What Makes A Great Software Engineer?

Paul Luo Li*+, Andrew J. Ko*, Jiamin Zhu+

Microsoft+
Seattle, WA
{pal,jiaminz}@microsoft.com

The Information School*
University of Washington
ajko@uw.edu

Abstract—**Good software engineers are essential to the creation of good software. However, most of what we know about software-engineering expertise are vague stereotypes, such as 'excellent communicators' and 'great teammates'. The lack of specificity in our understanding hinders researchers from reasoning about them, employers from identifying them, and young engineers from becoming them. Our understanding also lacks breadth: what are all the distinguishing attributes of great engineers (technical expertise and beyond)? We took a first step in addressing these gaps by interviewing 59 experienced engineers across 13 divisions at Microsoft, uncovering 53 attributes of great engineers. We explain the attributes and examine how the most salient of these impact projects and teams. We discuss implications of this knowledge on research and the hiring and training of engineers.**

Index Terms—**Software engineers, expertise, teamwork**

I. Introduction

Software engineering research has considered a vast number of factors that affect project outcomes, from process and tools, to programming languages and requirement elicitation. We rarely give consideration, however, to one of the most fundamental components of software engineering: the engineers themselves. Specifically, what makes a software engineer great? This basic question is at the foundation of nearly every part of our world's rapidly growing software ecosystem: employers want to hire and retain great engineers, universities want to train great engineers, and young engineers want to become great. And yet our understanding of what characteristics define software engineering expertise still lacks specificity, breadth, and rigor.

The research we do have on this subject is directionally sound, but often too indirect or abstract to form a foundational understanding of software-engineering expertise. For example, some research has considered experiences of new hires [1][2], finding that engineers need to contribute value to the team, not become blocked (i.e. have self-efficacy and be persistent), and effectively navigate large organizations. Other research hints at important attributes, but only indirectly. For example, research on teaching novices [3] and programmer productivity [4][5] indicate experts are generally more productive: producing solutions faster, producing more in the same amount of time, and/or having fewer bugs.

Software engineering education research is another source of information about software engineering expertise, but it is prescriptive rather than descriptive. For example, several studies suggest what ought to be in the ACM Computing Curricula [6][7][8][9], arguing that engineers need knowledge of technical areas and techniques such as programming fundamentals, verification/validation, and project management.

In this study, we sought to remedy the lack of specificity, breadth, and rigor in prior work by investigating the following about *software* engineers:

- What do expert software engineers think are attributes of great software engineers?
- Why are these attributes important for the engineering of software?
- How do these attributes relate to each other?

To answer these questions, we performed 59 semi-structured interviews, spanning 13 Microsoft divisions, including several interviews with architect-level engineers with over 25 years of experience. The contribution of this effort is a thorough, specific, and contextual understanding of software engineering expertise, as viewed by expert software engineers.

In the rest of this paper, we detail our current understanding of software engineering expertise. We then discuss our interview and analysis methodology, the attributes we discovered, and the implications of this knowledge for software engineering research, practice, and training.

II. Related Work

Much of our knowledge of software engineering expertise come from studying *new* engineers rather than experienced ones. For example, the closest work to ours is Hewner and Guzdial's investigation of what employers in a small game company look for in new graduates [2]. The authors interviewed and surveyed over 30 engineers, managers, and artists about qualifications for recent graduates. The authors identified programming skills as well as people skills, like the 'ability to work with others and check your ego at the door'. In addition to biases for the gaming industry, the authors also suggested differences in expectations between new and senior hires. Begel and Simon's 2008 ICER paper performed a similar investigation [1], following 8 new hires at Microsoft for 4 weeks and examining their daily tasks. The authors found that novices need to identify 'tasks that have an impact', to be 'persistent' (avoid lack of self-efficacy), and to collaborate effectively in a 'large-scale software team setting'. However, it was unclear whether experienced engineers had similar issues.

Some works are prescriptive, offering recommendations, but often providing few insights into why topics are (or are not) important. For example, Lethbridge [10] surveyed 168 software professionals about the relevance of computer science education topics from the ACM Computing Curricula [6]. A notable exception is Kelley's work examining star performers, including software engineers at HP and Bell Labs [11]. The authors prescribed nine working strategies and described how they lead to high productivity—blazing trails, knowing who knows, proactive self-management, getting the big picture, the right kind of followership, teamwork as joint ownership of a project, small-I leadership, street smarts, and show and tell.

Other works have considered related occupations such as "Information Technology" [8] and "Information Systems" [12]. Many of the needs, like 'supporting existing portfolio of applications' and 'analyze business problems and IS solutions' were directed towards selecting software rather than creating it.

Some insights into software engineering expertise have come from luminaries. At OOPSLA 2003 [13], Brechner—a director of development training at Microsoft—discussed the need for design analysis, embracing diversity (e.g. other nationalities), multidisciplinary project teaming, large-scale development, and quality code. Dijkstra, in his Turing Award speech [14], argued that good developers create obvious and elegant solutions, constructed with provable correctness. These attributes are likely important, but the luminaries were probably not aiming to exhaustively or rigorously identify key attributes.

Popular press and best-practice guides have also considered the topic. In a New York Times' interview [15], Bock—Google's vice president of people operations—indicated that a software engineer's ability to learn on the job was critical, also claiming that human judgment, inspiration, and creativity were more important than technical knowledge. Similarly, McConnell [16] argued that effective developers, in addition to technical skills, had various personality traits like being humble about their intelligence, curiosity, and intellectual honesty.

Comparisons of novices and experts also reveal insight into software engineering expertise, showing that experts are more productive, systematic, and well-prepared [3][17][18]. Sackman et al., in one of the first comparisons of developer productivity in 1968 [5], found that completion times of programming and debugging tasks can vary as much as 28:1 between the best and worst engineers. Researchers also suggest qualitative and environmental differences. Robillard et al. [19] found that effective developers were more methodical and better at recognizing relevant information. Ericsson et al. [20]—origin of the meme that 10,000 hours of deliberate practice is needed to achieve expertise—found that attaining expertise required time, materials, teachers, and facilities.

Research into various aspects of teamwork suggests other important attributes. Simon's research into effective organizations [21] argued that setting, communicating, and alignment of goals within teams are important. Gobeli et al. [22] found that effective conflict management (e.g. confronting and give and take) are important for successful projects. Research on collaborations [23][24][25][26][27][28] suggests that expert software engineers have knowledge of code ownership, the technical domain, and argumentation skills.

While related research is extensive, few works directly address software engineering expertise. Those that do, focus on a narrow subset of factors. In our work, we give greater breadth, depth, and rigor to our understanding of software engineering expertise than the current literature offers.

III. Method

Ideally, an empirical study of software engineering expertise would sample a wide-range of software companies, software products, and company cultures. As an initial effort, we tried to approximate the ideal by interviewing experienced engineers at Microsoft, a large company with a diverse set of software products and engineers. We chose face-to-face semi-structured interviews to identify an exhaustive list of attributes with detailed and contextualized understanding of their meaning and importance.

A key decision in our method was determining whose subjective opinions of software engineering expertise could be considered credible. Licensure and accreditation of engineers is still uncommon. The ACM's definition of software engineers as 'people who produce software for earnest use' [6] is vague. We therefore used the approach utilized by researchers of human expertise [20], basing our definition of expertise on people having achieved some degree of recognition as software engineering experts. We selected engineers at or above the Software Development Engineer Level 2 (SDEII) title. These engineers were confirmed as experts by other engineers via the hiring or promotion processes.

