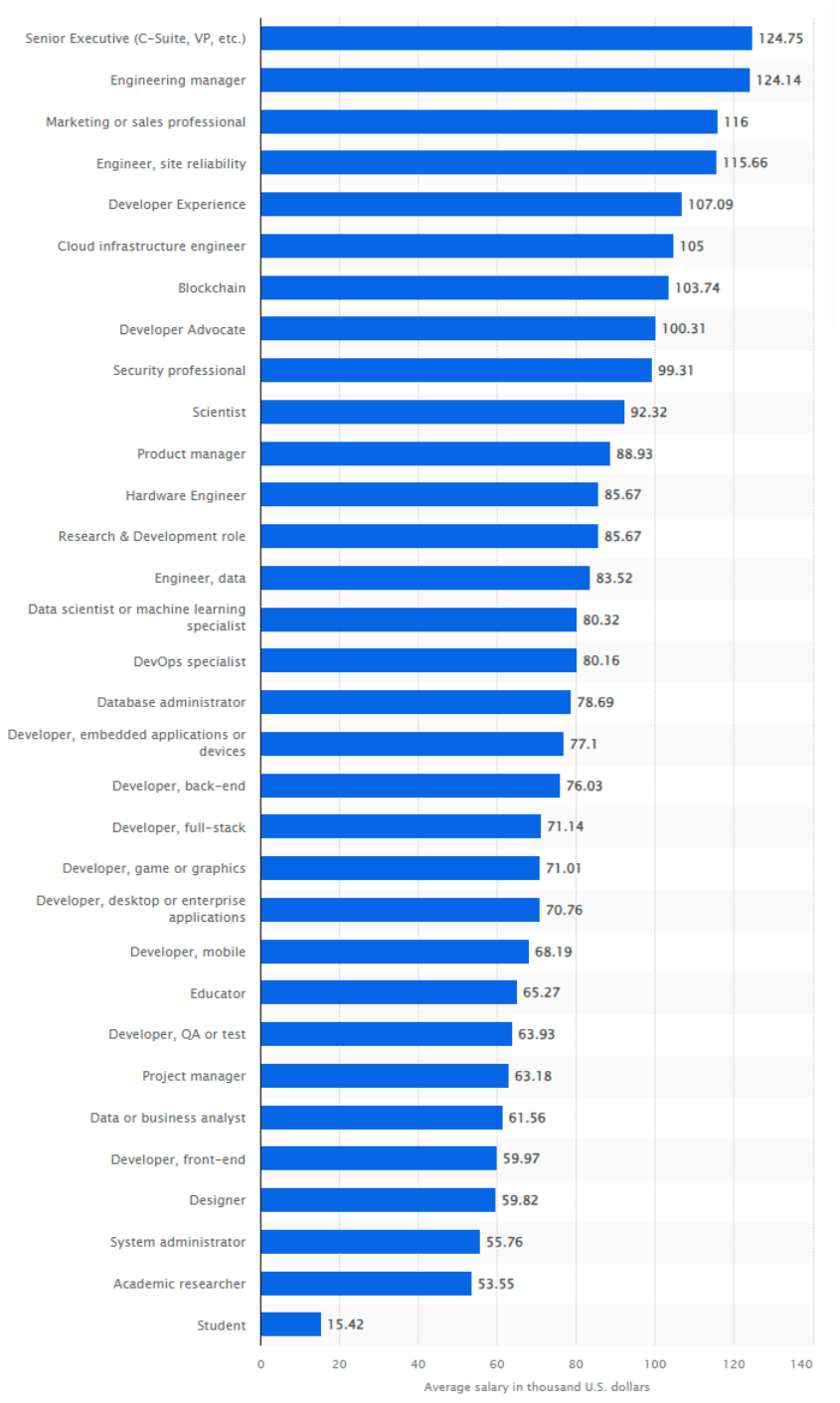


Figure 2.4: Average salaries of software developers worldwide, as of 2023, by role (in 1,000 U.S. dollars) [34].

2.6.1 Technical Assessments

Technical assessments are a common requirement in the interviewing process to evaluate a candidate's technical skills. For software developers, this stage includes a

programming component that tends to focus on algorithms, and it can be administered using different methods [2, 1]. Technical assessments are necessary, but it has been widely debated that they might affect a candidate's cognitive load and not collect representative data on their qualifications. The reason is that a rise of cognitive load might affect problem-solving, an essential element interviewers are looking to evaluate [2]. These interviews might prevent candidates from communicating their thought process effectively, affecting their chances of being offered a position at the organization [1]. Ideally, more important than the solution is how candidates work at getting to the solution [1]. However, candidates must know that knowing how to code is not the only necessary ability in these interviews [21].

The technical assessment type can vary depending on the resources the organization hiring has, the position they are recruiting for, the number of applicants, and other relevant factors. Depending on the context, companies might subject candidates to different tests to assess their technical and interpersonal skills.

Synchronous Evaluation

Synchronous evaluations are when an interviewer is present and evaluates a candidate as they engage in problem-solving. Standard methods include live coding, where the candidate must write code to solve a problem on a digital platform or a whiteboard. These interviews aim to obtain visibility and verifiability into a candidate's cognitive process [2]. In addition, synchronous methods are considered more personable: the interviewer can get to know candidates better, given the more intimate and personal setting provided by these assessments.

Talking aloud is greatly encouraged in this setting, as interviewers are interested in the problem framing. However, how a candidate approaches this requirement will significantly depend on their personality, which can introduce bias to the hiring process [44]. More outgoing candidates might seem better suited for the position, while introverted candidates might be disadvantaged, even if the technical communication skills are equivalent in both cases.

Problem-solving on a whiteboard is a popular technical interview technique used

in the industry [2]. However, one of the effects of COVID-19 was the substitution of the whiteboard with a digital platform that allows the candidate to type code and the reviewer to see all changes made to the code in real time. This allows carrying interviews remotely. Regardless, a study evaluated participants doing whiteboard coding and evaluating their cognitive load [2]. It was discovered that this setting pressures candidates into keeping shorter attention spans, and higher levels of cognitive load are experienced, compared to solving the same problems on paper [2]. Using an eye-tracking system, Behroozi and their team evaluated the cognitive load taken by candidates solving these problems [2].

Other environmental factors that affect the rise of cognitive load include the time pressure and self-efficiency imposed by both interviewer and interviewee, the absence of syntax highlighting, and the talking aloud component [2]. Critics have raised concerns about these types of interviews, as they cause unnecessary stress and reinforce bias in hiring practices, since least visibly stressed candidates have higher chances of securing the job [2].

Asynchronous Evaluation

Asynchronous methods are also available for companies to assess candidates. The candidate is also expected to solve a problem without access to resources during asynchronous evaluation methods. Still, in contrast with synchronous evaluation methods, they do not include the presence of an evaluator in real-time. Instead, candidates are responsible for submitting code that will later be reviewed, making this technique suitable for assessing large pools of applicants. Two general types can be found: take-home assessments and online timed assessments.

Take-home assessments resemble a small class project, where candidates are provided a prompt and are expected to craft code. Take-home assessments are lenient with the time limit aspect: The candidate has a generous window of time, which can go from a few hours to several days, to solve the problem and return to the evaluator.

In contrast, online assessments tend to be more time-pressured: candidates are provided a prompt and must use software engineering fundamentals, algorithms, and

data structures to solve the problem. Employers provide test cases showing whether the problem has been solved correctly and if all corner cases have been considered.

Asynchronous evaluation has advantages and disadvantages. Advantages include less pressure from having an interviewer present and less cognitive load for the resolution of the problem [3]. The advantage also becomes the disadvantage, as candidates do not have access to a person to clarify questions and to show their thinking process, making asynchronous testing evaluation very strict. Cheating is a potential complication for these types of assessments, inhibiting employers from accurately evaluating candidates [3].

2.6.2 Hiring Costs

Companies spend immense resources searching for talent and bringing engineers on board to existing teams [13, 5]. Recruiting sourcing costs add quickly, and even though salaries make up for most of the engineering budgets, sourcing costs are an afterthought and sometimes not considered from planning stages [13]. The hundreds of hours spent evaluating and assessing candidates, which includes reviewing resumes, phone screens, technical interviews, and on-site visits, incur significant costs, and not all candidates make it to hiring [13].

Most of the candidates could not be qualified, to begin with, and under-screening candidates can also become a problem because they can move on to later stages of the process, requiring more engineering time [13]. Engineering teams are primarily responsible for hiring since they must assess and evaluate all candidates, review code samples, run interviews, and do phone screens [13].

In smaller teams, the responsibilities for carrying out interviews rely on the vice presidents or even the C-suite, as with minimal infrastructure, they are forced to lead the process, also creating a significant distraction for other essential work duties [13].

The right balance within the hiring process is necessary to bring down costs. With some simple math, including hours necessary for each hiring task, the number of candidates assessed per stage, and the average hourly salary of teams responsible for reviewing and assessing candidates, an estimate on the cost of hiring software engi-

neers can be deducted [13]. Assuming a developer's productivity cost to a company is roughly $350 per hour, it is estimated that the cost of profiling and assessing a pool of candidates is $22,750 [13]. This does not account for the energy expense necessary to provide a good interview experience for candidates. Unnecessary interviews, therefore, create a significant toll on productivity [13].

Filtering candidates appropriately is necessary to minimize the number of interviews that assess coding competencies, allowing engineers to focus on their work and invest time in profiling candidates only when granted and necessary [13]. Improving the hiring process pipeline would increase the efficiency of the process, ensuring viable talent is allowed to make it to the next stage of the process [13].

2.6.3 Preparation

Given the extensive interviewing process software engineers need to endure, ample preparation is strongly encouraged. Job positions with multiple interviewing rounds might take longer, creating a perception of time loss if the candidate is not offered the position after multiple rounds.

Another essential factor that encourages ample interview preparation is the competitive market for software engineering positions, given the increase in market value and software demand, discussed in Section 2.4. Even though there are great offers and great demand, companies are always looking for the best possible talent, making the interview process even more competitive.

An Emerging Market

Consequently, online communities and problem practice platforms have emerged, allowing people to access tools to practice common algorithm problems others have found before and to share their experiences when interviewing for software engineering roles. These communities have also grown to host discussions around the best interview preparation methods and job offer negotiation.

Popular sites in the software engineering community include LeetCode, an online

platform that allows users to improve their skills and prepare for technical interviews. LeetCode hosts contests, provides assessments and training, and guides users through their technical careers [37]. The other main competitor in the space is HackerRank, which, in addition to offering a practicing platform for developers, also offers a service to companies to identify, test, and hire developers [20]. Companies can outsource the technical assessment component to organizations like HackerRank, allowing them to adapt and scale their hiring processes [20]. Both private companies were founded in 2010 and 2009, respectively, and have popularized over time as the best alternatives for software engineering interview preparation [37, 20]. A saturated industry has generated an entire market, forcing companies to scale their processes and developers to prepare better using resources such as LeetCode and HackerRank.

Another aspect of the preparation market includes books, courses, and the boot camp programs mentioned in Section 2.3.1. Cult favorites include "Cracking the Coding Interview" by Gayle Laakmann McDowell, where popular companies' different interviewing styles are outlined, and common types of questions are outlined: object-oriented design, recursion, dynamic programming, and system design [25]. Other algorithm books are also famous for brushing up on basic concepts that are usually evaluated in technical interviews. Still, practice problems provided by books such as Laakmann McDowell's and practicing platforms allow putting fundamental computer science concepts into more tangible practice.

A final resource available to candidates is mock interview hosting sites, where candidates can pay real interviewers to host a mock interview with them for a given amount of money. This can allow candidates to do trial runs of the interview, understand what to expect, and correct their mistakes before the interview occurs.

2.6.4 Pitfalls and Challenges

Communication skills will be assessed to a different degree depending on the type of interview. What all the interview styles outlined in Section 2.6.1 have in common is the evaluation of the candidate's technical proficiency. Combining these methods will ensure a candidate has the programming skills that the company is after. Even

though the success metrics in these interviews seem objective and precise, bias might be introduced in the process, and there might be a mismatch between what candidates think interviewers are assessing versus what criteria are used in practice [15].

Interview anxiety affects everyone, and research suggests it can harm interview performance, benefiting candidates that handle and migrate stress better [21, 4]. Underrepresented minorities are often affected the most, and it is crucial to consider this when designing an unbiased interviewing process [21]. Given that the interview preparation process is ample, those with better access to resources have an advantage over other candidates who might not have the time or financial means to pay for preparation courses, books, or mock interviews. The interview process can be considered analogous to standardized testing preparation, which has proven to be biased towards family income and wealth instead of actual knowledge, experience, and preparation [12].

Interviewers and candidates not being on the same page might be another issue of importance. Evaluators might make different interpretations in essential categories, such as problem framing and algorithm mastery, such as favoring specific solutions rather than the candidate's problem-solving approach, even though the latter is more important. Depending on the interviewer hosting the conversation, the interview process can be more biased and benefit a particular pool of candidates.

Literature cannot explain what individual factors are related to a successful solver [44]. A study by Wyrich et al. suggests that happiness, experience, academic performance, and personality might be linked to performance in assessments that require coding [44]. The study was conducted among software engineering students and did not consider the preparation aspect. Often, academic concepts are readily accessible to junior candidates and fresh-out-of-school prospects, favoring them in technical interviews that focus on computer science fundamentals and algorithm development. More senior candidates might need to refresh these concepts before applying to positions, requiring extra preparation time. It can be observed that there is an overall trend that favors preparation over experience in the software engineering hiring pipeline.

