Cross-Discipinary Perspectives on Collaborations with Software Engineers

Paul Luo Li
Microsoft
Redmond, WA USA
paul.li@microsoft.com

Andrew J. Ko
The Information School
University of Washington, Seattle
ajko@uw.edu

Andrew Begel
Microsoft Research
Redmond, WA USA
andrew.begel@microsoft.com

***Abstract*—Software engineering teams are usually interdisciplinary, consisting of both software engineers and non-software-engineers. While numerous studies have examined the success and failure of software engineering efforts from the perspective of software engineers, little is known about perspectives of expert non-software-engineers. In this study, we interviewed 46 experts across 10 roles at Microsoft (artists, content developers, data scientists, design researchers, designers, electrical engineers, mechanical engineers, product planners, program managers, service engineers) about their collaborations—good and bad—with software engineers. Overall, our experts described great software engineers as masters of their own technical domain, open-minded to the input of others, proactively informing everyone, and seeing the big picture of how pieces fit together. We discuss implications of our findings for practitioners, educators, and researchers.**

Keywords - team work, interdisciplinary teams, collaboration, software engineering expertise

I. Introduction

Engineering of software commonly entails collaborations not only between software engineers, but also with *expert non-software-engineers*. Artists [1], data scientists [2], designers [3], writers [4], program managers [5], and other experts perform essential tasks, important to the success of software engineering teams.

While software engineering educators [6], researchers [7], and practitioners [8] have examined success and failure of software engineering efforts from the perspective of software engineers, we know little about the perspectives of non-software-engineers. Their perspectives on attributes of software engineers that lead to successful collaborations is a gap in our understanding of why software engineering efforts succeed (or fail). We address this gap with an interview study of 46 expert non-software-engineers, across 10 roles at Microsoft. Our two research questions are

- What do expert non-software-engineers think are attributes of software engineers that lead to successful collaborations?
- How and why are these attributes important for successful engineering of their products?

In the rest of this paper, we describe our approach to answering these questions, and detail our discoveries from our interviewees' insights. We end with discussion of two meta-problems for future work.

II. Related Work

Numerous studies and industry reports have examined software engineers' perspectives on software engineering expertise, but few directly examine perspectives of other experts that engineers collaborate with. Trifonova et al. surveyed more than 50 research publications about software development projects with *artist* participation [9], finding that artists are involved in the engineering of software and that they have needs (e.g. tools, education, and engagement methods) and concerns (e.g. aesthetics) that differ from software engineers. The survey indicates that studies have not examined artists' perspectives on software engineering expertise.

Begel and Zimmermann surveyed Microsoft software engineers about the questions they would like *data scientists* to answer (e.g. how do users typically use my application and what parts of a software product are most used and/or loved by customers) [2]. Fisher et al. examined challenges analyzing 'big data' at Microsoft [10]. These reports indicate data scientists help software engineers perform important and challenging functions.

Numerous studies (e.g. [11] [12]) indicate that *project managers* help teams manage risks in software engineering projects. Some of the top issues like scheduling and timing risk indirectly imply that great software engineers managed, mitigated, or avoided these issues.

Lee and Mehlenbacher (in follow-up to a 1991 study that interviewed software engineers at DEC about attributes they wanted in *technical writers* [13]) surveyed 31 technical writers, including 16 that worked for software companies, about working with subject matter experts (SMEs) [4]. The most commonly reported issues when working with SMEs were 'time and accessibility', 'respect for the documentation process', and 'communication skills'.

In the well-known book "The Inmates Are Running the Asylum," Cooper describes software engineers ('the inmates') as making too many engineering decisions that impact product usability and providing biased information (e.g. over estimating costs of features) to intentionally derail projects ('running the asylum') [14]. The author calls for *designers* to use a disciplined approach (e.g. using personas) to ensure better products.

Existing literature indicates that perspectives on software engineering expertise from expert non-software-engineers is an important knowledge gap that neither existing literature nor our previous study examining attributes of great software engineers (from the perspective of software engineers) [15] have addressed directly.