Based on prior work, we aimed to obtain a stratified random sample of engineers across two important dimensions: *product type* (10 major divisions at Microsoft plus one for all others including Skype, Data Center Ops, and Distribution) and *experience level* ('experienced'—titles at or above SDEII—and 'very experienced'—titles at or above Senior Dev Manager—typically with 15+ years of experience). We used the corporate address book, which the 1st author had access to as a full time Microsoft employee. We randomly sampled engineers in the 22 strata in a round-robin fashion with 3 employees each round, aiming for at least 2 informants in each stratum. Of the 152 engineers we contacted, we interviewed 59 (39%), see Table 1.

The interviews were semi-structured and about 1 hour in duration. We started by describing our study, explaining how we located the interviewee, asking permission to record the interview, informing them that all personally identifiable

Table 1. Stratified random sample of experienced engineers at Microsoft

Experience Level \ Product Type	Ad Platform	Bing	Corp Dev	Dynamics	Office	Phone	Server & Tools	Windows	Windows Services	Xbox	Other	Totals
Experienced titles: SDE II, Senior SDE, Senior Dev Lead	2	2	2	2	2	3	3	6	3	2	2	29
Very Experienced titles: Architect, Technical Fellow, Partner Dev Manager, Partner Dev Lead, Principal Dev Lead, Senior Dev Manager, or Principal SDE	3	3	3	2	3	2	2	5	2	3	2	30
Totals	5	5	5	4	5	5	5	11	5	5	4	59

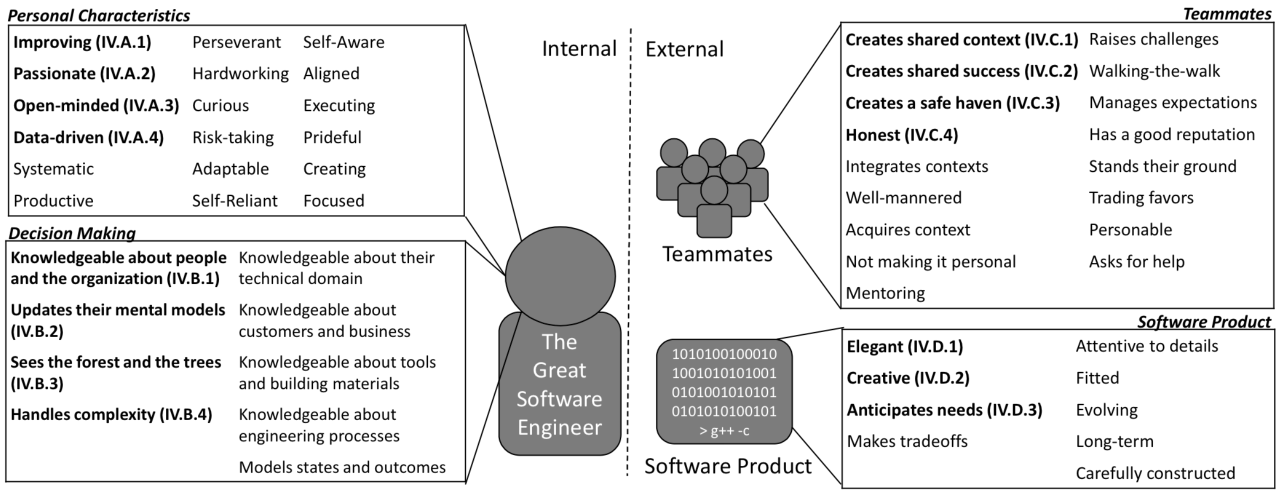

Fig. 1. Model of attributes of great software engineers, with attributes we discuss in detailed in bold.

information will be removed, and detailing their rights to refuse to answer any question and to have their responses removed later. We began the interview by asking: "I want to start by learning a bit more about you. What software products, at Microsoft and elsewhere, have you worked on?" This helped us to establish rapport and facilitated informants' reflections; this prior history was later removed during transcriptions to preserve anonymity. We then asked: "Think back to someone you've worked with that you thought was a great software engineer. What were some attributes that made the person 'great' in your mind?" We asked follow-up and clarification questions for attributes that we thought were interesting (e.g. novel, vague, or counter to prior informants).

In the second part of the interview, we asked about attributes that either lacked clarity or (we thought) might vary in interpretation. As we learned more about the attributes from interviewees, we updated the set of attributes we inquired about (once every ~10 interviews). For time considerations, we limited our discussions to 5 attributes of interest. We closed the interview by restating the purpose of the research and asking interviewees whether they had anything else to add.

To analyze the more than 60 hours of interviews and 388,000 words of transcripts, we used a grounded theory approach [29]. We began with open coding, identifying and assessing all excerpts that discussed attributes of great software engineers. Once we developed our initial attributes, descriptions, and groupings, we made a selective coding pass through our data—consolidating the attribute set. To validate our interpretations, we then solicited the help of a Senior Software Development Engineer (3rd author) to analyze roughly 1/3 of the interviews, developing her own attributes, definitions, and groupings, and then consolidating with the initial set. We made a final pass through all transcripts to produce the final set of attributes.

IV. Findings

Our analysis identified a diverse set of 53 attributes of great software engineers. At a high level, our informants described great engineers as people who are passionate about their jobs and are continuously improving; who develop and maintain practical decision-making models based on theory and experience; who grow their capability to produce software that are elegant, creative, and anticipate needs; who evaluate tradeoffs at multiple levels of abstraction, from low-level technical details to big-picture strategies; and whom teammates trust and enjoy working with.

To give readers a sense of how the attributes interconnect, we present a model of the 53 attributes in Fig 1. We organize the attributes into internal attributes of the engineer's personality and ability to make effective decisions, as well as external attributes of the impact that great engineers have on people and product. Making effective decisions involved recognizing situations as well as knowing alternative courses of action, likely outcomes, and values of outcomes. The external attributes focused on great engineers applying their emotional intelligence and decision-making models to their software, their teammates, and the potentially millions of users and stakeholders they serve via their software engineering efforts.

Many of the attributes are applicable to many professions, and some, to simply being a good person. Our objective was to identify, among all possible attributes, the set that expert software engineers viewed as important for the engineering of software. More importantly, we aimed to provide a contextualized understanding of why these attributes are important in real-world practice.

In the rest of this section, we provide a description of each attribute and quotes from informants (including their title and division when this information would not reveal their identity) that capture the sentiment in interviews. Due to space limitations, we focus detailed descriptions on attributes that we felt—based on prior work—were particularly interesting.

A. Personal Characteristics

Informants mentioned 18 attributes of engineers' personalities (see Table 2). With attributes like passionate and curious, these concerned who great engineers were as people. For many attributes, informants felt that the attributes were *intrinsic* to the engineer—formed through their upbringing—and were difficult (if not impossible) to change.

TABLE 2. PERSONAL CHARACTERISTICS OF GREAT SOFTWARE ENGINEERS. ATTRIBUTES DISCUSSED IN DETAIL ARE IN BOLD.