Other engineering disciplines have their own way of screening candidates and assessing their preparedness for a role. Interviews tend to be more verbal, avoiding showcasing fundamental skills needed for the role. More emphasis is given to past projects, experiences, and knowledge of theoretical fundamentals to assess these candidates. Software engineering is unique, where it is often the exception to bypass technical assessments due to a candidate's previous experience at another company. For this reason, interviews are highly favored to allow exposure to the field and to get a foot at the door in certain companies whose hiring policies might be more flexible for interns. Therefore, getting a job as a software engineer requires experience, time, and academic and financial resources to bypass a rigorous screening process that might or might not be consistent with the day-to-day responsibilities of the job.

Chapter 3

Research Methodology and Data Collection

MIT's Committee on the Use of Humans as Experimental Subjects (COUHES) deemed the research pertinent to this thesis as exempt from further regulation since the study activities were limited to adults only and the collection of information could not damage the subject's financial standing, employability, educational advancement, or reputation. Considered a benign behavioral intervention, the study was given clearance only after completing a course related to social and behavioral research investigations. Some of the relevant modules for this course included informed consent, privacy, and confidentiality. The surveyor and interviewer had to complete mandatory elements of this course, which aided in designing a survey and a semi-structured interview.

This thesis outlines both quantitative and qualitative data collection and analysis. Data related to surveyees' experience with timed coding assessments, called online timed assessments in Section 2.6.1, was collected through a short survey. The goal was to obtain a comprehensive pool of candidates with different educational backgrounds and experiences to minimize the research bias. The surveyees could provide contact information at the end of the survey if they wished to speak about their experience in more detail during a semi-structured interview.

3.1 Survey

A survey was designed to collect relevant information about professionals in software engineering or other similar fields. Recording demographic information and their experience interviewing for jobs was essential, especially with online timed assessments. Those with experience interviewing and hiring software engineers had to complete an additional portion of the survey. The goal was to collect both the candidates' and managers' perspectives on essential matters, such as the efficiency and efficacy of timed coding assessments. Thirty-nine participants answered the complete survey, allowing the synthesis of results.

3.1.1 Survey Content

Since the software engineering field is constantly evolving and there is no standard method of interviewing candidates, it was essential to define a timed coding assessment before the start of the survey. For the purposes of this research, a timed coding assessment was defined as "an evaluation in which the candidate has a specific time frame to complete simple programming tasks and/or complex algorithm problems." More clarification was added, mentioning "a problem statement outlines the task that must be completed, and the candidate is expected to write code that solves the given problem with a given time constraint and evaluation criteria." With this information, surveyees could pinpoint relevant experiences before starting the survey, minimizing the collection of inaccurate and/or biased information.

Non-exhaustive demographics were collected to contextualize the answers provided on the survey. Primarily, the survey considered educational background, years of experience, most recent job title, industry, and company size to investigate the answers to the remaining questions. This portion of the survey also worked as a grounding truth, allowing the comparison of answers to previous research done in the field, providing legitimacy to the survey, and ensuring that a diverse sample pool was evaluated.

Surveyees were sequentially asked about the frequency of their engagement with

timed coding assessments, methods they used to prepare for the interview process, their success in clearing the timed coding assessment stage and moving forward in the job interview process, and overall feelings about timed coding assessments. Non-binary options were provided, allowing respondents to rate their opinions more granularly.

Two critical questions in the survey were related to the efficiency and efficacy of timed coding assessments. The surveyees were asked to rate the efficiency of these assessments from a candidate perspective, considering if the technique allowed to minimize the time to profile a candidate's technical qualifications. Similarly, they were asked to rate timed coding assessments considering their efficacy, meaning if they are a valid method to profile a candidate's technical qualifications.

As a last important data point, participants had to score from 0 to 5 on the relevancy of the following skills in timed coding assessments, meaning if these skills are actually considered and judged during that process and to what extent:

- Language proficiency and syntax.
- Problem framing
- Computer science fundamentals
- Troubleshooting
- Version control
- Runtime and algorithm performance
- Time management
- Code organization
- Resource integration (libraries, frameworks, environments...)
- Technical communication
- Design and code architecture

In the last stage of the survey, those with management and hiring experience could show their opinions from a different perspective, understanding that candidates and managers might have different needs and biases towards the process. The same efficiency and efficacy questions were asked, considering their experience as hiring managers. In addition, they were also asked to score from 0 to 5 the relevancy of technical skills, previously outlined, for successful software engineering candidates. These mirrored questions aimed to assess the correlation between what timed coding assessments measure and what participants with hiring experience believe are fundamental skills for a successful software engineer. The entire survey is outlined in Appendix A.

The survey was designed to avoid collecting any personal or identifiable information. The only question that required identification was the last one, where surveyees were asked about their interest in discussing their experiences in more depth. If consenting, their name, last name, and contact information were collected.

3.1.2 Dissemination

The survey was disseminated through various channels, including LinkedIn, instant messaging platforms, academic communities, and word of mouth. The survey participants were encouraged to share the link to the survey with colleagues and professional contacts with relevant software engineering experience. When the survey link was shared, it was also disclosed that the information collected would only be used for the purposes of this thesis. The links were disseminated multiple times to maximize the survey's reach to as many participants as possible. The survey was kept active for six months.

3.1.3 Survey Data Analysis

Since the survey was hosted using Qualtrics, a comprehensive report on the answers was generated automatically, showing the count, mean, standard deviation, variance, and percentage distribution of answers. The data was imported into Excel to analyze

and interpret. Identifying patterns, trends, and relationships was fundamental during this process, considering the different variables acquired through the survey.

3.2 Semi-Structured Interviews

Surveyees interested in sharing more about their experience on an interview were able to specify contact information at the end of the survey. Those interested were approached via email to schedule a virtual meeting. A total of 12 participants agreed to be interviewed.

Interviews lasted one hour and were semi-structured, allowing the conversation to have a defined layout but leaving room for additional questions at the interviewer's discretion. However, it was essential to minimize bias in the interviewing process. Therefore, the interviewer maintained a neutral tone, not sharing their personal opinion about the subject through explicit opinions or analogies. It was fundamental to collect data organically, allowing interviewees to draw their own thoughts, conclusions, and analogies backed up by their experience in the field.

3.2.1 Consent and Disclosure

Despite the research for this thesis being exempt from further oversight, obtaining verbal consent from all participants before proceeding with the semi-structured interview was necessary. The consent involved a detailed explanation about the intentions of the interview and disclosing that the information would be used for the purpose of the research for this thesis only. Furthermore, the participants were guaranteed that all sensitive information would be de-identified when sharing results. Unlike the survey portion, identifiable information was collected during this stage, including companies the interviewees worked and interviewed for, their process, their experience with the interviewing process, and the work carried out at those companies.

Once the participants consented to have their information collected, they were asked for additional consent that would allow for the video and audio recording of the interview. The purpose of recording was to facilitate data collection and analysis.

3.2.2 Semi-Structured Interview Content

The central theme of the semi-structured interviews was the candidate's experiences with timed coding assessments. However, during the interview, it was necessary to put the candidates at ease and allow them to introduce themselves after the consent and disclosure were obtained and collected. The participants could provide more context to their demographics. Another icebreaker question was understanding why they got interested in the software engineering field before proceeding with questions about coding challenges. The interviewees were provided a brief description of what a timed coding assessment meant in the context of the interview, similar to the definition described in the online survey.

Experience with timed coding assessments was collected, regardless of how long ago participants experienced them, paying particular attention to the outcome they had during the assessment, either positive or negative, and their thoughts around the efficiency and efficacy of the examination. Similarly, ample questions about the preparation were asked to understand how much the interviewees studied for their interviews and which resources were used and considered best. Participants were also asked about their general experience with the interviewing pipeline and overall thoughts and opinions on timed coding assessments.

The next phase of the interview focused on correlating their job and the assessment used to qualify their technical skills during the interview process. Since not all candidates got their jobs via a timed coding assessment, other types of examinations were also discussed in this section. The main focus was to understand the correlation between the examination and the day-to-day responsibilities at the position once they were extended an offer letter. Candidates were encouraged to talk about their general work history and rehash past experiences, not only on their current or most recent job.

The interviews also sought to obtain candidates' and managers' perspectives on using timed coding challenges and general opinions on other types of technical assessment used in the software engineering industry. For those participants without hiring

experience, a hypothetical scenario was drawn where they were asked to describe the qualifications of a successful software engineer, in their opinion, and what evaluation methods would be helpful to assess those abilities.

Finally, the candidates were probed on what they think hiring in the software engineering industry will look like, considering recent technology developments and the overall climate of the job market. This was also an excellent opportunity to ask for additional information sources relevant to the research. The entire semi-structured questionnaire can be found in Appendix **??**.

3.2.3 Process

Interviews were carried out virtually, and participants had one hour to explain their in-depth experiences during the semi-structured interview. Once the disclosure of information was shared and consent was provided, the interviewer followed the script to kick-start the conversation. This thesis aimed to focus primarily on timed coding assessments, but upon interviewing several participants, it became apparent that other interviewing methods are widely popular. As a consequence, not only experience with timed coding assessments was collected, but also with the other methods, their efficiency and efficacy, and their correlation to day-to-day responsibilities on the job. As a result, the collected data for interviews can be clearly segmented between timed coding assessments and other interviewing methods.

Minimizing bias and engaging in active conversation with participants was fundamental during the interviews. Constant questioning about their answers took place to understand the meaning behind opinions entirely. The participants were requested to back up their answers with either experience or reasoning. Transcripts were acquired and then meticulously processed shortly after the interviews to allow for reconstructing the information, given the recency of the conversation.

3.2.4 Interview Data Analysis

For the interviews, considering the qualitative nature of the data collected, a mixture of content, thematic, and narrative analysis was used to interpret results and draw conclusions. Content analysis focused on the presence of words, subjects, and concepts in interviews. The thematic analysis focused on interpreting patterns and themes in the data. Finally, narrative analysis centered on parsing the participant's stories and considering their feelings and behaviors.

The transcripts generated by the audio recordings collected during the interview were cleaned and organized to allow better readability. It was essential to correct any errors that stemmed from transcription, and with audio recording available, it was possible to avoid information loss. Once the transcripts were cleaned, they were annotated, considering common themes with respect to other previously analyzed transcripts and unique experiences given the interviewees' exposure to specific situations, contextualizing their work and cultural environments.

Using a spreadsheet, a column with common themes was populated. The research questions outlined in Section 1.2 were kept present at this stage and allowed having a basis of themes. As each transcript was analyzed, more themes were added to the column. Consequently, each transcript was carefully read twice, allowing the registration of themes detected after the initial transcript inspection.

Additional columns were added for each participant, characterized as Px, where x is a number between 1 and 12, allowing the cross-correlation of themes to a participant in matrix form. The categories and themes yielded by all the analyses were then categorized into more significant and broader themes, allowing the association of different themes under similar topics. Finally, the segments were analyzed by counting how many participants mentioned a particular theme or had similar opinions. This allowed aligning with existing quantitative data from the previously carried survey and other previous work outlined in Chapter2.

Another fundamental aspect of the qualitative analysis was to analyze the interviewers' tone, their general feelings toward timed coding assessments, and the

currently popular trends for interviewing software engineers. Sentiment analysis was conducted with semi-structured interview scripts to understand whether the participants' tone was positive, neutral, or negative. The level of confidence in the routine was also recorded, as sentiment analysis tools have biases and limitations. For example, challenges such as subjectivity and tone, context and polarity, irony and sarcasm, and comparisons can alter results, as models are not equipped to deal with these scenarios well.

Different sentiment analysis tools have applications in which they perform best. A natural language processing (NLP) tool was used for this thesis, but it was marketed towards businesses looking to monitor brand reputation and customer feedback. However, this sentiment analysis model provided results consistent with subjective qualification of tone from the interviewer, allowing the minimization of bias.

Chapter 4

Results

The information collected from the survey and semi-structured interview was compiled and studied, yielding thought-provoking patterns that aim to answer the research questions outlined in Section 1.2. The sections below report the results for both data collection methods.

4.1 Survey

Since the survey was catered towards professionals in the software engineering industry, a fundamental question to pose was the respondent's current or latest job role. Most of the 39 volunteers who filled out the survey reported holding a software engineering position, some with varying seniority levels. Other positions included Chief Technology Officer (CTO), research engineer, embedded hardware engineer, technical product manager, and project manager. The distribution is outlined in Figure 4.1.

This information shows that a handful of technical disciplines involve technical examinations, including software engineering, as part of the interviewing process. This also shows a natural progression from software engineering to other positions, allowing professionals a career trajectory beyond developing code. It is fundamental to clarify that participants with some software engineering experience were encouraged to complete the survey, regardless of their current job status or role. For this reason, disciplines that usually do not involve administering a coding assessment are reported

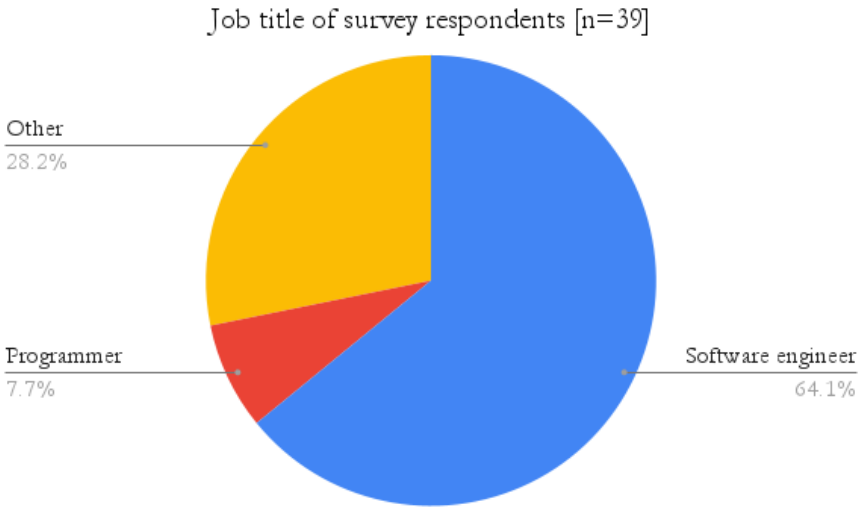


Figure 4.1: Job title of survey respondents [n=39]

in the results.