III. Method

To address our research questions—from the perspective of expert non-software engineers, what are attributes of software engineers that lead to successful collaborations, and how do those attributes contribute to their collective success—we sought to balance depth of understanding, breath of perspectives, and relevance of insights. We conducted semi-structured interviews with 46 senior-level employees across 10 distinct roles at Microsoft. We asked these them open-ended questions about attributes of engineers they have

worked with that lead to success, as well as semi-structured questions about a set of attributes that software engineers self-identified as important [15] (to overcome saliency bias and to ensure holistic thinking). We analyzed the data using an inductive approach, identifying key themes about software engineering expertise and distinguishing them by role.

A. Selecting Expert Non-Software-Engineers

Though non-software-engineers in many roles work with software engineers, we wanted to focus on ones most likely be essential to successful engineering efforts. We selected non-software-engineers within two Microsoft-defined categories—Engineering and Hardware Engineering—since nearly all software engineers are in these two areas. Because of the organizational arrangement, expert non-software-engineers in these two professions at Microsoft are likely collaborating closely with software engineers.

To ensure that we obtained in-depth and relevant knowledge, we focused on full-time employees at the 'senior' level or above—typically 5+ years of experience at Microsoft or elsewhere—based on their titles in the company address book. Via the hiring and/or promotion processes, these individuals have been affirmed by their peers as experts in their field. We organized our experts into 12 roles based on our understanding of titles. We pruned down the roles to 10—requiring at least 50 employees in that role in the Seattle area—to ensure sufficient people to sample for interviews. This scoping was also necessary to facilitate face-to-face interviews, and to ensure relevance of insights, as Microsoft's product development is centralized around Seattle, WA. We solicited interviewees via personal emails from the 1st author, who was a Microsoft employee.

We conducted interviews in a round-robin among the roles, which facilitated identification of cross-cutting themes and interesting questions. Of the 102 people recruited, we interviewed 46 across the 10 roles (a response rate of 45%), including at least 3 employees in each role, (see Table I).

B. Interview Protocol

After explaining the study and obtaining consent, we asked: "*What is your background? And how did you come to be a <role> at Microsoft?*", "*What—in your opinion—is the function of <role> in engineering teams?*" Then we asked: "*How have you engaged with developers?*", "*How does that engagement vary in the various phases of development?*" These provided important context for understanding and interpreting their perspectives. Next, we asked: "*What are the positive attributes of good developers you've worked with that you believe contributed to successful outcomes?*", "*What are negative attributes that you've seen contribute to less than successful outcomes?*"

We then asked interviewees about a list of 54 attributes of software engineering expertise from our previous studies of software engineers [15]: "*Read through the list of attributes and note any that stood out as too high or too low, and tell us why. Then we'd like your top five attributes, from the perspective of successfully collaborations with <role>.*" For example, the highest rated attribute was: 'Pays attention to coding details, including error handling, memory consumption, performance, and style.' Examining this list helped interviewees overcome saliency effects; numerous interviewees amended or clarified their perspectives after reading the attributes.

When appropriate, we asked clarifying questions, seeking explanations, details, examples, etc. This facilitated understanding why that attribute was important in real-world engineering projects. We concluded by asking, "*Ideally, how would you like to see people in your role and developers collaborating together to engineer software products?*" Interviews lasted approximately one hour.

C. Analysis

To analyze the 38+ hours of interviews and 350,000+ words of transcripts, we analyzed the data using a 'Straussian' grounded theory approach [16], focusing on software engineering expertise, making three coding passes through the data. First, we read through the entire transcript to gain an overall understanding of the data and to tag relevant discussions. A second pass identified key themes and highlighting key excerpts. Third, we analyzed the qualitative data for each role separately. We extracted contextual perspectives about collaborations, and then analyzed important attributes for each role. Finally, we interpreted the data through the lens of both prior work and contexts of interviews. The first author performed the coding, and then the other authors validated the interpretation. Additional detail about the coding and approach is in the first author's dissertation [17].