Attribute and description	*Excerpt that capture interviewees' sentiment*
Improving—not satisfied with the status quo: constantly looking to improve themselves, their product, and/or their surroundings.	***"…Always looking to do something better, always looking for the next thing, studying about the newer thing… to do things better."*** **-Senior Dev Lead, Xbox**
Passionate—intrinsically interested in the area they are working in (i.e. not just in it for extrinsic rewards like a pay check).	***"You can't be a great engineer and not enjoy what you're doing… 9 to 5 wouldn't make you a great engineer…not just to get a paycheck."*** **-Principal SDE, Xbox**
Open-minded—willing to judiciously let new information change how they think.	***"The problem is... not being willing to take the input of others…not invented here, that's a huge problem."*** **-Principal Dev Lead, Office**
Data-driven—taking and evaluating measurements of their actions and of the software, often relative to expectations.	***"The difference between fact and hypothesis… How can I prove that? A new fact might show up, that this proves what I thought was my theory."*** **-Principal Dev Lead, Xbox**
Systematic—taking actions in logical and ordered steps	*"…Have to be patient and not rush to the solution... go through a mental gymnastics in order to get to a solution."* -Principal SDE Lead, Windows
Productive—achieving the same results as others faster, or taking the same amount of time as others but doing more.	*"He codes quickly and fast… Just write the code and figure out how to get it working."* -Principal Dev Manager, Bing
Perseverant—not discouraged by setbacks and failures.	*"I will try to find out a solution. Those people always succeed …There is always a way."* -Senior Dev Lead, Dynamics
Hardworking—working more than is expected to finish deliverables and/or to accomplish their improvement goals.	*"Sometimes… that's just arduous. You really just need to grind through"* -SDE2, Server & Tools
Curious—wanting to know how and why things happen (i.e. how the code and the conditions produce a software behavior or customer reaction).	*"I was always asking why. Why does that thing work? Why does it do this? What is it? For me, I kind of had to have a need to know what made something tick and it's that curiosity…"* -Technical Fellow, division removed to preserve anonymity
Risk-taking—willing to go into high-value areas even though they may not have knowledge or expertise (e.g. new technologies).	*"They're willing to take on the challenge. So that's the most important one."* -Senior Dev Lead, Bing
Adaptable—adapting to changes in their environment, including changes in what they do (e.g. the software product) and how they do it (e.g. people, processes, and tools).	*"Things are going to change, what are you going to do about that?… move forward… adapt to work with what you have to work with?"*-SDE2, Service Engineering
Self-reliant—getting things done independently (i.e. not always going to their manager for help); removing roadblocks by leveraging their abilities and resources (e.g. asking experts for help).	*"Rather than looking around for somebody to solve it for them... try to figure out how they can do this on their own… get yourself unblocked attitude works really well."* -Principal SDE, Windows
Self-aware—continuously assessing one's situation and taking corrective actions when necessary.	*"A little bit of an intuition… being able to recognize this ain't working, I better start over."* -Principal Dev Lead, Xbox
Aligned—acting for the good of the product and the organization, not for one's own self-interest.	*"A mismatch of value… their number one goal is really to learn… you are paid because we are a business."*-Principal Dev Manager, Windows Services
Executing—knowing when to execute; no analysis paralysis	*"They should not get into analysis paralysis… most optimal solution for the problem on hand, not the most accurate solution."* -SDE2, Phone
Prideful—taking pride in oneself and ones' product; letting their output be a reflection of their skills and trying their best to deliver.	*"Really being able to demonstrate something that you've done, that you're really proud of … it's quality work."*-Principal Dev Lead, Xbox
Creating—wanting to bring ideas and thoughts into reality (e.g. a software product or a feature).	*"They feel more accomplished at the end of the day if they've actually built something… designed something, maybe they wrote some code."* -Senior SDE, Windows
Focused—allocating and prioritizing their time for the most impactful work.	*"In an environment like Microsoft where there's a lot of meeting and interruptions… get focus and when to get their focus."* -Principal Dev Lead, Xbox

1) Improving

Informants described great engineers as improving: not satisfied with the status quo and constantly looking to improve themselves, their product, and/or their surroundings. Informants believed that engineers did not start their careers being great, but that young engineers needed to learn and improve. Informants also felt that because the software field was rapidly changing and evolving, unless engineers kept learning, they would not become and would not continue to be great software engineers. This notion of running up an infinite escalator was prevalent among informants:

"Computer technology, compared to other sciences or technology, it's pretty young. Every year there's some new technology, new ideas. If you are only satisfied with things you already learned, then you probably find out in a few years, you're out of date… good software engineer [sic], he keep investigate, investment. [sic]" -SDE2, Corp Dev

2) Passionate

Informants described great engineers as passionate: intrinsically interested in the area they are working in, and not just for extrinsic rewards such as money. Informants felt that software engineering required a tight fit between a person's passion and the project to achieve high quality:

"I think that there are people who are great software engineers who are in the wrong place and aren't motivated and they end up not performing well." - Principal Dev Lead, Dynamics

There was also a sentiment that no matter the subject matter, there will be someone with a natural affinity towards it:

"I found that there's always a person who's passionate about every type of thing, you just have to find the right people… I ended up in the wrong job for six months. It was painful. People around me, they loved their work" - Principal Dev Lead, Phone

3) Open-minded

Informants described great engineers as open-minded: willing to judiciously let new information change how they think, not taking the current understanding as gospel. Informants felt that outcomes in software engineering (e.g. user reactions and commercial success) were difficult to predict:

"You should be open… what you think need not be the right thing tomorrow… like the Facebook explosion, when Myspace was already there, but it exploded… no one knew that Facebook would explode when it started". - Senior SDE, Windows Services

TABLE 3. DECISION MAKING OF GREAT SOFTWARE ENGINEERS. ATTRIBUTES DISCUSSED IN DETAIL ARE IN BOLD.

Attribute and description	*Excerpt that capture interviewees' sentiment*
Knowledgeable about people and the organization—informed about the people around them: their responsibilities (i.e. organizational structure), their knowledge, and their tendencies	***"Companies like Microsoft, there's literally people here who have created a world, the technological world that we live in today. They're stars in that regard… tap into this wealth of knowledge that Microsoft brings to the table, the talent pool that's here."* -SDE2, Xbox**
Sees the forest and the trees—considering situations at multiple levels, including technical details, industry trends, company vision, and customer/business needs	***"…Both a very, very narrow extremely technical prospective on his code, but also know where it fits in with the bigger picture, and to be aware of how it affects even our major external customers, and the company vision."* -Principal SDE, Windows**
Updates their mental models—keeping up to-to-date their mental models through evaluating changes in their context	***"Unlearning… two thirds or three quarters of what you know is still valuable, quarter to a third is the wrong thing in this world and so the trick is to figure out which is which really quickly."* -Technical Fellow, division removed to preserve anonymity**
Handles complexity—able to grasp and reason about complex and intertwining ideas	***"Frighteningly intelligent and smart, and they just walk around with this picture in their head all the time of how everything fits together."* -Principal SDE, Windows**
Knowledgeable about their technical domain—thoroughly conversant about their software product, including knowledge about the domain and competitors	*"You should have a very good understanding of the entire system as well as all of the moving parts… the architects behind big systems, complex systems, and know it, all the gotchas, in and out."*-Senior SDE, Windows Services
Knowledgeable about customers and business—conversant about the role their software product plays in the lives of their customer and the business proposition that entails	*"Really understanding the point like who is the customer, why are we doing this."* -Principal Dev Lead, Xbox
Knowledgeable about tools and building materials—versed in strengths and limitations of the tools and building materials used to construct their software product	*"The fundamentals, you've got to learn your ... data structure, algorithm stuff inside out… because everything else is building on them."* -Partner Dev Manager, Corp Dev
Knowledgeable about engineering processes—skilled in best practices for building the product: their purpose, how to do them effectively, and their cost in time and effort	*"Having good practices around, how you do the code reviews and check ins and having unit tests that enforces things don't break and that kind of thing it is way, way more important than the actual having a beautiful architecture."* -Principal Dev Lead , Windows Services
Models states and outcomes—building mental models linking the current state, alternative actions, probabilistic outcomes, and value of outcomes	*"I think that mostly just comes from experience… learning where the hard parts of the problems are probably lurking and what trouble they might cause you or something like that… having a good pattern of recognition"* -Principal Dev Lead, Corp Dev