4.1.1 Demographics

Education and Experience

The results from the survey are consistent with research previously carried out about the educational profile of software engineers. Most participants from the survey held a bachelor's degree, closely followed by those with a master's. Figure 4.2 shows the distribution of educational degrees across the 39 participants.

In terms of years of experience, there was reasonable distribution across those surveyed. Similar to broader surveys and government reports, most participants had between 2 and 5 years of experience, followed by those with between 5 and 8 years of experience. It was also valuable to have those with minimal experience and those who could be considered experts in the field with more than ten years of experience. This even distribution allows collecting information about timed coding assessments more consistently.

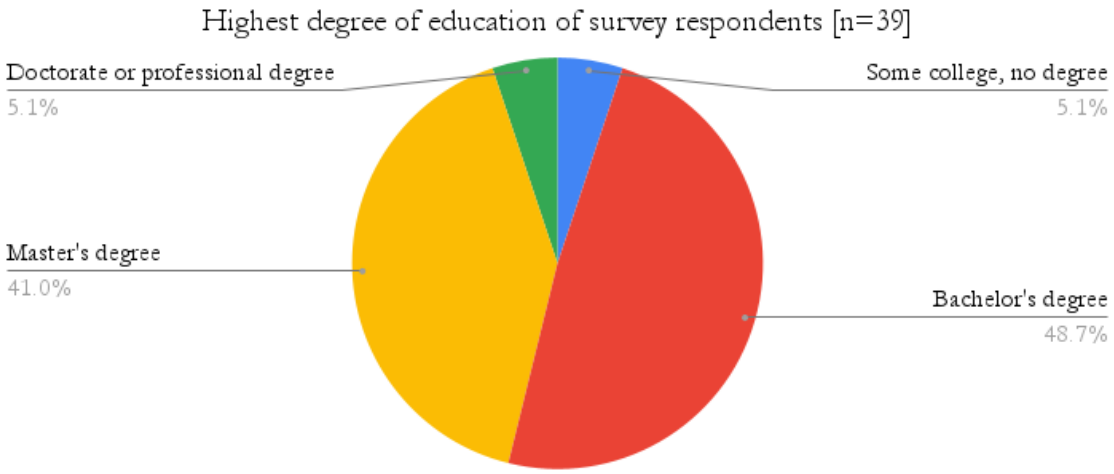


Figure 4.2: Highest degree of education of survey respondents [n=39]

4.1.2 Timed Coding Assessments

Among the questions related to timed coding assessments, candidates had to answer with which frequency they encountered timed coding assessments throughout their entire employment history. This question was asked to determine how much of a common practice timed coding assessments are in the entire software engineering industry. Figure 4.4 shows how 46% of the participants mentioned most of the time they had to take an assessment of this nature to be considered for a position that involved programming. The option "never" was included in the survey, but none of the participants checked this option, showing how standard the practice of technical assessments is.

The specific industry within software is another data point that can be correlated to the frequency of taking assessments. Some industries might be more competitive than others regarding recruiting talent. Figure 4.5 shows the majority of participants belonging to the technology sector. Timed coding assessments have been reported to become widely popular in the FAANG companies (Facebook, Apple, Amazon, Netflix, and Google) as part of their recruitment process. Other technology firms not part of the FAANG acronym could also belong to this segment chosen by participants. Other sectors reported included education, automotive, manufacturing, and

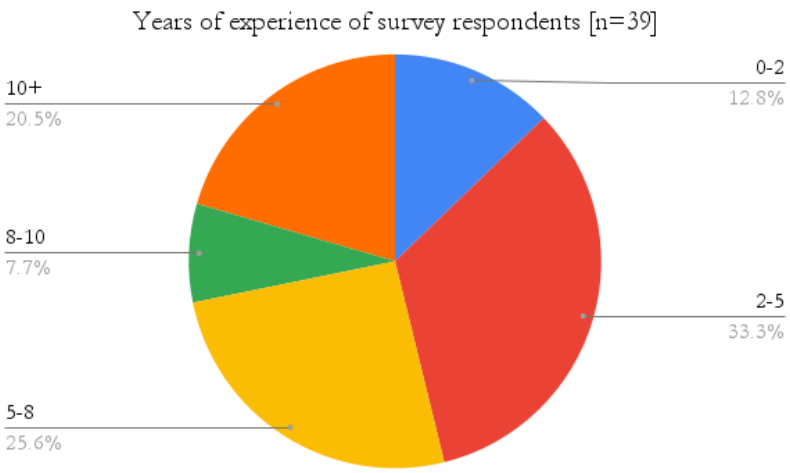


Figure 4.3: Years of experience of survey respondents [n=39]

transportation. The results from this question reflect how asynchronous assessments have been adopted widely, especially in the technology sector.

More than half of the participants answered their companies surpassed the 250 employees, showing that larger companies might benefit from timed coding assessments as part of their interview process to filter candidates in high-demand positions. By filtering out "inadequate" candidates, companies might save money and time in the hiring process, granting interviews to candidates who can pass strict assessments only.

Preparation

Those who completed the survey also reported their preferred preparation methods and time dedicated to studying for timed coding assessments. For methods, more than half of the respondents cited using popular online platforms, such as LeetCode and HackerRank, for practicing problems. Other popular methods included books and more structured education methods, such as courses and boot camps. Participants also reported YouTube videos to be particularly helpful in the preparation process. Participants could list more than one method of preparation used in this question. The distribution of answers is shown in Figure 4.6.

Answers from participants also confirm practicing problems could be considered

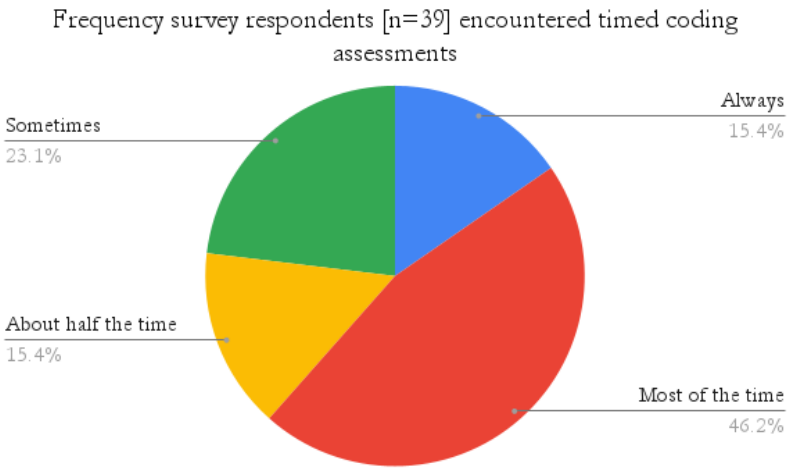


Figure 4.4: Frequency survey respondents [n=39] encountered timed coding assessments

the most effective method of preparation since it is relatively common to find similar questions during interviews. Online platforms also feature community tabs where users can report which questions have been asked by which companies, aiming to help other fellow software engineers, which might be especially helpful to those without interviewing experience in the field.

In terms of the amount of time used to prepare for asynchronous assessments, participants preferred the two extremes: either little preparation or extensive preparation. Figure 4.7 outlines the distribution of answers across participants.

The distribution of answers to these questions might be attributed to the candidate's years of experience. Candidates fresh out of school might require less preparation, as the standard computer science and software engineering curricula feature some of the skills considered necessary in the hiring process. A refresher might be needed for older candidates who have been in the industry for years. Finally, the preparation time can also be attributed to each participant's strengths, weaknesses, and responsibilities outside of interviewing.

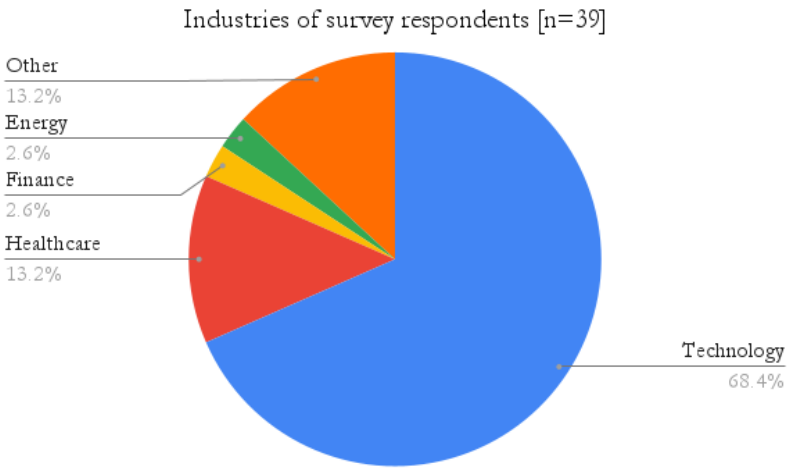


Figure 4.5: Industries of survey respondents [n=39]

Efficiency and Efficacy

During this phase of the research, obtaining both candidates' and managers' perspectives on the efficiency and efficacy of timed coding assessments was fundamental to helping create the semi-structured interview script that would proceed to the survey stage of the research. Recognizing the bias between the manager and candidate positions and the evaluation method and criteria used was critical. For this reason, participants were asked to rate the efficiency and efficacy of timed coding assessments when considering if they were hiring candidates or if they were the candidates interested in the position. Notably, those with managerial experience were a subset of all participants who answered the survey: from the 39 participants, 20 reported having hiring expertise.

When considering efficiency, if companies can get candidates quickly enough, candidates reported timed assessments to be mostly slightly and moderately efficient, showing a negative trend. The opinion of managers was similar: a majority recorded that asynchronous assessments are only slightly and moderately efficient. Refer to Figure 4.8.

When considering efficacy, meaning if companies can get qualified candidates for the position through asynchronous examinations, most candidates reported them to

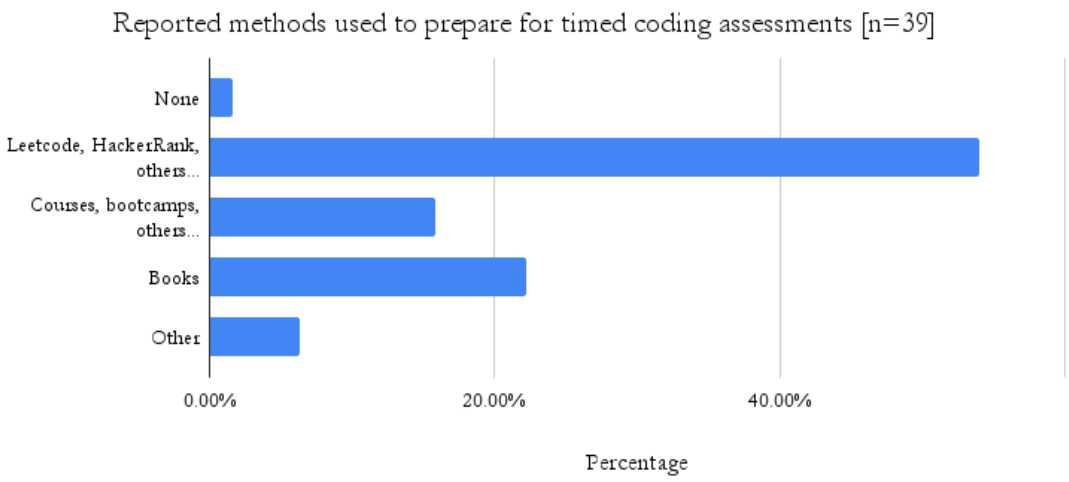


Figure 4.6: Reported methods used to prepare for timed coding assessments [n=39]

be moderately or slightly effective. Managers expressed a similar thought. Most managers consider examinations moderately and slightly effective but never extremely effective.

Managers considered the process to be less efficient overall than their candidate counterparts. Candidates tend to have a slightly more pessimistic outlook on the efficiency and effectiveness of the process, while managers have a more positive vision. Still, the distribution of the results seems to be relatively proportional, showing no significant bias between managers' and candidates' opinions on timed coding assessments to find qualified candidates quickly.

4.1.3 Correlation of Interviews with Job Duties

Surveyees were asked to report the degree of correlation between their job duties and timed coding assessments they had encountered during their career. The distribution of answers was even, showing that around half of the respondents believed there was a correlation. In contrast, the other half was split between those who thought there was no correlation or there might have been some correlation. Figure 4.10 illustrates the distribution of answers for the 39 volunteers.

These results do not offer a concrete answer on whether timed coding assessments

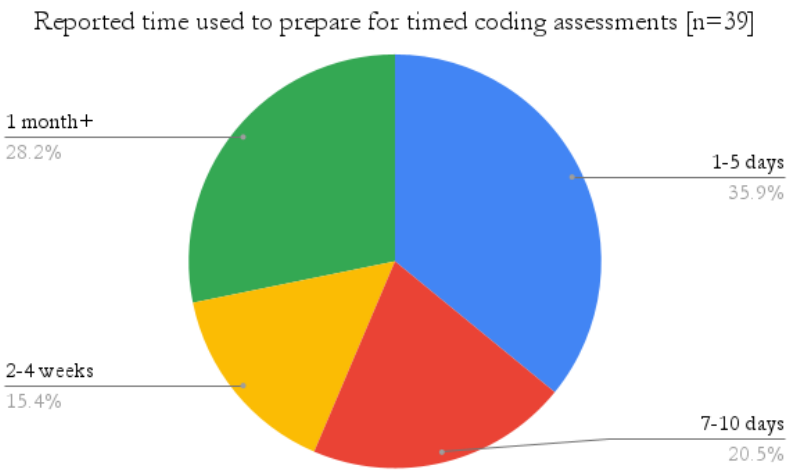


Figure 4.7: Reported time used to prepare for timed coding assessments [n=39]

relate to job duties, meaning different companies might have different styles of administering asynchronous assessments, some having a more robust correlation and providing a better interview experience to candidates.