IV. Results

Broadly, our expert non-software-engineers described great software engineers as masters of their own technical domain, open-minded to the input of others, proactively informing everyone (enabling others to make optimal decisions), and seeing the big picture of how the pieces (even non-code-related parts) fitted together. Table I shows a synopsis of the attributes that our experts felt contribute to

TABLE I. Attributes Contributing to Success From the Perspective of Expert non-software-engineers

Collaborator	Attributes Contributing to Success
Artists	Resourceful; egalitarian; voice of engineering reality; open to changing objectives and requirements; hardworking; artistically aware
Content Developers	Technically competent; customer oriented; responsive; cognizant of impact to others
Data Scientists	Data-driven; thoroughly knowledgeable of their software; willing to change directions
Design Researchers	Open to uncertainties of customer reactions; avoids self-referencing; respectful of qualitative research
Designers	Respectful of the design discipline; not rushing to coding; seeks shared understanding
Electrical Engineers	Knowledgeable about electrical engineering; system thinkers/problem-solvers
Mechanical Engineers	Knowledgeable about mechanical engineering; system thinkers/problem-solvers
Product Planners	Avoids self-referencing; not dogmatic; helps others understand technicalities
Program Managers	Proactive communicators; takes on new challenge to enable team success; creates shared understanding technical reasoning
Service Engineers	Respects the networking discipline; open to feedback and collaborations

success (and avoid failure). In the following sections, we detail how these attributes manifest (often in different ways) in collaborations with experts.

In this paper, we focus on roles and insights that were the most interesting and the most relevant to overall themes across all interviews—omitting design researchers, mechanical engineers, product planners, and service engineers. Interested readers can refer to the first author's dissertation [17] for more details.

A. Program Managers

Across all roles, the largest group, by far, was program managers (PMs); nearly every software engineering team at Microsoft had a PM. We focus on 'feature' PMs, who worked closely with software engineers in the development of products. Almost all interviewees had deep technical knowledge. Most had degrees in computer science or their specialty area, and many had experience as software engineers. One interviewee stated that being a PM at Microsoft was not merely being a 'schedule jockey'.

At a broad level, program management at Microsoft combined requirements definition and prioritization (commonly done by engineering managers elsewhere), scheduling, tracking (typically done by project managers elsewhere), as well as coordinating with other teams and experts. PMs were the primary points of contact for many other expert non-software-engineers. For some, this bridge was beneficial because PMs helped facilitate conversations with engineers. However, for others, PMs were a hindrance, blocking access to engineers who had the accurate technical answers they needed.

With their focus on facilitating successful completion of projects, our PMs emphasized attributes that helped to avoid problematic plans and schedule deviations. Our interviewees felt that software engineers often had critical insights, during plan formulation and during execution; therefore, they needed to 'speak up':

Be blunt and honest with me. Tell me how it is, why it is… I want to know it upfront. If it's sugar-coated, you can't address it in as timely manner as probably as needed or in a direct manner as probably as needed… That can later come back and cause more problems than good.

– Senior HW Program Manager, Devices

Our interviewees felt that engineers, too often, get recognition for 'fighting fires.' PMs preferred to work with great engineers 'who prevent the fires before they even start.' Great engineers foresaw challenges with plans, asked the right questions, and helped PMs to *avoid* problems, Literature on project management commonly discusses risk management for teams [11][12]. Our findings indicate that an important aspect of risk management may be *proactively* providing needed information:

And we have methods and we have ways to do it. We have a daily scrum. You should just go surface these things there, just don't sleep on them…there is some level of transparency between the devs… they minimize risks and they surface risks and the PM or the dev manager can have a backup plan. The more you identify these problems early in the process, the better. If you just keep them as surprising issues at the end, nobody is able to handle them. When the plane is landing, you cannot just go say, "Oh, the engine is not working now. Oh, I knew about it a week ago."

– Senior Program Manager Lead, Web Applications

B. Electrical Engineers

Our electrical engineers described their role as "making physical things with electrons flowing through them," typically consumer electronics (e.g. Xbox). All our electrical engineers had degrees in electrical engineering.