Informants also felt that because software products were large, complex, and constantly changing; it was rare for anyone to have a complete understanding. Therefore, even great engineers needed to be open to changing their understanding:

"No matter how much you know, the software industry is so large… there's so many other areas… If that person has something to say that hadn't occurred to me, I'll stop everything and say, ok, explain this. What did you see, that I didn't see?" -Senior SDE, Office

4) Data-driven

Several informants described great engineers as data-driven: taking and evaluating measurements of their actions and of the product, creating behavioral feedback loops for optimizing software and processes. Informants believed that decisions, when possible, should be made using data, not intuition or arguments. Many viewed this approach as a way to avoid confirmation bias, but lamented that it was no panacea:

"One thing that surprises me… even though we are driven by data, at least we try to believe we are… Some data gets shown to us. We figure out some ways to ignore it. So, maybe, maybe everybody thinks that they're data driven, but I've seen people come up with excuses for why the data doesn't apply to them. I've seen that a million times." -Senior SDE, Office

B. Decision Making

Informants mentioned 9 attributes of engineers' ability to *decide* (see Table 3): synthesizing the current context, decision alternatives, probabilistic outcomes, and values of outcomes. Informants felt strongly that having *book* knowledge was not sufficient; great engineers understood how decisions play-out in complicated real-world conditions. Great engineers not only knew what *should* happen, but also what *can and likely will* happen.

To make effective decisions, our informants discussed engineers needing knowledge about context along several dimensions—technical domain, customers and business, tools and building materials, and engineering practices—as foundational to engineering decisions. We detail one area of knowledge, people and organizations, because it had insights into interpersonal dynamics not often discussed in the literature.

We also discuss engineers updating their mental models, seeing the forest and the trees, and handling complexity. Informants' insights into these attributes revealed that great engineers have complex and multi-faceted decision-making models that were continuously being updated. This reflected the complex decisions that great engineers often had to make.

1) Knowledgeable about people and the organization

Informants described great engineers as knowledgeable about people and the organization. This included being informed about their coworkers' responsibilities, knowledge, and tendencies. For example, knowing ownership enabled great engineers to determine key stakeholders for decisions and to communicate with the right people to align their work. This alignment commonly meant their management chain, but informants also discussed aligning with key partner teams (e.g. other parts of a product offering):

"Make sure that you are aware of that big picture, you know where you fit in and how you interact with everyone else to optimize what you are doing." - Principal Dev Lead, Ad Platform

Knowing who had expertise enabled great engineers to find the right people for help—often domain experts—and for great engineers in leadership positions, to take corrective actions to address knowledge gaps (e.g. assigning a more senior person):

"[This great engineer] would go through his organization and looked very carefully at the tasks that were being assigned and whether people had the right level of training and understanding and if they didn't, who their supervisor and whether that person did and would demand code reviews..."- Software Architect (division removed to preserve anonymity)

Knowing people's tendencies enabled great engineers to adapt their engagement techniques to obtain desired outcomes:

"You have to understand people so that you can influence or impact them... You have to do that both down and up and out." -Principal Dev Lead, Phone

2) Sees the Forest and the Trees

Informants described great engineers as seeing the forest and the trees, considering situations at multiple levels of abstraction, including technical details, industry trends, company vision, and customer/business needs. Informants indicated that mental models could exist at various levels and that great engineers reasoned at all levels quickly and accurately:

"What differentiated [this great engineer] from other people in management positions… capability to zoom into the details, and he was not just a high level guy, ...know the reality of the stack or the reality of the software…" -Senior Dev Lead, Ad Platform

Informants felt that this ability enabled great engineers to make globally optimal decisions, avoiding local optimizations:

"The challenge is having the ability to look at things from many different perspectives, at many different levels of abstraction or detail. Then, being able to choose how to lay things out… make a set of choices." -Technical Fellow (division removed to preserve anonymity)

3) Update Their Mental Models

Informants described great engineers as continuously updating their mental models at all levels of abstraction—ranging from technical details to industry trends—by explicitly evaluating changes in their context. Related to being open-minded, this attribute concerned the process of updating mental models, sometimes discarding existing models for new ones:

"You can always follow patterns too much... It's worked in the past, but conditions have changed. You always need to look and take a little bit of risk with each one of your tasks. If you're not then you're not really going to find out what's possible." -Principal Dev Lead, Office

Important contextual changes discussed by informants were commonly shifts in long-held understandings in software. Informants felt that these foundational shifts and their implications were critical for great engineers to understand and adapt to:

"Sometimes what used to be a second or third order effect comes to dominate. So way back in the day, if you wanted to performance optimize something you counted instructions. Processors got faster and faster, but memory references didn't. There became a day when it made more sense to count memory references than it did to count instructions. Unless you're conscious of when those things will intersect, you'll be on the wrong side of history and be frustrated." -Technical Fellow (division removed to preserve anonymity)

4) Handling Complexity

Many informants described great engineers as handling complexity with ease, grasping and reasoning about complex and intertwining ideas with agility. Informants felt that some software problems were inherently complex. This might have been especially salient at Microsoft, where products commonly build on top of multiple layers of technologies and interact with many other components. Building an accurate mental model of dependencies and connections was seen as critical:

"To solve the problem, [great engineers] have to have the ability to connect things… You are always debugging layers of stacks of code… this layer talks to some other layer in the horizontal... you need to solve the problem and you don't know what's going on." -Senior SDE, Windows Services

Some informants felt that the ability to handle complexity was a natural ability. Others felt that great engineers could effectively augment their natural abilities using tools and processes (e.g. externalizing knowledge by writing it down):

"Ability to capture… simulate the architecture in their head… there's probably a little bit of innate skill and cognitive ability… That said, the fact that you don't have that skill doesn't mean that there's no other ways of doing it that may be more brute force… writing things down and studying very carefully the architecture you've put down is putting the brute force time into studying a problem." -Partner Dev Lead, Windows

C. Teammates

Informants mentioned 17 attributes of engineers' interactions with teammates (see Table 4). Informants felt that great engineers were expected to positively impact teammates. For many informants, this was an important part of their job as leads or managers.

Attributes in this area revolved around four concepts: being a reasonable person, being a good leader, communicating effectively, and building trust. We discuss, in detail, attributes related to communicating effectively and building trust, as these concepts are frequently mentioned in the literature but often with little contextual understanding.