4.1.4 Metrics of Success

Surveyees were asked to rank a series of skills in order of importance, considering what is highly evaluated in timed coding assessments. Scoring the skills between "not considered" with a score of 0 and "greatly considered" with a score of 5, it was possible to analyze what participants consider timed coding assessments to evaluate. It is essential to clarify that the mean provided by Qualtrics does not reflect a mean truly, as the scale that participants used to rank the skills is not integer and is ordinal. Given the inherent subjectiveness in perception when ranking the skills, a weighted approach or simple count could have been implemented. However, the mean provided by Qualtrics was used as a proxy for relative importance. With this in mind, the rank was collected and is illustrated in Table 4.1

The top 4 answers included computer science theory, problem framing, time management, and language proficiency. Timed coding assessments feature problems that force candidates to think about computer science fundamentals, such as selecting

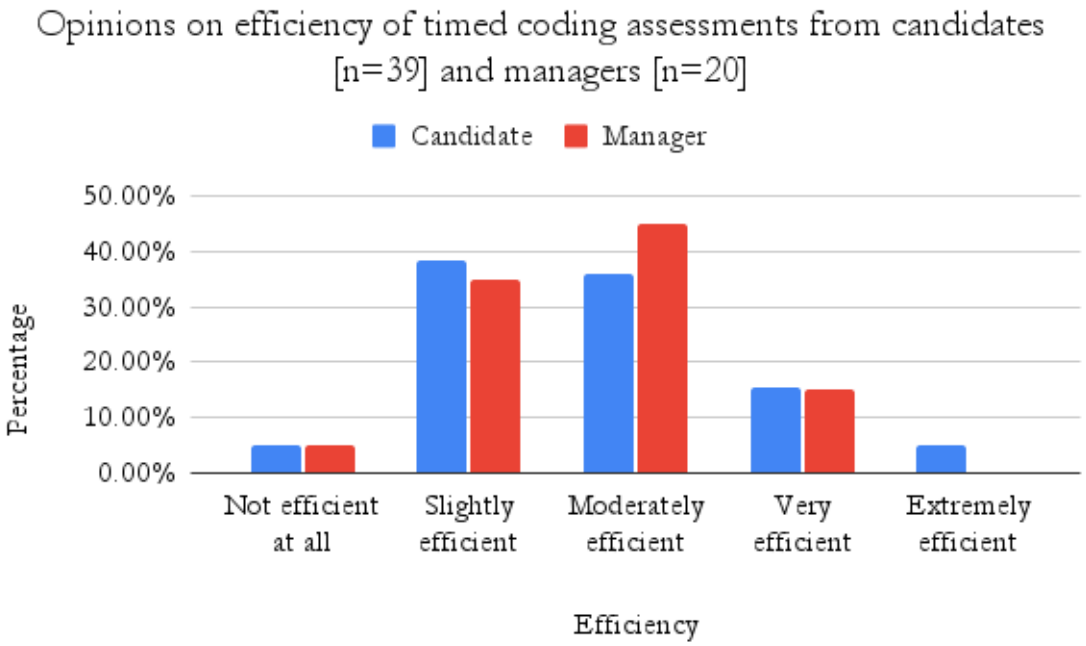


Figure 4.8: Opinions on the efficiency of timed coding assessments from candidates [n=39] and managers [n=20]

Table 4.1: Opinions on skills evaluated in timed coding assessments

Skill	Mean
Computer science theory	3.67
Problem framing	3.49
Time management	3.1
Language proficiency	3.05
Technical communication	2.85
Troubleshooting and debugging	2.46
Runtime and performance	2.46
Code organization	2.46
Design and architecture	2.44
Resource integration	1.62
Version control	0.87

the correct data structure for the problem at hand. Similarly, given the tight time constraint, it is no surprise time management was also scored as a top skill evaluated.

With the hope of correlating evaluated skills with desirable traits, those with hiring experience were asked to do a second ranking, considering the skills necessary

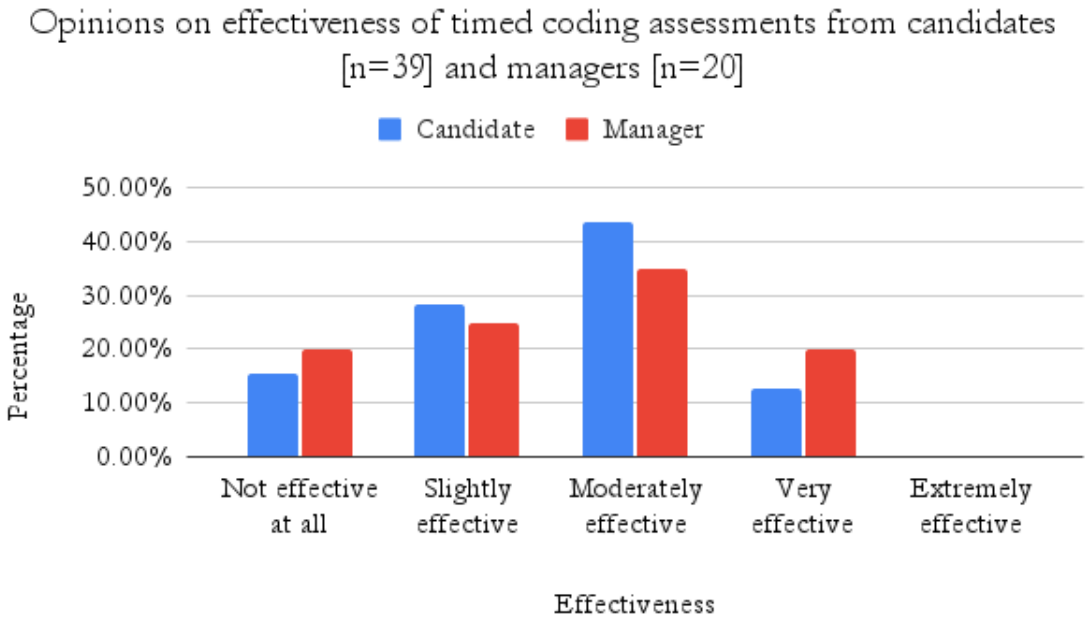


Figure 4.9: Opinions on the effectiveness of timed coding assessments from candidates [n=39] and managers [n=20]

for a successful candidate in the software engineering industry. Scoring the skills between "not necessary" with a score of 0 and "absolutely necessary" with a score of 5, the mean of the answers was calculated. Table 4.2 shows the ranking provided by managers, also sorted in Qualtrics' mean descending order.

Table 4.2: Manager opinions on skills necessary for a successful candidate

Skill	Mean
Technical communication	4.15
Troubleshooting and debugging	4.1
Problem framing	4.05
Code organization	3.95
Design and code architecture	3.65
Time management	3.6
Runtime and performance	3.1
Computer science theory	3
Language proficiency	2.75
Resource integration	2.7
Version control	1.8

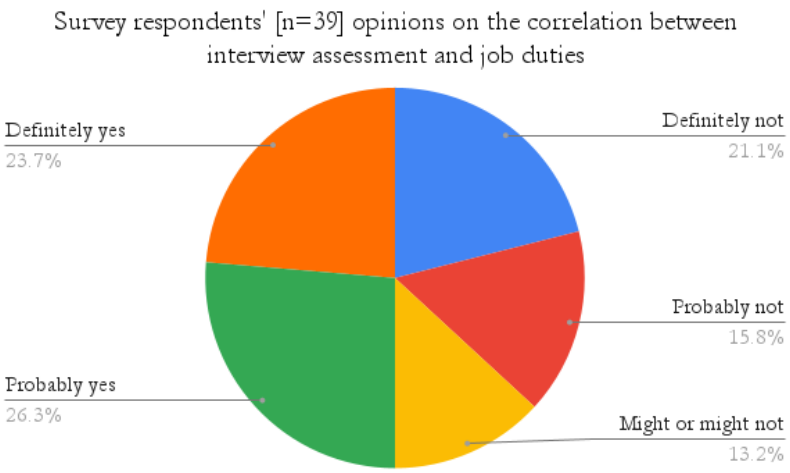


Figure 4.10: Survey respondents' [n=39] opinions on the correlation between interview assessment and job duties

This time, the top 4 skills included technical communication, troubleshooting and debugging, problem framing, and code organization. The top skill ranked by managers as the most desirable is a soft skill, technical communication, which has been reported to be marginalized in computer science curricula, as discussed in Section 2.3.1. Not surprisingly, technical communication could allow teams to function better, making technical proficiency secondary. Another skill highly ranked was troubleshooting and debugging: creating new programs and applications could be less common than maintaining already functional and deployed software. Therefore, managers might appreciate this skill and consider it fundamental for successful candidates.

The only skills ranked in the same positions for both categories were runtime and performance, resource integration, and version control. Another skill ranked similarly in both scenarios was problem framing, scoring second and third on skills assessed and desired skills, respectively. Problem framing allows interviewers to understand the thinking process of candidates, which might differ significantly but could provide insights into their work personality and outlook on challenges.

With the survey answers, semi-structured interviews were carried out to understand participants' experiences when interviewing for software engineering roles in more detail.

4.2 Semi-Structured Interviews

4.2.1 Demographics

Education and Experience

Following the analysis process outlined in Section 3.2.4, it was possible to quantify the frequency of similar answers across candidates. The first category analyzed was general demographics. Limited demographics were collected during the interview portion of this research since these answers were already recorded when participants filled out the online survey. As Section 3.2.4 mentioned, each participant was given a unique Px identifier from 1 to 12. The analysis of these demographics yielded consistency on previous studies described in Section 2.3. Table 4.3 shows an average of 5.9 years of experience in participants, 66% computer science or similar majors, and 41% of professionals holding bachelor's degrees.

Table 4.3: Education and years of experience

Participant	Education Degree	Years of Experience	CS Major or Similar
P1	M.S.	9	X
P2	Ph.D.	10	
P3	B.S.	3	X
P4	M.S.	3	X
P5	B.S.	2	
P6	B.S.	5	X
P7	M.S.	10	X
P8	M.S.	4	
P9	M.S.	6	X
P10	B.S.	6	
P11	M.S.	10	X
P12	B.S.	3	X

Navigating the Job Market

When interviewing participants, it became apparent that a significant percentage had gotten their full-time software engineers or developers positions through internships,

referrals, or moving internally within the company. Mention of any of these mechanisms for career advancement was recorded and never explicitly asked, showing that they might be familiar to avoid the friction of transitioning between jobs or the interviewing process as a whole. Obtaining employment through regular interviews was assumed for all candidates. Figure 4.11 reflects different alternative means for finding employment that the participants used.

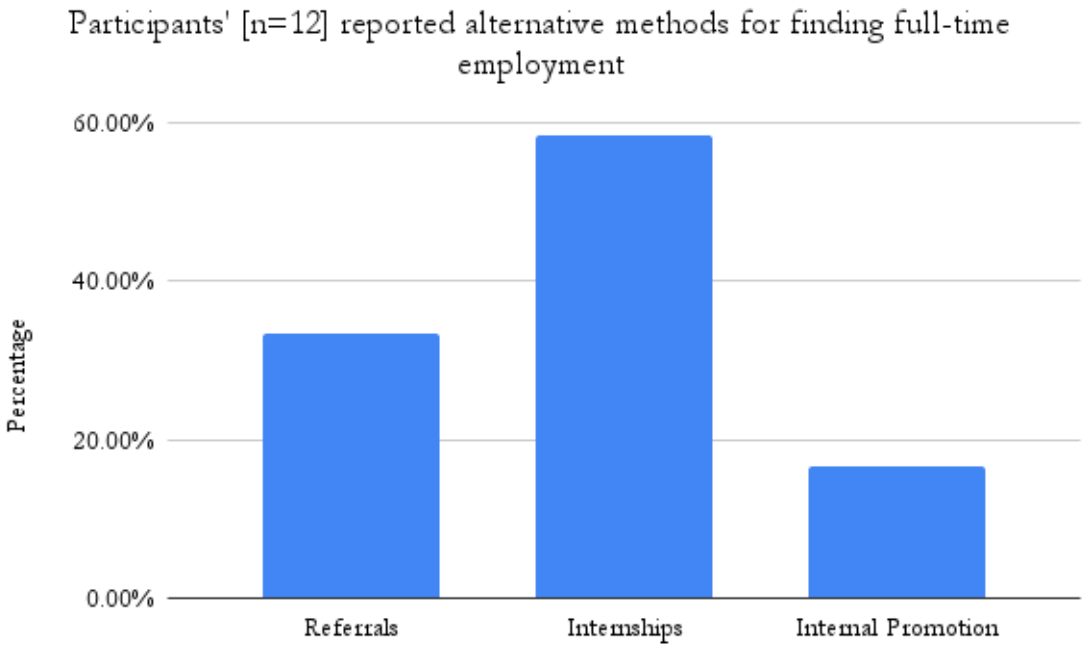


Figure 4.11: Participants' [n=12] reported alternative methods for finding full-time employment

Interview Pipeline

Half of the participants complained that the interview process for software engineering positions is too long. Five candidates complained about the process being too generic and not targeted towards the position or the interviewee's experience. In that regard, a third of the participants mentioned that the interview process should cater to the role companies are looking to fill and the seniority level. For example, junior engineers should be evaluated differently than senior engineers or engineers in directory or executive positions.