Our interviewees stated that the software engineers they interacted with were predominantly 'embedded' software engineers (also referred to as 'firmware' engineers), who had extensive knowledge of electronics and hardware, far beyond the knowledge of the typical 'Windows' software engineer:

It could be something like sampling registers, checking for button presses, reading the data from an optical engine and then doing something with it… That's the embedded firmware software aspect of it. There's a tight coupling of system level design architecture between the hardware folks, which is myself, and the embedded firmware folks.

– Senior Electronic Engineer, Devices

Our electrical engineers worked on program teams with software engineers (and other experts, such as Program Managers and Mechanical Engineers) to produce consumer electronics: Xbox (encompassing Kinect), HoloLens, Surface, Keyboard/Mice, and phones. At project initiation, a group of very experienced engineers from various disciplines gathered to scope the project, choosing the functions and features to deliver as well as hardware and software components. Many of these choices were tightly coupled, had great uncertainty, and occurred years in advance of actual product development.

For example, one interviewee described selecting the scrolling wheel for a mouse. Mechanical engineers would choose the physical component based on physical dimensions and functional requirements. Electrical engineers would then take into consideration how data was obtained from the component (e.g. optically or electronically) to decide how to read and transport the data to microcontrollers. Embedded software engineers would then decide how to communicate the data to the PC, including considerations for efficient algorithms and sampling intervals. When problems arose, the team would have to decide where an adjustment—mechanical, electrical, or firmware—should best be made.

After plans were set, individual disciplines independently fleshed out and produced their own parts. The program team would continue to work closely throughout the development process. In addition to periodic sync ups, the program team would usually come together during major milestones (e.g. prototype complete) to verify that the project was progressing on-schedule.

All our electrical engineers felt that great software engineers should be able to 'speak hardware.' They needed to understand the limitations and capabilities of available hardware, as well as work within electrical constraints:

The really great ones have a really good understanding of both, the software and the hardware. Like I said, figuring out the limitations of what the hardware can do and what it can't do and asking the right questions and phrasing it right to get it either in software lingo or hardware lingo. You know, the people that can do that are fairly rare…there's different terminologies and different expectations.

– Senior Architect, Devices

A common complaint was the lack of understanding about scheduling. Our interviewees stressed that the hardware operated with very different timelines (designs can take

months to produce, fabrication can take weeks, and testing can be another several weeks) and therefore, software engineers who did not understand the differences were difficult to work with:

Since [software] programs are very malleable, they can make changes up to the last second, right? … And it's hard to get across, "No this is fixed. Once it's burned it's not going to change."

– Senior Architect, Devices

Finally, since their products were composites of hardware and software, problems could commonly be solved by software or hardware, but with different compromises. Lack of holistic understanding often resulted in inferior products despite similar hardware (e.g. worse battery life).

C. Artists

Our artists were concentrated within teams that develop games. All our interviewees had training as artists, with industry experiences in games (e.g. Ubisoft, Bungee) or entertainment (e.g. Disney, Industrial Light and Magic) prior to joining Microsoft. Our expert artists emphasized that games were entertainment products, necessitating a *holistic* experience, including both technical game play as well as 'look and feel.' Our interviewees felt that, in years past, games were commonly engineer-constrained (i.e. teams built what was technically possible), leading to engineering dictating direction of projects, often (imperiously) influencing artistic choices. However, with technology advancement and industry maturation, interviewees felt that game development was increasingly even-handed, with artists having an equal voice in decisions. They felt game development Microsoft as balanced between artistic and engineering concerns.

The primary challenge facing their teams was needing to 'push the envelope,' under technical constraints while shipping on time. Due to the competitive nature of the gaming industry, interviewees discussed needing to offer something *outstanding*, which was checked by limits of technology (e.g. hardware) as well as time (e.g. yearly refresh cycles or shipping for the holiday season). Consequently, many attributes of great software engineers emerge from collaborations to overcome these challenges. Our interviewees had an overarching desire for software engineers to have some understanding of the art domain, with the focus on mindset and language. This enabled productive and meaningful communications—understanding 'what we're talking about.'