1) Creates Shared Context

Informants felt that creating shared context, which involved molding another person's understanding of a situation, was the most important aspect of "communicating effectively". This involved tailoring the message to another person's perspective:

"You perceive who you are talking to, and you are able to judge on those levels that they are, or you just ask important questions. Do you know about this? And then, be able to simplify the problem to the level that they're working in, or you estimate the amount of information given to them." -Senior SDE, Windows

This sentiment is closely related to the concept of "grounding" proposed by Clark and Brennan, which, when done successfully, required parties to "coordinate the content and the process" of communication [30]. Since the engineering of software involves many people, getting everyone to have a shared understanding was seen as essential to success:

"One person can only accomplish so much so you've always got to be working as part of the bigger group. People who can't communicate are only going to be sort of so-so effective…" -Principal Dev Lead, Corp Dev

Informants also stated that great engineers, especially ones at higher-levels, often had to communicate with people that do not have a complete (or the same) understanding of the situation but are critical to success (e.g. partner teams, customers, or management). Therefore, crafting the message such that others can comprehend the situation was important:

"Our areas where the things are inherently difficult to talk about… business partners or with a customer… When you go outside and you talk to customers, they think about things in much different terms and so in some ways you have to kind of switch gears… why you should care about it and here is how you should think about it." -Principal Dev Lead, Corp Dev

2) Creates Shared Success

Informants described great engineers as creating shared success for everyone involved, possibly involving personal compromises. Informants felt that software engineering was a collaborative process, requiring many people, often with different personal motivations and organizational objectives. Great engineers needed to get everyone making decisions aligned to a shared goal:

TABLE 4. GREAT SOFTWARE ENGINEERS' ENGAGEMENT WITH TEAMMATES. ATTRIBUTES DISCUSSED IN DETAIL ARE IN BOLD.

Attribute and description	*Excerpt that capture interviewees' sentiment*
Creates shared context—molding another person's understanding of the situation while tailoring the message to be relevant and comprehensible to the other person.	***"Most compellingly relate the value of that abstraction as it goes to non-abstract to very abstract to each person… empathize with your audience... get them to get it."* -SDE2, Windows**
Creates shared success—enabling success for everyone involved, possibly involving personal compromises.	***"Find the common good in a solution… express here's the value for you... It's a win-win situation."* -Senior Dev Lead, Windows**
Creates a safe haven—creating a safe setting where engineers can learn and improve from mistakes and situations without negative consequences.	***"If you learn something from a failure, that's a wonderful sort of thing… [but not] If you're afraid of getting smacked upside the head… encourage the people to experiment, possibly succeed, possibly fail*". -Senior SDE, Office**
Honest—truthful (i.e. no sugar coating or spinning the situation for their own benefit).	***"When you do make mistakes, you've got admit you made a mistake. If you try to cover up or kind of downplayed mistake, everybody will see it, it's super obvious. It affects your effectiveness."* - Partner Dev Manager, Corp Dev**
Integrates contexts—integrating different contexts together into their own understanding, including noticing and asking questions about gaps and incongruities.	*"Disparate ideas and pieces of information… put pieces together… asking good questions... organize your thoughts that will help you make those connections."* -Principal Dev Lead, Dynamics
Well-mannered—treating others with respect, not obnoxious about their title, accolades, or knowledge.	*"Smart but not cocky… He's the one who knows all the information. He never comes across that way… [does not] make the other people seem like, 'Oh, I feel so stupid.' "* -Senior SDE, Windows Services
Acquires context—effectively acquiring contexts and knowledge from others.	*"To get the software to work… each things need to be integrate together [sic]... learn from others and you need to know the things others are working on."* -SDE2, Corp Dev
Not making it personal—divorcing oneself from personal feelings and biases.	*"You can have a very open and heated discussions. But it is all very professional; none of this is ever taken personally."* -Principal Dev Lead, Server & Tools
Mentoring—instilling knowledge to others; helping others improve.	*"He's seen stuff that you haven't seen yet, and he's willing to share his knowledge… Let's spread some of that good knowledge around."* -Senior SDE, Office
Raises challenges—pushing others to action, expanding the team's limits.	*"…Shared confidence: so it's like he's done it and so you can do it… spark your imagination and your sense of self confidence for you to boot strap yourself up."* -SDE2, Windows
Walking-the-walk—acting as the exemplar (e.g. using good practices) for others to follow.	*"I would like to model myself against that behavior (of a great software engineer)… it inspires me to do the same thing."* -Senior Dev Lead, Ad Platform
Manages expectations—setting clear expectations, updating them, and then delivering on them.	*"Your leads, your managers … setting expectations, they know what you're going to do, you do it."* - SDE2, Servers and Tools
Has a good reputation—having the belief, respect, and confidence of others to make good decisions.	*"Build up that reputation and that trust through your years… worth of good deeds essentially, so that when you make that recommendation, they go, I am going to listen to him."* -Principal Dev Manager, Windows Services
Stands their ground—firm against outside pressure (e.g. management), when appropriate, based on sound principles	*"He will say no, if he has to. If what they're asking him to do jeopardizes something else… stand up and be brave about it."* - Principle SDE, Windows
Trading favors—creating personal equity with others.	*"Returning a favor here and there… above and beyond to help somebody else out and then somewhere down the road that person has that extra good will to come help you out.*" -Senior Dev Lead, Windows
Personable—cool people that one would engage with in a non-work setting.	*"One of the characteristics I look for in every person that I get… Can I have a beer with this guy? …but they're very, very stubborn and you know that you can only put them on one thing and that's it."* - SDE2, Servers and Tools
Asks for help—finding and engaging others with needed knowledge and information.	*"He does his homework and anything that he doesn't know… he goes and finds a person that does know. He doesn't try to know it all himself."*-Principal SDE, Windows

"No matter how good is our code, if our partner [sic] cannot give it a good product for us then we cannot share our greatness to the whole world. A lot of time I see our support to our client is not very well [sic]… we should have a good result combined together." -Senior SDE, Phone

Many informants said that great engineers made shared success bidirectional between managers and individual contributors. Managers needed to put engineers in positions to succeed; great engineers needed to engage management to facilitate mutual success. Great engineers often had better understandings of the details; managers often had a broader perspective of the situation:

"It's a two-way communication… there's something going to happen down the road, this piece of code or this feature going to have some issues, need to make your manager aware." -SDE2, Phone

This attribute likely helped to avoid dysfunctional 'time famine' situations as discussed by Perlow [31], where crises arise in teams due to a lack of shared understanding about status and objectives.

3) Creates A Safe Haven

Many informants described great engineers as creating a safe haven where other engineers can learn and improve from mistakes and situations without negative consequences. Usually associated with leaders, informants felt that if engineers are afraid of mistakes, then their growth would slow:

"Chasing after a career path or something… you will deliver your best performance if you are not insecure… One of the challenges as a manager people face these days is retaining talent because there is so much attrition all over." -Senior Dev Lead, Ad Platform

Informants also saw the *lack* of this attribute as a major contributing factor for talent loss. Informants did not want to work in environments where they felt insecure, and often avoided those teams/organizations:

"If you make one mistake or don't know something and you're sort of dinged by that… and you're only judged if you say everything's perfect even if it isn't… Then you start to have this really kind of I think dysfunctional environment set up where everybody just doesn't say the truth." - Principal Dev Manager, Windows Services

TABLE 5: GREAT SOFTWARE ENGINEERS' SOFTWARE AND DESIGNS. ATTRIBUTES DISCUSSED IN DETAIL ARE IN BOLD.