Similarly, a third of the participants reported poor interviewing experiences due to the company representative conducting the interview process. A candidate outlined in detail how they were rejected for a position for not providing a specific solution to a problem when interviewing for a large car manufacturing company. Even though the answer was correct, it did not satisfy the interviewer. This shows that bias remains present in the interviewing process. Even though the process and problem framing are said to be favored over the solution, there can be cases where a lack of training for interviewers or personal bias can affect the decision to hire a successful candidate.

Another candidate reported frustration related to a lack of transparency from interviewers, who failed to foreshadow the candidate about the interviewing process and the inclusion of technical assessments to evaluate abilities needed for the job. The participant did not know they would be subjected to an evaluation until the day of the interview. Another participant reported that they thought the technical assessment could be completed in any programming language but found out that only a specific language could be used during the assessment.

Similarly, a third of candidates expressed frustration over the slow interviewing process, lack of communication, or ceasing communication completely with candidates without notice or reason. This data point could be correlated to job interviewing overall and not necessarily unique to the software engineering and technology industries.

4.2.2 Experience with Timed Coding Assessments

Exposure

Another relevant data point to collect was how exposed the candidates were to timed coding assessments. All of the candidates had exposure to these assessments, some more than others. Figure 4.12 reflects the degree to which candidates engaged with timed coding assessments. It can be noticed that more than half of the participants had ample exposure to these types of assessments, meaning they encountered them more than twice in their careers. Limited exposure refers to participants who only encountered them once or twice in their career, either due to fewer years of experience

or alternative methods for finding employment.

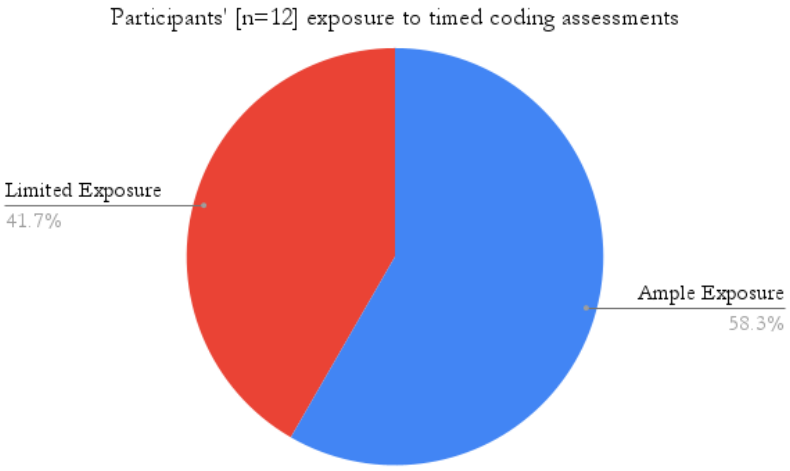


Figure 4.12: Participants' [n=12] exposure to timed coding assessments

Success

It must be emphasized that from all twelve participants interviewed, only one participant declared obtaining employment after completing a timed coding assessment. The rest of the participants had poor performance in the assessment or failed to clear any subsequent interview rounds. Seemingly, timed coding assessments are strict evaluation methods, whereas other alternatives, such as take-home assessments, verbal interviews, and live coding assessments, are less strict and allowed the remaining eleven participants to find employment. Some correlations could be drawn from this information: the unique successful participant also held a Ph.D., a degree that notoriously tests candidates strictly and regularly. However, with a limited size and variance in the sample pool, this must be considered just an observation.

A quarter of the candidates reported struggling with self-confidence after failing to clear timed coding assessments, and a third of the participants compared timed assessments to standardized testing, usually used during academic admissions, to assess student qualifications and abilities. As Section 2.6.4 outlines, standardized testing favors those with financial and time resources and does not concretely measure knowledge or expertise.

Efficiency and Efficacy

A fundamental data point to collect during this section of the semi-structured interview was the candidate's opinion on the efficiency and efficacy of timed coding assessments. Participants were also asked to qualify their personal experience with timed coding assessments. General feelings and opinions about this interviewing technique were recorded. It is essential to consider that participants were encouraged to share more than one experience with assessments, yielding candidates that had both good and bad experiences on separate occasions.

A single participant qualified timed coding assessments as effective but clarified their bias due to their current managerial position and overall work history that required working with cutting-edge technology in large technology firms. The remaining candidates categorized timed coding assessments as ineffective in assessing technical skills. Four of those eleven participants described that the method is helpful mainly for filtering poor candidates. Still, good candidates could have been filtered out due to the harshness and difficulty of timed coding assessments.

Eleven candidates agreed that timed coding assessments measure technical ability. A third of the group also thought that the capacity to extract information is measured during these examinations. Even though both of these qualifications are extremely important for a software engineer, participants did not mention communication abilities or soft skills as evaluation criteria during these assessments. One candidate shared their opinion on the matter, stating that they have seen candidates not clearing the assessment but having outstanding work performance when allowed to join the organization, including the tenacity to solve problems and having initiative and ownership of their work.

Preparation

Considering preparation is a fundamental aspect of interviewing for software positions, an entire section of the interview was dedicated to understanding what is common among candidates when studying for interviews. Eleven participants cited using on-

line tools like LeetCode and HackerRank to practice problems. This method was the only one used consistently across candidates, showing that it might provide the best benefits during interview preparation.

The second most used resource was books, which a third of the participants reported using at some point during the preparation. Volunteers also mentioned using other tools, such as evaluating old class notes, working on personal projects, participating in online communities, and engaging in mock interviews. At most, two participants reported using these methods individually, showing that the preparation journey is also personal and that different people might benefit from different methods, depending on their strengths and weaknesses.

Next, participants were also asked to share how long they prepared for interviews using the tools mentioned. A third of the group mentioned the preparation took too much time. A volunteer expressed dreading the process since it felt daunting, but they could get momentum and improve their technical interviewing skills with enough practice. More than a third of the volunteers reported to not preparing enough. Participants who did not clear the timed coding assessments for jobs they were pursuing understood they needed to prepare more and adapted their studying schedules and strategies. Considering what the bigger pool of candidates answered on the survey, illustrated on Figure 4.7, it could be possible that those candidates who used little time to prepare failed the assessments, and engaged in more extensive preparation for their next interview opportunity.

Finally, participants were asked about their outlook on the preparation process: if they enjoyed the task or if it felt like a burden. More than a third of the participants cited not liking the process and feeling it was a lot of extra work. Still, after some time, the preparation became enjoyable. This could be attributed to the improvement they saw from the preparation, which could have diminished the frustrations. A fourth of the participants flat-out reported not enjoying the process at all. A volunteer shared that common problems showcased on popular online practicing sites became repetitive and boring, encouraging this participant to attempt solving problems in different programming languages to remain engaged.

4.2.3 Experience with Other Interviewing Techniques

Since volunteers were allowed to discuss other experiences outside of timed coding assessments, the different interview methods they experienced and their frequency were also recorded. All participants cited engaging in verbal interviews that included technical questions. During these interviews, interviewees are probed for concrete technology knowledge and must provide facts or opinions about technical topics. Sixty-six percent of participants experienced live coding assessments, where they were asked to write code in the presence of an interviewer while also explaining their thinking process. These two were the most common methods used.

Some other less popular methods included aptitude tests. In these tests, participants were evaluated on their knowledge of mathematics. A volunteer who experienced this method of interviewing failed to see the correlation to day-to-day responsibilities for a software engineering position in a large technology firm and shared that "it was unnecessary and a waste of time." However, the volunteer saw a clear relationship between aptitude tests and other technical roles, such as data analysis and statistics, pointing out that the interviewing process should be modified depending on the role that will be filled.

Take-home assessments were also somewhat popular and appreciated by participants since they allowed showcasing programming skills without the pressure of an interviewer being present and increased cognitive load. The other less popular method but still prevalent was code reviews, where candidates are expected to evaluate a section of code, understand the code, suggest improvements, and optimize runtime. Volunteers were also asked to select a method that they believe could evaluate candidates' best, but this will be discussed in Section 4.2.5

4.2.4 Correlation of Interviews with Job Duties

An important question posed at the beginning of the research was whether timed coding assessments and overall technical interviewing are actually correlated to the job duties once the candidate is hired. This was a fundamental question during the

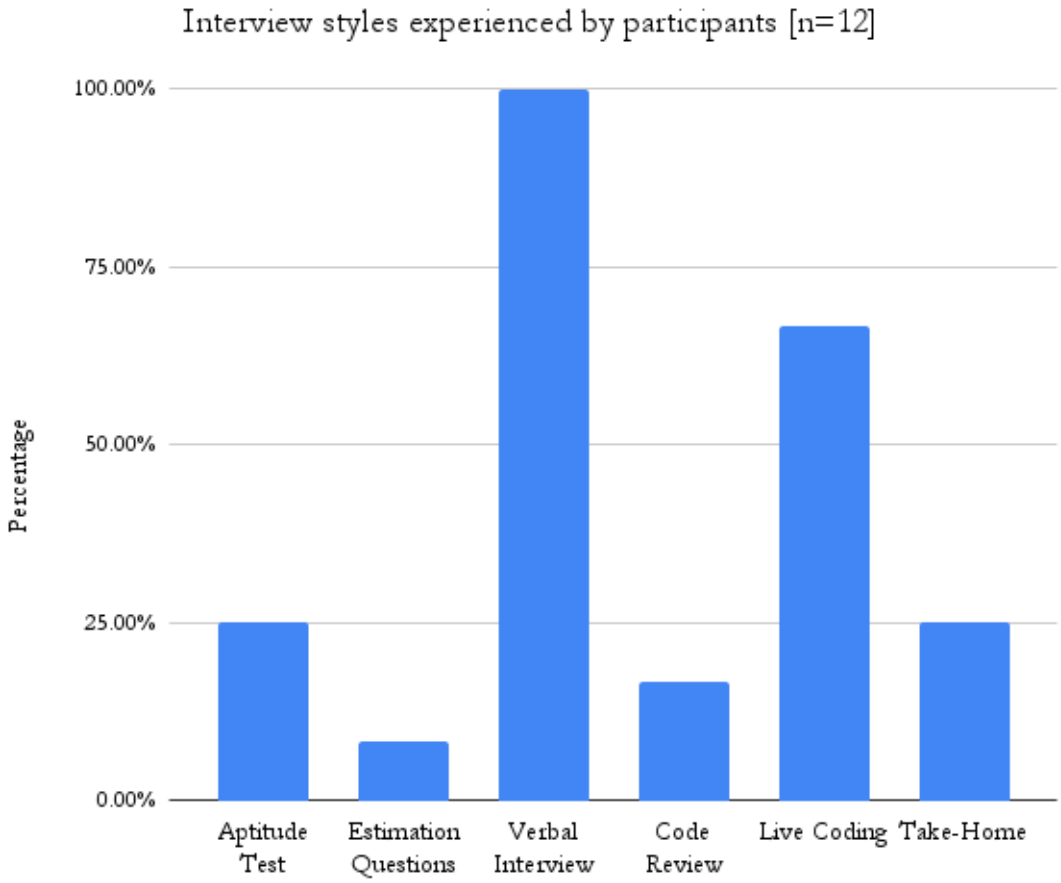


Figure 4.13: Interview styles experienced by participants [n=12]

semi-structured interview process, allowing candidates to concretely correlate some of the questions they were asked during the interviewing rounds and skills that were fundamental for the job. Since all participants but one failed to clear the timed coding assessment as part of their interview process, this question was modified to allow for the correlation of the evaluation method, not necessarily asynchronous timed assessments, with the job responsibilities. Figure 4.14 shows the distribution.

Most participants saw a correlation between the interview method and their job tasks once hired, but only by a small margin. Those who correlated their daily responsibilities to the evaluation were subject to verbal interviews, code reviews, live coding, and take-home assessments. Still, within these interview styles, some participants shared frustration about the lack of correlation, showing that it is not

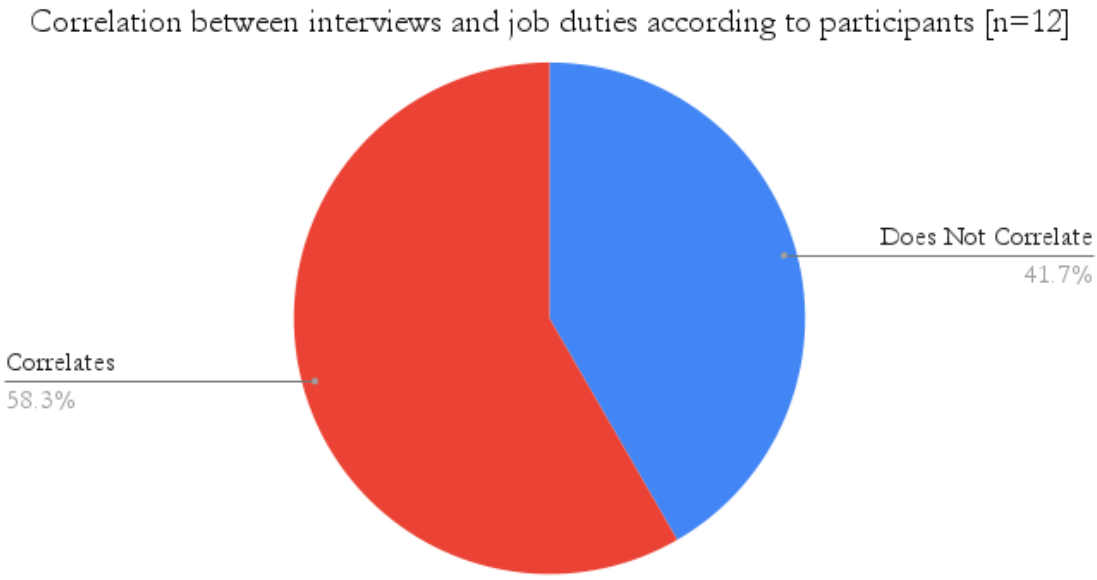


Figure 4.14: Correlation between interviews and job duties according to participants [n=12]

the method that defines the correlation but the questions used with the alternative interviewing style. Additionally, several participants shared frustration around the questions asked in technical interviews, suggesting they have become brain teasers instead of an actual measurement of knowledge. A volunteer shared that they had noticed how some candidates failed to clear the technical screening in interviews but that had personalities that allowed them to excel in the organization.

The participant who was successful in clearing the timed coding assessment reported that the interview evaluation correlated to their daily tasks. Still, with such a small sampling pool, it is impossible to draw a concrete conclusion about timed asynchronous assessments and the positions they are used to recruit for. Participants who described the process as correlated with their jobs still shared that not everything can be screened in the process but that their technical questions and tasks made sense in retrospect and were accurately tricky for the position they were applying for.

Another question posed for volunteers in this section was to describe skills that were not evaluated in the interviewing process but were fundamental to carrying out their jobs successfully. In this section, a fourth of the participants mentioned version

control and specific platform knowledge. A volunteer shared that organizations tend to assume proficiency in such categories, especially version control, and that the onboarding process might neglect to cover these topics and the best organizational practices around them. This observation ties in with the gap in academic curricula surrounding communication and teamwork since version control is a tool fundamental for these vital processes.

The process for hiring software engineers is highly customizable, and it can be made or broken by those leading interviews and designing the interview processes. Interviews are highly pressured time windows that only allow for evaluating so much, so organizations must carefully choose how to evaluate candidates for both parties benefit. Candidates will show frustration if assessments are not correlated, and organizations might under or over-evaluate candidates, incurring high expenses.

4.2.5 Managerial Perspective

Considering the bias in opinion about timed coding assessments for managers and candidates was relevant for this research. Candidates might feel the process is entirely unnecessary, while managers might put too much weight on a candidate's performance in the assessment to judge them for the position. Considering their hiring expertise and exposure to different candidate assessment methods, volunteers were asked what methods they believed were best for evaluating potential employees. Figure 4.15 shows the breakdown in percentages.

Half of those interviewed agreed that technical questions are helpful to both break the ice with the interviewee and to understand their technical proficiency at a higher level. Code reviews, live coding, and system thinking questions were the runner-up approaches in popularity: volunteers agree code reviews are expected already in the industry, and understanding code is as important as knowing how to write it. Live coding exposes the candidate's thinking process, and system design questions are fundamental to creating projects and applications that are maintainable in the long run and trivial to implement. A third of the volunteers agreed that considering a candidate's past projects is also a good option. Knowing that volunteers could list a

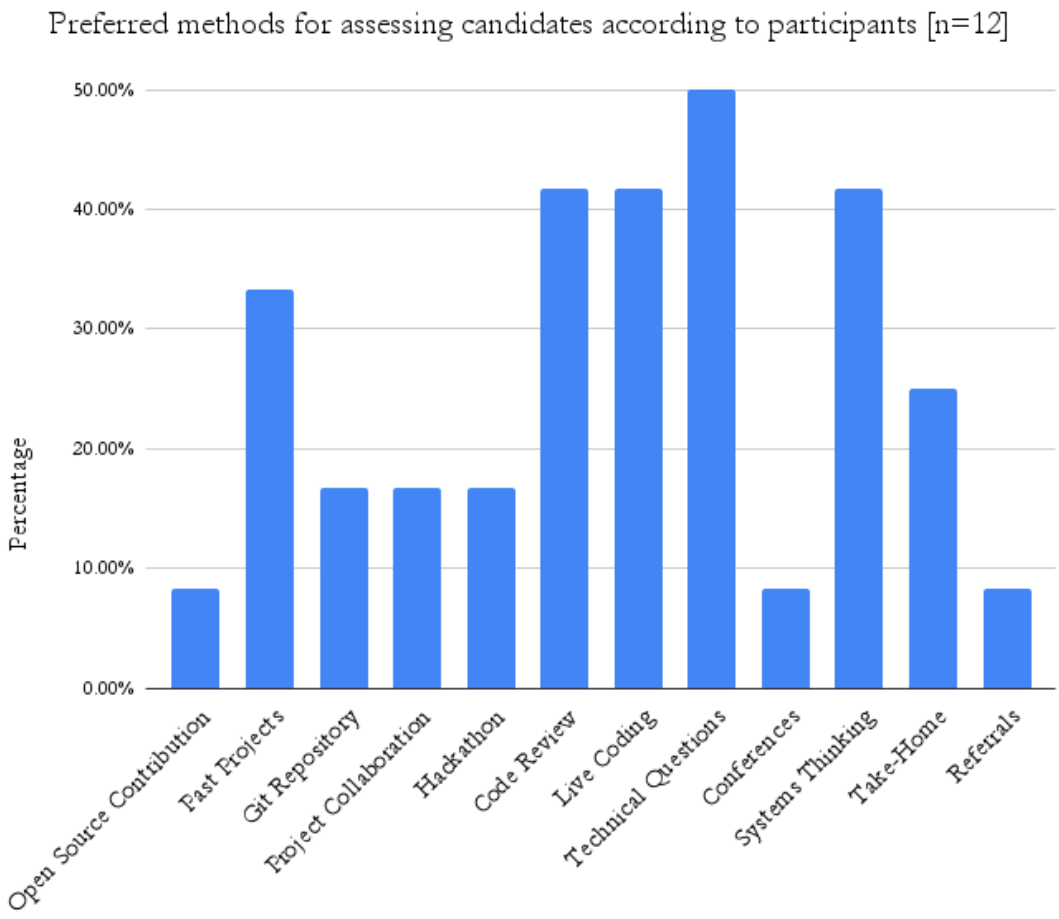


Figure 4.15: Preferred methods for assessing candidates according to participants [n=12]

combination of methods, not a single tool, is fundamental. Another observation from this section is that no participants mentioned using timed coding assessments.

Improving Interview Processes

The purpose of asking volunteers if they participated in the hiring process at their companies was to understand if the interview processes were defined by interviewers or someone else at the company. Smaller organizations appear to allow their interviewers to select the methods and questions. For larger organizations, a due process was observed to be defined that interviewers had to follow. However, some organizations were receptive to feedback from candidates they interviewed and modified their

interviewing processes accordingly to provide a better experience. A third of the volunteers shared that their feedback was heard and used to improve the interviewing process for efficiency and to allow for finding better candidates.

A volunteer recalls that once hired, they suggested improvements to the company's examination, suggesting tweaks that would clarify the assessment and involve less setting up time for the candidate. This allowed future candidates to do only work being evaluated for the position and not environment setup, which included creating and labeling data tables. The volunteer confessed their company was excited about incorporating changes and did so after some time.

Another volunteer with experience designing an engineering hiring process offered insights about the conceptualization process for their interviews. They recognized they needed to provide different types of assessments, one for front-end engineers and another for back-end developers, and that they were cognizant of the bias of allowing candidates too much time to complete the take-home assessments. At this volunteer's company, candidates had a few hours to complete the take-home to level out the playing field between candidates: those with other commitments and responsibilities would be affected negatively, while those with free time would perform best. This shows managers can be aware of biases and improve their hiring processes accordingly.

Another participant mentioned that their company started administering timed coding assessments once they were on board. A colleague from this participant firmly opposed the use of online assessments for evaluating candidates since they thought they would fail the assessment if they were asked to clear it, but they were good at their job and could manage the responsibilities. In other words, this volunteer's coworker did not see the relationship between timed coding assessments and their day-to-day responsibilities at the firm.

A Successful Software Engineer

As part of one of the final exercises, candidates were asked to describe and list skills that are fundamental in a successful software engineer, hoping to correlate these skills with those assessed by timed coding assessments. Curiously, the only skill that

matched these two categories was technical ability. Volunteers did not consider any of the remaining good qualities of a software engineer to be evaluated by a timed coding assessment. Figure 4.16 illustrates the frequency of answers across those interviewed, considering that more than one trait could be mentioned in this section.

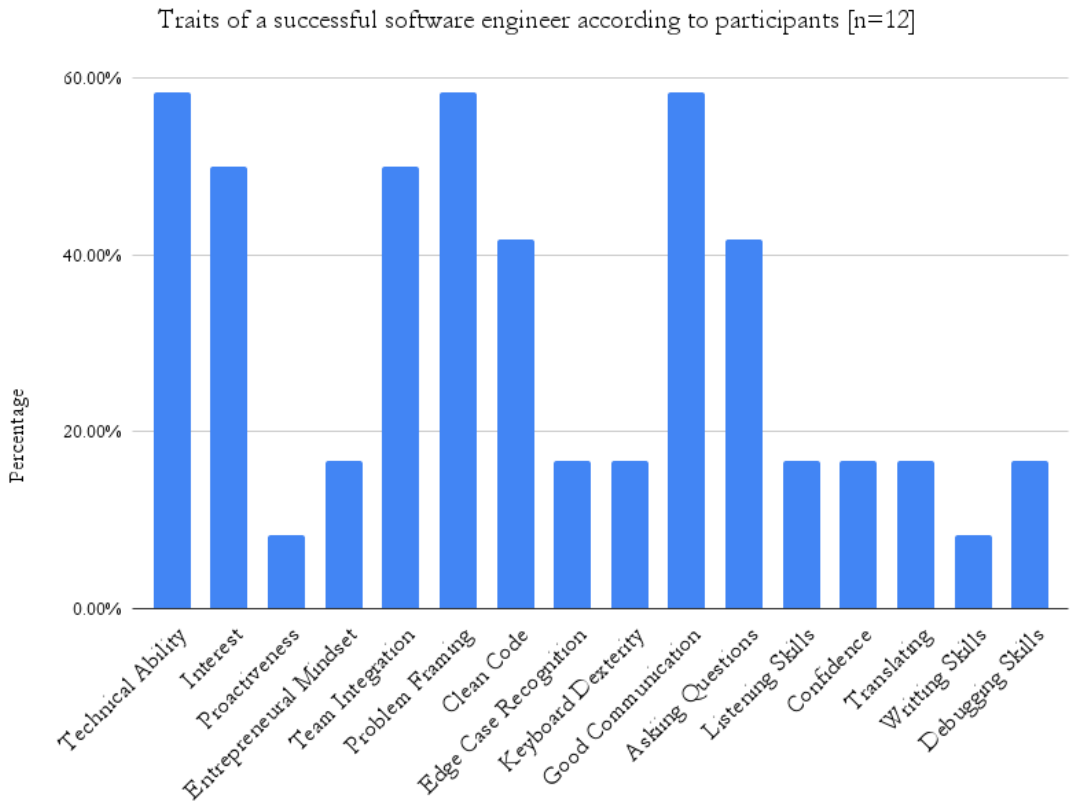


Figure 4.16: Traits of a successful software engineer according to participants [n=12]

Volunteers also agreed that problem-framing and good communication are fundamental skills in software engineering. Problem framing can be evaluated during live coding interviews and communication skills, but not by timed coding assessments, where candidates are secluded to answering algorithmic questions by themselves in a computer. Other relevant traits considered are interest and integration with the existing team and soft skills that cannot be measured easily through an asynchronous examination.

4.2.6 Sentiment Analysis

The sentiment of each participant's recorded transcript was analyzed at the end. The interviewer qualitatively categorized the tone and sentiment of the transcript, which was compared to the tone detected by the sentiment analysis NLP tool. Table 4.4 illustrates the obtained results. The level of confidence provided by the NLP tool was also recorded.

Table 4.4: Sentiment Analysis

Participant	Sentiment (Interviewer)	Sentiment (NPL)	Level of Confidence
P1	Negative	Negative	70.5%
P2	Positive	Positive	40.9%
P3	Negative	Negative	36.5%
P4	Negative	Positive	58.8%
P5	Negative	Negative	57.7%
P6	Negative	Negative	57.6%
P7	Negative	Negative	57.6%
P8	Negative	Positive	44.2%
P9	Negative	Negative	41.0%
P10	Negative	Negative	59%
P11	Negative	Positive	52.6%
P12	Negative	Negative	52.1%

The NLP tool's judgment matched the interviewer's judgment for nine participants. For the remaining three participants, even though the sentiment did not match, the confidence level in the NLP tool was lower than 60%. Considering the NLP qualifications, two-thirds of the participants have negative feelings toward time coding assessments and interviewing for positions in software engineering overall since the entire interview script was provided to the NLP sentiment analysis tool. It is essential to consider that during the interview, several topics surrounding the central theme of the thesis were discussed, and the entirety of the participants' answers were used to assess their sentiments. Refer to Appendix **??** for detailed questions asked of volunteers.

Chapter 5

Conclusions and Recommendations

Based on the analysis, this chapter outlines several conclusions and interesting observations that could be a foundation for future research about technical interviewing. Four recommendations are proposed based on these conclusions and observations that could allow organizations to better their interviewing processes and provide better experiences to candidates.

5.1 Conclusions

Consistency with Previous Studies

The demographic data was collected for two purposes: to understand the population being studied more in-depth and to draw a correlation with previous studies done in the software engineering field. This research showed that the sample of applicants was representative, showing demographic figures that resemble previously done studies on a larger scale. This provides additional credibility to the data collected concerning timed coding assessments.

Most of those surveyed and interviewed hold a bachelor's degree and have between 5 and 8 years of experience, consistent with previous studies. Furthermore, the interviewed volunteers echoed the lack of importance given to soft skills in the software engineering curricula and how that affects engineers in the workplace.

"Standard" Practice

Timed coding assessments are a standard part of interviewing processes in the software engineering industry. Professionals responded in the survey and interview portions of the research how common the practice of asynchronous assessments is to evaluate technical skills when applying for technical positions requiring code implementation. Figure 4.4 outlines how the assessments should be expected by candidates opting for these types of positions, and it is essential to consider that no survey respondents checked the option "never" in this question. Consequently, it can be expected that candidates will need to endure some technical examination during software engineering interviews to help measure their technical proficiency, especially in the technology sector.