Attribute and description	*Excerpt that capture interviewees' sentiment*
Elegant—simple and intuitive (i.e. not complex) software/designs that others can understand.	***"They are simple... very easy to understand in a sense that it's very simple. Doing something well and in a very simple way is very very hard."*** **-Principal Dev Lead, Bing**
Creative—novel solutions based on understanding of the context, existing solutions, and the limitations of existing solutions.	***"Think outside the box… here's a traditional solution… often have constraints...take the difficult circumstance and actually make it into something that could actually still work, but without a huge complex overhead."*** **-Senior SDE, Principal Dev Lead**
Anticipates needs—producing software that accommodate likely needs and problems based on contextual knowledge	***"Think where you're going to get into trouble potentially… What are we ultimately trying to do, and what can I do today that will save me time over the lifetime of the system?"*** **-Technical Fellow, division removed to preserve anonymity**
Makes tradeoffs—making trade-offs (e.g. quality for time to market) based on the context and the situation.	*"Weight the tradeoffs. Is this really the right thing to do?"* - Principal SDE Lead, Windows
Attentive to details—paying attention to coding details during development including error handling, memory consumption, performance, and style.	*"The quality of the code, performance, space, and how many bugs it has, how robust it is, and how it handles exceptions will have great differences."* -SDE2, Severs & Tools
Fitted—thought-through designs that take the context (e.g. other components) into consideration.	*"You understand better, interactions around you or around your code. How your code is supposed to work… if I tweaked this here I'm not going to break something else."*-Senior Dev Lead, Xbox
Evolving—structure the software to be efficiently delivered iteratively or in pieces.	*"A very clean step-wise process moving forward… how can we break this down so that we have really concrete deliverables on an ongoing basis."* -Senior Dev Lead, Bing
Long-termed—acting with an eye towards the future, not just short term gratification.	*"A bunch of isolated, fragmented, short-term solutions together, what do you get? Not something great. Definitely, someone needs to have this long-term vision and say: we make decisions not based on the immediate problem…"* -Partner Dev Manager, Corp Dev
Carefully constructed—using the right processes (e.g. unit testing) to produce the software.	*"Unit testing, of the code. Well before that was fashionable… almost no bugs ever found in the product and that was actually his track record."* -Senior Dev Manager, Windows

Though informants felt that safe havens were important, many expressed the need to balance a safe environment with feeling the pain of mistakes. The reasoning was that pain from mistakes was the best teacher: if an engineer was hurt by something, then the engineer quickly learned to avoid it:

"I believe in having people feel the pain of their own mistakes… dealing with the ramifications of the decisions that are being made, I guess is the best way to learn." -Principal Dev Lead, Office

4) Honest

Informants felt that being honest was the most important attribute related to 'trust'. This was about great engineers providing credible information. Engineers that presented a version of the situation that suited their own benefits were viewed negatively. Informants felt that they needed to be able to take action based on information that an engineer provided:

"Influence comes to someone else trusting you, part of that trust is that they go, 'You know what? I know that this person always speaks the truth.' As a result of that, when they say something is good, I will totally believe them because they are not trying to kind of misrepresent something or make them look better or whatever." -Windows Services Principal Dev Manager

Informants also did not appreciate wasting time shifting the blame for problems. They felt that great engineers focused their attention and efforts on addressing the problem:

"Rather than thinking about how to actually fix the problem at hand, [other engineers were] more like "How do I make sure that nobody will come back and think that maybe that happened because of something that I might have done?" [This great engineer] has a way of kind of saying: It doesn't matter…What matters is right now. How do we actually work through it?" -Senior SDE, Windows

D. Software Product

Informants mentioned 9 attributes regarding the software that great engineers produced (see Table 5). Like artists appreciating masterpieces of other artists, our informants, many of whom are great engineers themselves, saw beauty in the software produced by other great engineers.

1) Elegant

Informants described software of great engineers as elegant: possessing simple and intuitive designs that another person (or themselves later) could easily understand. Among all software-related attributes, elegance was the most revered. Informants recognized that some problems in software were complex and highly constrained, making it difficult to have a simple solution that met the requirements:

"The style… always, an idea, and it was all clean… very concise. Just looking at it, you can say, "Okay, this guy, he knew what he was doing."... There's no extra stuff. Everything is minimally necessary and sufficient as it should be. It's well thought-out off screen." -Windows, Senior SDE

Informants also felt that it was critical to avoid complexity. Complex solutions increased the likelihood of bugs and increased maintenance costs (if problems were fixable at all). Evolving the software was also more costly when the design was brittle to change:

"Never complicate any things… when you simplify things it becomes easier for you to maintain, going forward for customers… You get lesser number of issues reported by a customer." -Senior Dev Lead, Dynamics

2) Creative

Many informants described the software of great engineers as creative, involving novel solutions based on understanding of the constraints of the context, existing solutions, and the limitations of existing solutions. Informants felt that there were two important parts to having creative solutions. First, great engineers understood constraints and requirements of the particular context/problem:

"If you're looking for really an innovative …or just a solution that's outside the current norm… think through the problem…constraints that are currently imposed on the environment." -Principal SDE Lead, Windows

Second, great engineers knew of and knew when to apply existing solutions (and *not* be creative). This was important because informants felt that known solutions were generally preferable since they were less costly and less error-prone:

"You are now using all of your creativity to reinvent things that are already invented and that is just basically wasteful."-Principal Dev Manager, Windows Services

Still, most informants felt that novel problems occurred frequently in software engineering, needing great engineers with the ability to come up with innovative solutions:

"Understanding patterns and understanding how to apply something is very important so you don't recreate wheels all the time… when there isn't an obvious pattern… Are you creative enough… come up with something new?" -Senior Dev Lead, Windows

3) Anticipates Needs

Many informants described software of great engineers as <u>anticipating needs</u>, accommodating possible future requirements not known at the time when the software was initially produced. Informants commonly mentioned scale (e.g. more users), feasibility (e.g. technology advancing to the point where new things were possible), and integration (e.g. working together with additional software products):

"QQ, the Chinese chat program. It now has hundreds of millions of users. That system was designed fifteen years ago, when QQ only had a few million users. It still works today, that's amazing, to have a system that scales that well, to foresee all the issues it would have to face." -SDE2, Severs & Tools

More than any other attribute, informants discussed the propensity to overly anticipate needs in the face of uncertainty, incurring high costs to add unneeded flexibility. Some thought that any prediction of the future was foolish and preferred to design for current needs and being open to rewrites:

"'..Architect something now that's going to survive well 20 years from now?' Nobody is that smart to be able to predict the future that well, I will refactor towards new requirements and I constantly do that."-Senior SDE, Office

V. Threats to validity

As with any empirical study, our results are subject to various threats to validity. There are threats to the *construct* validity from the lack of a clear and shared definition of a 'software engineer'. Though, in general, informants understood that we meant people that wrote code to be used by customers, and we clarified whenever there was confusion. Our interview and analysis processes contain threats to *internal* validity. Informants could generally only mention a few salient attributes unprompted; given more time to think, informants may have produced more attributes. Moreover, while our analysis was systematic, other researchers may discern different attributes, definitions, or models than ours. Our sampling method also contain threats to *external* validity. Though our 59 interviews yielded rich insights, it was a small sample, even for Microsoft, which employs tens of thousands of engineers. This led to some natural biases, such as underrepresentation of women; we had only 3 among our 59 informants. In addition, we only sampled engineers in Seattle, USA. Findings may not generalize to other cultures, especially attributes associated with management. The size of the organization may also affect generalizability, especially for attributes related to people and organizations. Microsoft also had an established set of practices, tools, and products; findings may not generalize to other situations (e.g. start-ups). Microsoft is a software-centric company; informants discussed unfavorable conditions in non-software centric industries, like finance and retail. It is unclear whether the same attributes (or their standards) would generalize. Nonetheless, since Microsoft is a successful organization that produces software used by billions, findings are relevant and interesting.