A Filtering Method

Given the incredible popularity of the software engineering industry in the last decades, propelled by the creation of The Internet, companies have become more selective when hiring candidates. Timed coding assessments have been reported to feel like a filtering method by participants of this research, meaning that large organizations that do not have the resources to interview every seemingly qualified candidate need another tool to narrow the applicant pool. The large cost of hiring talent in the software engineering industry is a clear motivator for using this interviewing practice. Similarly, software engineers can expect to be high earners when hired into organizations, giving companies the right to become more selective with talent.

Despite the benefits of having a filtering method to reduce the applicant pool, good candidates might be filtered out in the process, just as underqualified individuals could clear the timed assessment and continue in the interviewing process. Asynchronous assessments are not a perfect method to reduce the applicant pool with complete confidence.

Preparation is Fundamental

Preparation for timed coding assessments and software engineering interviews, in general, was found to be fundamental. Participants reported that they found practice problem platforms to be the most effective in their preparation efforts, contributing to the financial success of such platforms.

The time used to prepare was observed to vary between candidates with different educational and experience profiles. During interviews, a third of the group mentioned that the interview preparation took longer than desired. A third of the group admitted to not preparing enough and having poor results in asynchronous assessments.

Preparation was either seen as a burden or became enjoyable, but after some time, perhaps due to the progress noticed by candidates. Another common thought in the preparation aspect was that participants were overall qualified for positions but attributed failing the assessments to lack of preparation, not due to lack of qualifications.

Finding Employment

A trend was observed in the interviewed participants where internships and internal promotions within a company were prevalent methods to transition to full-time software engineering positions. This could be attributed to the great friction candidates experience when interviewing with new organizations, requiring extra preparation. Contrary, if transitioning from an internship, candidates might be more familiar with the internal company culture and methods of working, providing an edge to internal candidates over external ones, and facilitating the transition to a high-paying software development position.

Negative Sentiment

Half of the participants interviewed explicitly mentioned experiencing inefficient, slow, and negative interviewing processes. The interview pipeline was reported to be too relaxed, diminishing their interest in the company. In addition, candidates reported

having negative experiences with interviewers, some demanding specific solutions to problems and not prioritizing problem framing in the answers to their questions.

Participants from the study also reflected that the process might be too generic and need to be tweaked depending on the position and seniority level. This supports the findings of considering timed coding assessments as filtering tools more than assessment tools. Similarly, some participants found aptitude testing irrelevant, and given their analogy with standardized testing, they can be considered heavily biased to favor those with time and financial resources.

The sentiment analysis supports this assumption by showing 8 negative and 4 positive interview transcripts with a somewhat high confidence level. The sentiment analysis by the interviewer detected 11 negative and 1 positive interview transcript, showing the natural language processing tool to be more conservative. Still, the proportion of negative transcripts outweighs the positive ones, showing candidates have strong negative feelings towards the interview process in the software engineering industry. Further, these feelings might not be exclusive to this industry and might be experienced across other non-technical industries, which could be further explored with future work.

Desirability of Soft Skills

Managers reported prioritizing soft skills when assessing the overall qualification of candidates for software engineering roles. When asked what timed coding assessments measured, the same pool of volunteers listed only technically relevant skills, which although fundamental, do not seem to be the priority of managers looking to source talent. Among common desirable traits for software engineers, technical ability is the only technical skill valued. The remaining listed attributes consisted of good communication skills and positive team integration.

Despite soft skills being the most sought-after in software engineers, they are deprioritized in academic curricula and when evaluating candidates. Timed coding assessments do not account for communication or positive team integration, making assessing these fundamental skills managers desire inefficient. There is a mismatch

between the skills sought after and those evaluated by asynchronous assessments.

5.2 Observations

From the synthesized data, some trends were observed that could be addressed in the software engineering industry to improve the hiring experience of candidates.

Manager vs. Candidate Opinions on Technical Interviews

Compared to managers, candidates have a slightly more pessimistic view of timed coding assessments and technical interviewing overall. Managers tend to see the process as minimizing risk and ensuring the candidate who is being evaluated checks all boxes before being offered a position. Candidates believe the assessments can become overcomplicated and sometimes lose relevancy concerning the job position. A dedicated study could be conducted to fully prove this observation, as some of the participants interviewed did not have hiring experience and were asked to put themselves hypothetically in the hiring manager position.

Relevancy of Technical Interviews

Timed coding assessments and interview methods can be relevant to the job duties of the position they seek to assess. The study participants seemed to have a homogenous view in this matter: some saw the relevancy and relationship of the questions asked in the position they later filled. In contrast, others saw no relationship or relevance at all. A small margin favors those who saw relevance for methods different than timed coding assessments, such as live coding assessments, take-home examinations, and code reviews.

Even though timed coding assessments and technical interviewing have become standard practice, there is no standard method, meaning companies, organizations, and interviews have complete freedom to assess candidates and provide different experiences. This could be why some candidates reported seeing relevancy, while others did not.

Difficulty of Timed Coding Assessments

Of the participants who agreed to be interviewed, only one could find employment by clearing a timed coding assessment. Curiously, this participant was the only one holding a doctorate. This data point could suggest that higher preparation and education and academic environments could have a positive benefit when it comes to success in asynchronous assessments.

The remaining interview participants suggested that the timed coding assessments they experienced were too tricky and borderline brain-teasing, and even though they did not have experience working for those competitive companies, they were skeptical of the relevancy of the assessment. It is important to reflect all candidates interviewed had relevant experience in the software engineering field.

5.3 Recommendations

From the conclusions and observations outlined above, the following recommendations could continue improving how hiring is conducted for software engineering roles:

1. **Organizations should consider desirable skills in the composition of evaluation methods.** Even though technical proficiency is fundamental for success in a software-centric role, managers seem to prioritize communication skills and team integration, considering technical proficiency can be learned and improved more easily.

2. **Companies should adapt timed coding assessments to the role.** Asynchronous assessments have been reported to be too generic, favoring junior candidates with expertise mainly in back-end software development. A positive recommendation would be to adapt the questions asked on timed coding assessments to fit description roles better or to be skipped altogether for more senior candidates. Candidates with credible work and educational experience should be assessed accordingly, providing a seamless experience and reducing the costs of hiring in the industry,

3. **Companies should be aware of the role of the interviewer.** Organizations must be mindful of the training provided to interviewers, considering they might make or break the experience of candidates. Prioritizing problem resolution and communication skills is said to be fundamental, but participants reported the contrary was true in some situations. Companies looking to streamline their hiring pipeline and retain a reputable image regarding the treatment of candidates need to evaluate their interviewing practices carefully and who is allowed to lead these conversations. On a similar note, candidates need to be foreshadowed on what to expect during the interview and evaluation process, since participants reported at times that they relied on online communities to know what to expect fully or were altogether surprised the day of their interviews due to the lack of transparency and communication with hiring managers and recruiters.

4. **Organizations must consider and minimize bias.** Organizations need to consider what method of interviewing minimizes and reduces bias the most. Even though it is challenging to find a bulletproof method since humans have an inherent bias, it is possible to minimize it by considering methods that truly evaluate experience and not preparedness. Preparedness can always be obtained with more time and financial resources, while experience and qualifications should be of uttermost importance when assessing a candidate's profile.

5.4 Future Work

Considering the limitations of the research, outlined in Section 1.3, an even more comprehensive candidate pool could be sourced to continue recording experiences and opinions about timed coding assessments and technical interviews. Expanding into a more global population could further unbias the data, providing a clearer picture of global working cultures and their thoughts on technical skill evaluation methods.

Some interesting insights might be found from internal studies carried out at companies. Since more prominent companies have their processes, those must have

been informed by feedback, surveys, and other data collection methods.

Another important observation during the interviews was the mention of assessing a candidate's potential. There is no standard method or test to evaluate a candidate's potential at the organization interviewing, since multiple factors contribute to potential, such as previous experience, personality, academic record, and professional strengths and weaknesses. Still, the candidate might not be a good match given the company culture. If organizations could assess a candidate's potential and the impact they could have on the organization, it could significantly help improve hiring in the software engineering industry.

References

[1] Mahnaz Behroozi, Chris Brown, and Chris Parnin. Asynchronous technical interviews: reducing the effect of supervised think-aloud on communication ability. In *ESEC/FSE 2022: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 294–305, 2022.

[2] Mahnaz Behroozi, Alison Lui, Ian Moore, Denae Ford, and Chris Parnin. Dazed: Measuring the cognitive load of solving technical interview problems at the whiteboard. In *2018 IEEE/ACM 40th International Conference on Software Engineering: New Ideas and Emerging Technologies Results (ICSE-NIER)*, pages 93–96, 2018.

[3] Mahnaz Behroozi, Chris Parnin, and Titus Barik. Hiring is broken: What do developers say about technical interviews? In *2019 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 1–9, 2019.

[4] Mahnaz Behroozi, Shivani Shirolkar, Titus Barik, and Chris Parnin. Does stress impact technical interview performance? In *ESEC/FSE 2020: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 481–492, 2020.

[5] Marc Blatter, Samuel Muehlemann, and Samuel Schenker. The costs of hiring skilled workers. *European Economic Review*, 56(1):20–35, 2012.

[6] Britannica. Computers. https://www.britannica.com/technology/computer/The-Turing-machine. Accessed: 2023-11-25.

[7] Frederick P. Brooks. *The mythical man-month – Essays on Software-Engineering.* Addison-Wesley, 1975.

[8] Frederick P. Brooks. No silver bullet: Essense and accidents of software engineering. *IEEE Computer*, 20(4):10–19, April 1987.

[9] Luis F. Capretz. Bringing the human factor to software engineering. *IEEE Software*, 31(2):104–104, 2014.

[10] James W. Cortada. Building the system/360 mainframe nearly destroyed ibm. https://spectrum.ieee.org/building-the-system360-mainframe-nearly-destroyed-ibm. Accessed: 2023-11-26.

[11] Coursera. What is a data engineer? a guide to this in-demand career. https://www.coursera.org/articles/what-does-a-data-engineer-do-and-how-do-i-become-one. Accessed: 2023-11-24.

[12] Ezekiel J. Dixon-Roman, Howard T. Everson, and John J. Mcardle. Race, poverty and sat scores: Modeling the influences of family income on black and white high school students' sat performance. *Teachers College Record*, 115(4):1–33, April 2013.

[13] Nathan Doctor. The hidden costo of hiring software engineers - $22,750/hire. https://www.qualified.io/blog/posts/the-hidden-cost-of-hiring-software-engineers. Accessed: 2023-11-25.

[14] National Center for Education Statistics. Average undergraduate tuition, fees, room and board for full-time students in degree-granting postsecondary institutions. https://nces.ed.gov/programs/digest/d20/tables/dt20_330.20.asp. Accessed: 2023-11-24.

[15] Denae Ford, Titus Barik, Leslie Rand-Pickett, and Chris Parnin. The tech-talk balance: What technical interviewers expect from technical candidates. In *2017 IEEE/ACM 10th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, pages 43–48, 2017.

[16] Alice George. Margaret hamilton led the nasa software team that landed astronauts on the moon. https://www.smithsonianmag.com/smithsonian-institution/margaret-hamilton-led-nasa-software-team-landed-astronauts-moon-180971575/. Accessed: 2023-11-26.

[17] Raffaele Giordanelli and Carlo Mastroianni. The cloud computer paradigm: Characteristics, opportunities and research issues. In *Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR)*, pages 1–23, 2010.

[18] Glassdoor. What does a front end engineer do? https://www.glassdoor.com/Career/front-end-engineer-career_KO0,18.htm. Accessed: 2023-11-24.

[19] Sheryl Grey. Are coding bootcamps worth it? https://www.forbes.com/advisor/education/are-coding-bootcamps-worth-it/. Accessed: 2023-11-24.

[20] HackerRank. Hackerrank. https://www.hackerrank.com/. Accessed: 2023-12-11.

[21] Phillip Hall and Kinnis Gosha. The effects of anxiety and preparation on performance in technical interviews for hbcu computer science majors. In *SIGMIS-CPR'18: Proceedings of the 2018 ACM SIGMIS Conference on Computers and People Research*, pages 64–69, 2018.

[22] Watts S. Humphrey. The software engineering process: definition and scope. *AGM SIGSOFT Software Engineering Notes*, 14(4):82–83, June 1989.

[23] Indeed. How to become a software testing engineer. https://www.indeed.com/career-advice/finding-a-job/software-testing-engineer. Accessed: 2023-11-24.

[24] Kinsta. Software engineering statistics: Market share, trends, and growth patterns. https://kinsta.com/software-engineering-statistics/. Accessed: 2023-11-24.

[25] Gayle Laakmann McDowell. *Cracking the Coding Interview*. CareerCup, LLC, 2016.

[26] Nancy Leveson and Clark Turner. An investigation of the therac-25 accidents. *Computer*, 26(7):18–41, 1993.

[27] Michael S. Mahoney. The roots of software engineering. *CWI Quarterly 3*, 1990.

[28] Meredith Morris, Andrew Begel, and Ben Wiedermann. Understanding the challenges faced by neurodiverse software engineering employees: Towards a more inclusive and productive technical force. In *the 17th International ACM SIGACCESS Conference*, pages 173–184, 2015.

[29] Mohammad Moshirpour, Robyn Paul, Zain Rizvi, and Hadi Hemmanti. Designing a programming bootcamp for non-software engineers. In *Proceedings 2019 Canadian Engineering Education Association*, 2019.

[30] U.S Bureau of Labor Statistics. Employment projections: Calculation. https://www.bls.gov/opub/hom/emp/calculation.htm. Accessed: 2023-11-25.

[31] U.S Bureau of Labor Statistics. Occupational outlook handbook: Sofware developers, quality assurance analysts, and testers. https://www.bls.gov/ooh/computer-and-information-technology/software-developers.html. Accessed: 2023-11-24.

[32] Damla Oguz and Kaya Oguz. Perspectives on the gap between the software industry and the software engineering education. *IEEE Access*, 2019.

[33] Stack Overflow. 2021 developer survey. https://insights.stackoverflow.com/survey-/2021section-experience-learning-how-to-code. Accessed: 2023-11-24.

[34] Stack Overflow. Average salaries of software developers worldwide as of 2023, by role. https://www-statista-com.libproxy.mit.edu/statistics/793602/worldwide-developer-survey-average-salaries/. Accessed: 2023-11-26.

[35] Stack Overflow and Amazon. Levels of formal education for software developers, worldwide, as of 2023. https://www-statista-com.libproxy.mit.edu/statistics/793568/worldwide-developer-survey-level-formal-education/. Accessed: 2023-11-26.

[36] David L. Parnas. Software engineering programs are not computer science programs. *IEEE Software*, 1999.

[37] PitchBook. Leetcode overview. https://pitchbook.com/profiles/company/227792-62overview. Accessed: 2023-12-11.

[38] RedHat. Who is a devops engineer? https://www.redhat.com/en/topics/devops/-devops-engineer: :text=easy Accessed: 2023-11-24.

[39] IEEE Computer Society. Software engineering body of knowledge. https://www.statista.com/outlook/tmo/software/worldwide. Accessed: 2023-11-24.

[40] Statista. Market insights: Software - worldwide. https://www.computer.org/education/bodies-of-knowledge/software-engineering. Accessed: 2023-11-24.

[41] Michael Wales. 3 web dev careers decoded: Front-end vs back-end vs full stack. https://www.udacity.com/blog/2020/12/front-end-vs-back-end-vs-full-stack-web-developers.html. Accessed: 2023-11-24.

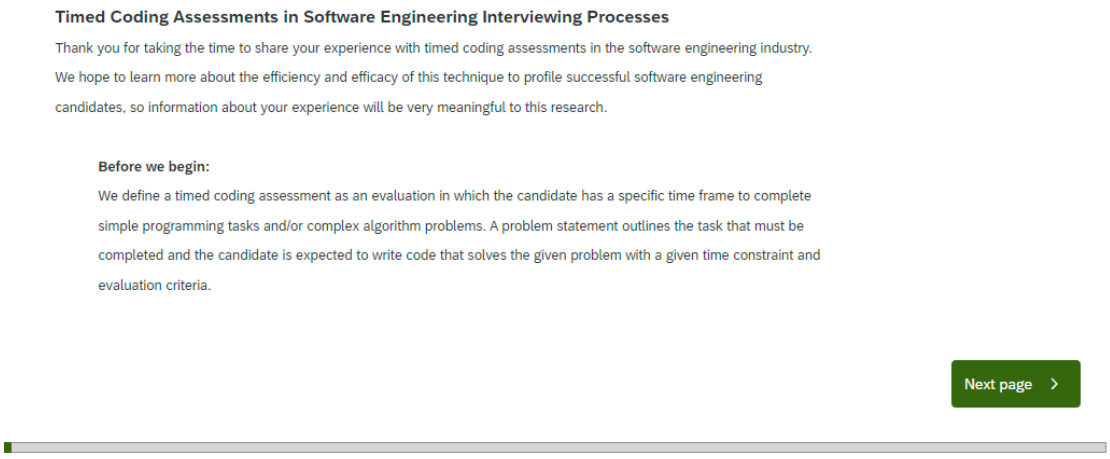
[42] Anthony I. Wasserman. How the internet transformed the software industry. *Journal of Internet Services and Applications*, 2(1):11–22, May 2011.

[43] George F. Weinwurm. Managing the economics of computer programming. In *ACM '68: Proceedings of the 1968 23rd ACM national conference*, 1968.

[44] Marvin Wyrich, Daniel Graziotin, and Stefan Wagner. A theory on individual characteristics of successful coding challenge solvers. *PeerJ Computer Science*, 2019.

Appendix A

Survey



*What is your most recent job title?

- ◯ Software engineer
- ◯ Programmer
- ◯ Data analyst
- ◯ Developer
- ◯ Other

*How many years of experience do you have?

- ◯ 0-2
- ◯ 2-5
- ◯ 5-8
- ◯ 8-10
- ◯ 10+

*What is the highest degree of education you have completed?

- ◯ Less than highschool
- ◯ Highschool degree or equivalent
- ◯ Some college, no degree
- ◯ Associate's degree
- ◯ Bachelor's degree
- ◯ Master's degree
- ◯ Doctorate or professional degree

Next page >

*Have you ever taken a timed coding assessment as part of a job interview?

- ◯ Yes
- ◯ No

*Across your professional career, how often did you have to take a timed coding assessment as part of a job interview?

- ◯ Always
- ◯ Most of the time
- ◯ About half the time
- ◯ Sometimes
- ◯ Never

*What methods have you used to prepare for timed coding assessments?

- ☐ None
- ☐ Leetcode, CodeChef, other platforms for practicing questions.
- ☐ Courses, bootcamps, other structured learning
- ☐ Books
- ☐ Other

*How much time in advanced have you used to prepare for timed coding assessments?

- ◯ 1-5 days
- ◯ 7-10 days
- ◯ 2-4 weeks
- ◯ 1 month+

*Across your professional career, how many times did you proceed to the next step after a timed coding assessment?

- ◯ Always
- ◯ Most of the time
- ◯ About half the time
- ◯ Sometimes
- ◯ Never

*In your experience as a candidate, what skills do timed coding assessments judge? Score the options below

Not considered — Greatly considered

Language proficiency and syntax

0 — 5

Problem framing

0 — 5

Computer science theory (algorithms and data structures)

0 — 5

Troubleshooting and debugging

0 — 5

Version control

0 — 5

Runtime and algorithm performance

0 — 5

Time management

0 — 5

Code organization

0 — 5

Resource integration (libraries, frameworks, environments...)

0 — 5

Technical communication skills

0 — 5

Design and code architecture

0 — 5

*Rate your experience and feelings around taking timed coding assessments:

None at all — A great deal

Enjoyment from preparing for a timed coding assessment

0 — 5

Anxiety from preparing for a timed coding assessment

0 — 5

Enjoyment from taking a timed coding assessment

0 — 5

Anxiety from taking timed coding assessments

0 — 5

*In your experience as a candidate, how efficient are timed coding assessments to judge a candidates' technical qualifications? (Think of efficiency as minimizing the amount of time to profile a candidate's technical qualifications)

- ◯ Not efficient at all
- ◯ Slightly efficient
- ◯ Moderately efficient
- ◯ Very efficient
- ◯ Extremely efficient

*In your experience as a candidate, how effective are timed coding assessments to judge a candidate's technical qualifications? (Think of effectiveness as the degree to timed coding assessments are successful to profile a candidate's technical qualifications)

- ◯ Not effective at all
- ◯ Slightly effective
- ◯ Moderately effective
- ◯ Very effective
- ◯ Extremely effective

Next page ›

For the following questions in this section, refer to a current or recent job position where you had to code more than 50% of the time

*Did you take a timed coding assessment as part of the interview process for this position?

- ◯ Yes
- ◯ No

Next page ›

*What other method was used by the interviewer to assess your technical qualifications for the position?

- ◯ Verbal interview
- ◯ Previous projects, repository or portfolio
- ◯ Take-home project/presentation
- ◯ Other

*What is/was your company size? (In employees)

- ◯ 0-10
- ◯ 10-49
- ◯ 50-249
- ◯ 250+

*What is/was the company industry?

- ◯ Technology
- ◯ Healthcare
- ◯ Entertainment and media
- ◯ Retail
- ◯ Finance
- ◯ Insurance
- ◯ Energy
- ◯ Government
- ◯ Other

*Where is/was the company located?

- ◯ North America (USA, Canada, Mexico)
- ◯ Central/South America
- ◯ Europe
- ◯ Middle East
- ◯ Asia
- ◯ Africa

*Is/was there a correlation between your daily responsibilities and the method used to evaluate your technical skills during the interview process?

- ◯ Definitely not
- ◯ Probably not
- ◯ Might or might not
- ◯ Probably yes
- ◯ Definitely yes

Next page ›

*Have you participated in the hiring process of other software engineers?

- ◯ No
- ◯ Yes

Next page ›

*Did this process involve giving the candidate a timed coding assessment?

- ◯ Yes
- ◯ No

Next page ›

*How did you assess the technical skills of the candidate?

- ◯ Verbal interview
- ◯ Previous projects, repository or portfolio
- ◯ Take-home project/presentation
- ◯ Other

*Were you successful at finding qualified candidates for the position?

- ◯ Definitely not
- ◯ Probably not
- ◯ Might or might not
- ◯ Probably yes
- ◯ Definitely yes

*In your experience as a hiring manager, what are the necessary technical skills for a successful software engineer? Score the options below

Not necessary — Absolutely necessary

Language proficiency and syntax

0 — 5

Problem framing

0 — 5

Computer science theory (algorithms and data structures)

0 — 5

Troubleshooting and debugging

0 — 5

Version control

0 — 5

Runtime and algorithm performance

0 — 5

Time management

0 — 5

Code organization

0 — 5

Resource integration (libraries, frameworks, environments...)

0 — 5

Technical communication skills

0 — 5

Design and code architecture

0 — 5

*In your experience as a hiring manager, how efficient are timed coding assessments to judge a candidates' technical qualifications? (Think of efficiency as minimizing the amount of time to profile a candidate's technical qualifications)

- ◯ Not efficient at all
- ◯ Slightly efficient
- ◯ Moderately efficient
- ◯ Very efficient
- ◯ Extremely efficient

*In your experience as a hiring manager, how effective are timed coding assessments to judge a candidate's technical qualifications? (Think of effectiveness as the degree to which timed coding assessments are successful to profile a candidate's technical qualifications)

- ◯ Not effective at all
- ◯ Slightly effective
- ◯ Moderately effective
- ◯ Very effective
- ◯ Extremely effective

*Do you think timed coding assessments are overall helpful to find right candidates for software engineering positions?

- ◯ Definitely not
- ◯ Probably not
- ◯ Might or might not
- ◯ Probably yes
- ◯ Definitely yes

Next page ›

*Are you interested in being interviewed to discuss your experience in more detail?

- ◯ No
- ◯ Yes

Next page ›

*Enter your full name

*Enter your email

Next page ›

Appendix B

Semi-Structured Interview Script

Information Disclosure

Thank you for agreeing to be interviewed for this research. I am looking to understand the efficiency and efficacy of timed coding assessments in the software engineering hiring process. Before we start, I would like to share that the information collected from this interview will only be used for the purpose of this research. All identifiable information, such as demographic and company information, will be masked and de-identified when sharing the results from this research. Do I have your consent to proceed with the interview?
I would also like to obtain your consent to audio record this interview to facilitate data collection and analysis down the line. Do I have your consent to record this session?

Introduction

- Most recent job title?
- Years of experience?
- Highest degree of education completed?
- Area of university degree?
- Motivation for getting into software engineering?

Experience with Coding Challenges

- Experience with coding challenges in Software Engineering?
  - If yes: what kind of coding challenges? How were they structured?
  - If no: have you heard of coding challenges before?
- How often did you take coding challenges as part of an interview?
- How is your preparation process for coding challenges?
- How much time did you allocate to prepare?
- What tools did you use to prepare?
- Did you use free or paid resources?
  - If paid: what were they? Were they helpful/worth the investment?
- What outcomes have you had from coding challenges?
- Positive/negative. Why do you think that?
- How do you feel about the process (preparing for and taking the coding challenge)?
- What feelings does it cause?
- Do you experience similar feelings for regular job interviews?
- Do you think coding challenges are efficient? Effective? Accommodating?
- What do you think is the concept or value behind them?
- What do you think are the primary skills that coding challenges measure?
- What would be your preferred method to assess your technical skills? In which kind of examination do you thrive?
- What is your ideal environment to successfully engage in software development?

Your Job and Coding Challenges

- Tell me about your current job, company size, company industry, and location.
- Do you code for more than 50% of the time?
  - If not: think of your latest position where you did code 50% of the time.
- What are your responsibilities?
- Did you take any coding challenges as part of the interviewing process?
  - If yes, please describe what you remember.
  - If not, how did they assess your skills?
- How would you describe the interviewing process? Was it smooth? Stressful?
- How long have you been working there?
- If moved on to another field, why?
- Do you see a correlation between your day-to-day responsibilities and the coding challenge you took as part of the interview?
- What fundamental skills were required in your day-to-day responsibilities that were not assessed during the interviewing process?
- Tell me about a different experience, in another company/job position. The same questions apply.

Experience Hiring Software Engineers

- Do you have experience interviewing or recruiting software engineers?
  - If yes: how much experience? Have you been doing this for a long time?
- In your opinion, what are some indicators of a qualified software engineer?
- What method was used to assess the technical competencies of the candidates?

- Was that company imposed?
- Do you agree or disagree with the methods used? Why?
- If coding challenges are mentioned:
  - Do you think there is a correlation between a successful coding challenge and indicators of a qualified software engineer?
  - Were you successful at finding good candidates using coding challenges?
  - Was the process efficient? Why?
  - Was the process effective? Why?

Wrap-Up

- With recent technology developments, do you think the hiring process for software engineers is bound to change in the following years? Why or why not?
- Do you have anyone on your network who could provide insightful information for this research?

Thank you for volunteering your time to help me with this research. We hope to understand more about timed-coding assessments from candidates' and managers' perspectives.