VI. Discussion

Overall, nearly all of the attributes of great software engineers we uncovered have been mentioned to some degree in prior work, e.g. [1][3][27], and many attributes overlap with ones important to other professions, e.g. [11][21]. Our results, however, are the first time a comprehensive set of attributes for software engineering has been identified and described. In addition, there are several important implications. First, our results suggest that <u>productivity</u> is only one criterion for excellence. *How* the engineering is conducted, relative to management (e.g. <u>managing expectations</u>), subordinates (e.g. <u>creating a safe haven</u>), teammates (e.g. <u>asking for help</u>), partners (e.g. <u>creating shared success</u>), and even oneself (e.g. <u>perseverant</u>), are all critical. This reinforces the perspective that software engineering is a *socio*technical undertaking, and not just a technical one. Furthermore, simply delivering the code is also insufficient. With attributes like <u>elegant</u>, <u>creative</u>, <u>long-termed</u>, and <u>seeing the forest and the trees</u>, our results indicate that good software engineering requires engineers to make complex, experienced-driven, contextual considerations.

Second, though rarely discussed in the software engineering literature, results suggest that effective decision-making is critical. Informants felt that there are usually myriad options—not all good—for what to do and how to do it. And as engineers grow in their careers, they are tasked with making decisions in increasingly more complex and ambiguous situations, often with significant ramifications. Making effective decisions, entailing attributes in Section IV.B as well as other sections, is an important skill for engineers to develop.

Third, results suggest that being able to *learn* new technical skills is likely more important than any individual technical skills. Informants, even ones in the same division, used diverse technologies—sometimes project specific tools (e.g. Cosmos, a Microsoft version of Hadoop). There was no consensus on any specific technical topic (e.g. architecture) as being essential. Rather, most informants stressed needing to learn new skills (i.e. <u>continuously improving</u>) and nearly all viewed it as a critical attribute of great engineers.

The attributes we have identified and described may have wide-ranging implications for researchers, novice engineers, managers, and educators. In the rest of this section, we discuss implications and opportunities to build upon our results.

A. For Researchers

Our findings raise important questions about our current understanding of *what affects* software engineering outcomes. Much of the prior work on this topic focuses on processes [32], cost-estimation (e.g. COCOMO [33]), coordination [34], and requirements engineering [35]. While many of the attributes we uncovered underlie these concerns, such as <u>creating shared context</u> and <u>carefully constructed</u>, there are also others that have not been considered, such as <u>aligned</u>, <u>creates a safe haven</u>, and <u>fits</u>. These likely reflect higher-level concerns such as individuality, organizations, and productization.

Our results also suggest several new directions for tools research. For example, we are not aware of any tools that help engineers be more <u>well-mannered</u> in emails or <u>evaluate tradeoffs</u> and <u>see the forest and the trees</u> when making decisions. Tools research may explore facilitating and training engineers, especially novices, in these attributes

An important part of better understanding these attributes is developing measurements that operationalize these attributes. These may enable rigorous science to better understand how the attributes vary and their effects on teams and outcomes. Such measurements may also form a critical foundation for managers to identify and cultivate talent, for novices to improve, and for educators to assess learning outcomes.

B. For Novice Engineers

New software engineers are often unsure of how to become great engineers [1]. Our findings enumerate a set of attributes that they might aspire to achieve. Improvements might come from trainings, projects at work, mentoring, or self-adjustments (e.g. for personality traits). Researchers might also investigate interventions that help achieve the attributes quickly and effectively.

Furthermore, novice engineers may also use our results to assess their fit with prospective employers. As mentioned in Section IV.A.2 on <u>passionate</u>, the fit of the engineer with the project is critical. Novice engineers might assess their fit, in terms of the attributes they value, with a prospective team.

Our findings may also help novices better present themselves to employers. Since experienced engineers and managers value these attributes, novice engineers might consider demonstrating to employers that they have or can develop these attributes. This also extends to highlighting the qualities when authoring their resumes or presenting themselves in interviews.

C. For Managers

Our informants discussed many attributes that were important for engineers in senior and leadership positions, such as <u>mentoring</u>, <u>raising challenges</u>, and <u>walking-the-walk</u>. New research may explore ways to help engineers improve these attributes to become better managers.

Beyond improving themselves, our results may also help managers make more effective hiring decisions. Managers may better identify candidates that fit the culture and context of the team. They may also better *avoid* engineers *without* the attributes, such as not <u>aligned</u> (off doing their own projects), not <u>well-mannered</u> (being an 'ass', as many engineers described it), or not <u>asking for help</u>.

Finally, our results strongly suggest that managers may consider *cultivating* the attributes within their teams. Managers may consider using the findings—with help from further research—to build a culture that is conducive to attracting, producing, and retaining great engineers.

D. For Educators

Our findings also raise significant questions about curriculum choices, teaching methods, and learning objectives in formal computer science and software engineering education. Educators may consider adding courses on topics not found in their current curricula. For example, we found decision making to be a key part of software engineering, but this specific topic is not a part of the ACM's Computing Curricula [6]. A course specifically about decision making (e.g. discussing Simon's model of rational choice [36], Klein's naturalistic decision-making approach [37], or case studies of software engineering decisions), might be valuable to students.

Software engineering educators might also use our results to examine their teaching methods. Most attributes of great engineers focus on *how* rather than *what*, whereas most instructions in software engineering focus on teaching skills and knowledge (the *what*), such as prior work on tools for automated testing and analysis. Educators might consider improving *how* software engineering goals are attained. For example, existing project-based courses might use attributes presented in this paper to help student evaluate each other's behavior, as well as grading non-functional attributes of the code, such as <u>elegance</u>, <u>anticipates needs</u>, and <u>creative</u>.

Finally, educators might consider explicitly discussing what students will *not* learn in school, allowing them to be aware of potential knowledge gaps and empower them to seek out opportunities outside of the academic setting (e.g. internships or open-source projects). For example, attributes like <u>self-reliant</u> may not be reasonable to teach in an academic setting and might be better learned through mentorship on the job; nevertheless, educators should consider informing students that it is a critical component of software engineering expertise.

VII. Future Work

Though this paper is a good start at better understanding software engineering expertise, there are countless opportunities for future work. The combinations of attributes that are interesting or prevalent could be examined. Each specific attribute we uncovered in this study could also be the subject of future empirical studies to provide deeper and more nuanced definitions, approaches for measurement, or assessments of impacts on software engineering outcomes. Comparison of the attributes of software engineering expertise to attributes in other fields could provide more insights into the unique qualities of the software engineering phenomenon.

Our results provide little insight into the relative importance of the attributes (e.g. weighting or criticality) and the effects of contextual factors (e.g. gender or background). We did not conduct an analysis for those insights for our interview study because it would have lacked validity. We did not ask informants about all the attributes, and informants often agreed or amended their thinking when prompted with attributes from other interviews. Future studies can examine this rich area.

With studies like these and the many others that our findings provoke, our research community can begin to understand software engineering not just as a purely technical discipline of tools and processes, but a *socio*technical one, with individual human contributors and their collaborations fueling software progress.

Acknowledgments

This work was supported in part by Microsoft, Google, and the National Science Foundation (NSF) under Grants CCF-0952733, CNS-1240786, and IIS-1314399. Any opinions, findings, conclusions or recommendations are those of the authors and do not necessarily reflect the views of NSF.

References

[1] A. Begel and B. Simon, "Novice software developers, all over again," Int'l Computing Education Research Workshop, 2008, vol. 1, no. 425, pp. 3–14.

[2] M. Hewner and M. Guzdial, "What game developers look for in a new graduate: interviews and surveys at one game company," ACM Technical Symposium on Computer Science Education, 2010, pp. 275–279.

[3] L. Gugerty and G. M. Olson, "Debugging by skilled and novice programmers," ACM Conference on Human Factors in Computing Systems, 1986, pp. 171–174.

[4] J. D. Valett and F. E. McGarry, "A summary of software measurement experiences in the software engineering laboratory," Hawaii Int'l Conference on System Sciences, 1988, pp. 293–301.

[5] H. Sackman, W. J. Erikson, and E. E. Grant, "Exploratory experimental studies comparing online and offline programmmg performance," Communications of the ACM, vol. 11, no. 1, pp. 3–11, 1968.

[6] R. Shackelford, Andrew McGettrick, Robert Sloan, H. Topi, G. Davies, R. Kamali, J. Cross, J. Impagliazzo, R. LeBlanc, and B. Lunt, "Computing curricula 2005: the overview report," SIGCSE Bulletin, vol. 38, no. 1, pp. 456–457, 2006.

[7] T. C. Lethbridge, J. LeBlanc, R.J., A. E. Kelley-Sobel, T. B. Hilburn, and J. L. Diaz-Herrera, "SE2004: recommendations for undergraduate software engineering curricula," IEEE Software, vol. 23, no. 6, pp. 19–25, 2006.

[8] C. S. Miller and L. Dettori, "Employers' perspectives on it learning outcomes," Information Technology Education, 2008, pp. 213–217.

[9] E. M. Trauth, D. W. Farwell, and D. Lee, "The IS expectation gap: industry expectations versus academic preparation," MIS Quarterly, vol. 17, no. 3, pp. 293–307, 1993.

[10] T. C. Lethbridge, "A survey of the relevance of computer science and software engineering education," Conference on Software Engineering Education and Training, 1998, pp. 56–67.

[11] R. E. Kelley, "How to be a star engineer," IEEE Spectrum, vol. 36, no. 10, pp. 51–58, 1999.

[12] D. M. S. Lee, E. M. Trauth, and D. Farwell, "Critical skills and knowledge requirements of IS professionals: a joint academic / industry investigation," MIS Quarterly, vol. 19, no. 3, pp. 313–340, 1995.

[13] E. Brechner, "Things they would not teach me of in college : what Microsoft developers learn later," ACM SIGPLAN Conference on Object-oriented Programing, Systems, Languages, and Applications, 2003, pp. 134–136.

[14] E. W. Dijkstra, "The humble programmer," Communications of the ACM, vol. 15, no. 10, pp. 859–866, 1972.

[15] A. Bryant, "In head-hunting, big data may not be such a big deal," The New York Times, 2013. [Online]. Available: http://www.nytimes.com/2013/06/20/business/in-head-hunting-big-data-may-not-be-such-a-big-deal.html.

[16] S. McConnell, Code complete: a practical handbook of software construction, 2nd Edition. Microsoft Press, 2004.

[17] M. T. H. Chi, R. Glaser, and Ernest Rees, Expertise in problem solving. University of Pittsburgh, 1981.

[18] J. W. Alba and J. W. Hutchinson, "Dimensions of consumer expertise," Journal of Consumer Research, vol. 13, no. 4, pp. 411–454, 1987.

[19] M. P. Robillard, W. Coelho, G. C. Murphy, and I. C. Society, "How effective developers investigate source code : an exploratory study," IEEE Transactions on Software Engineering, vol. 30, no. 12, pp. 889–903, 2004.

[20] K. A. Ericsson, R. T. Krampe, and C. Tesch-romer, "The role of deliberate practice in the acquisition of expert performance," Psychological Review, vol. 100, no. 3, pp. 363–406, 1993.

[21] H. Simon, Administrative behavior, 3rd ed. The Free Press, 1976.

[22] D. H. Gobeli, H. F. Koenig, and I. Bechinger, "Managing conflict in software development teams: a multilevel analysis," Journal of Product Innovation Management, vol. 15, pp. 423–435, 1998.

[23] J. Anvik and G. C. Murphy, "Determining implementation expertise from bug reports," Int'l Workshop on Mining Software Repositories, 2007, pp. 298–308.

[24] G. Jeong, S. Kim, and T. Zimmermann, "Improving bug triage with bug tossing graphs," European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, 2009, pp. 111–120.

[25] A. Podgurski, D. Leon, P. Francis, W. Masri, M. Minch, J. Sun, and B. Wang, "Automated support for classifying software failure reports," Int'l Conf. on Software Engineering, 2003, pp. 465–475.

[26] P. Runeson, M. Alexandersson, and O. Nyholm, "Detection of duplicate defect reports using natural language processing," Int'l Conference on Software Engineering, 2007, pp. 499–510.

[27] D. Bertram, A. Voida, S. Greenberg, and R. Walker, "Communication, collaboration, and bugs: the social nature of issue tracking in small, collocated teams," ACM Conference on Computer Supported Cooperative Work, 2010, pp. 291–300.

[28] J. Aranda and G. Venolia, "The secret life of bugs: going past the errors and omissions in software repositories," Int'l Conference on Software Engineering, 2009, pp. 298–308.

[29] J. M. Corbin and A. Strauss, Basics of qualitative research: techniques and procedures for developing grounded theory, Fourth Edi. SAGE Publications, Inc, 2014.

[30] H. Clark and S. Brennan, Perspectives on socially shared cognition. American Psychological Association, 1991.

[31] L. A. Perlow, "The time famine : toward a sociology of work time," Administrative Science Quarterly, vol. 44, no. 1, pp. 57–81, 1999.

[32] J. Herbsleb, D. Zubrow, D. Goldenson, W. Hayes, and M. Paulk, "Software quality and the Capability Maturity Model," Communications of the ACM, vol. 40, no. 6, pp. 31–40, 1997.

[33] B. W. Boehm, C. Abts, A. W. Brown, S. Chulani, B. K. Clark, E. Horowitz, R. Madachy, D. J. Reifer, and B. Steece, Software cost estimation with COCOMO II. Prentice Hall, 2000.

[34] J. D. Herbsleb and A. Mockus, "Formulation and preliminary test of an empirical theory of coordination in software engineering," European Software Engineering Conference and ACM SIGSOFT Int'l Symposium on Foundations of Software Engineering, 2003, pp. 138–147.

[35] B. W. Boehm, "Verifying and validating software requirements and design specifications," IEEE Software, vol. 13, no. 2, pp. 25–35, 1996.

[36] H. Simon, "A behavioral model of rational choice," Quarterly Journal of Economics, vol. 69, pp. 99–188, 1955.

[37] C. E. Zsambok and G. Klein, Naturalistic decision making. Lawrence Erlbaum Associates, 1996.