Our interviewees described great software engineers as having great technical knowledge, which was valuable in three ways. First, it enabled them to scope the project to keep within bounds of technical feasibility and schedule. Our interviewees hinted that artists suggested "pure fantasy" and needed engineers to be the "voice of reality." Second, they offered alternatives or novel possibilities, even predicting future technical advances. This enabled the team to achieve even better look and feel than envisioned by artists. Third, great engineers worked with artists on creative solutions:

[Engineers] who, like the MacGyver kind of attitude where, "Hey, given these constraints, here's what we can make." … So the ability to be able to say, "Well, what is it gonna take to get us there in the timeframe that we need?" And so the best software engineers that we've worked with from an art perspective are the ones who can think quickly on their feet and improvise to come up with creative solutions to help meet the needs.

– Technical Art Director, Gaming

('MacGyver' is a fictional American TV character famous for being resourceful and possessing expansive knowledge.)

Furthermore, our interviewees believed that engineers in the game industry needed certain mentalities. Foremost, to be successful, teams needed to 'push the envelope'; therefore, engineers could not be risk-adverse. Second, engineers needed to be open-minded and adaptable. 'Correct' and 'best' may not be known ahead of time, prototyping was often required to understand the optimal solution (the mentality behind knowing by doing [18]). Engineers needed to adapt to "deliver what's actually useful and maybe not what was on paper." Finally, our interviewees felt that Engineers needed to be hardworking. In addition to myriad challenges throughout development, extra work was often needed at the end to 'push the product across the finish line.'

D. Designers

Our designer interviewees characterized their role as ensuring enjoyable user interactions by engaging visual and interaction design process. Our designers had backgrounds in art and graphic design, and most worked on teams with high concentrations of user-facing features.

Interaction designers focused on ensuring that users can easily use interfaces and can understand the information presented in those interfaces. Our interviewees felt that interaction design also involved 'information architecture', showing users the right amount of information, at the right time and at the right place, to enable users to accomplish their tasks without overwhelming them. In contrast, visual designers made visual elements, including icons, logos, background, layouts, and even marketing materials. They also ensured that the software product was atheistically sound. While sharing some of the same tasks (e.g. creating visual assets) as Artists (Section C), in addition to visual aesthetics visual designers also considered usability. Adjusting for context of the user, rather than focusing on athletics, is a key differentiator between Artists and Designers. In small teams, designers provided both visual and interaction designs, contributing whatever the team needed.

Designers usually worked alongside software engineers to produce features. The prevailing sentiment among interviewees was that engineers were responsible for what happened "underneath the covers," getting the software feature to "work just right," whereas designers were responsible for how users interacted with the software feature, ensuring that users can use the software.

Interviewees felt that the primary challenge facing software engineers and designers was reducing complexity. They felt that many features involved large amounts of technology, which would overwhelm and frustrate the typical user; therefore, engineers and designers needed to work together to iteratively design features that were easy and enjoyable for users.

The dominant sentiment among our interviewees was that great engineers left design decisions to designers. Great

engineers respected the design discipline and did not think that they could do the designers' job:

An important attribute is when developers also respect the expertise of designers, understanding that there's a time for feedback… but for them to also defer to designers when it comes to the design and the user experience.

Because just as a user experience designer will not tell a developer how to do their job, so too should a developer be very respectful of the designer's position and their years of expertise in the field.

– User Experience Visual Designer, Gaming

Many interviewees discussed cases when engineers lacked respect. Some engineers, when encountering problems with design, would produce fixes without consulting designers. These would usually be suboptimal as the engineers had neither design training nor the 'hundreds and hundreds of hours' of experience observing actual users. Some engineers felt that designers only made things 'look pretty'—telling them to "put some UI on it" at the end of development. This commonly resulted in unusable features that required substantial redesign.

Our interviewees also felt that great engineers did not rush into coding, rather they took the time to fully understand the problem and worked with designers on optimal trade-offs. This helped to avoid suboptimal designs that were unchangeable (or too costly to change):

We end up getting dev involved in building code too early… we are doing design and research that may actually go against what is being built and it becomes this awkward…we are saying you actually need to change it based on our user research, but they have already invested time into it so they don't want to change it or it's already too far along… the end product doesn't meet the user's goals or they are not able to use it as easily as they should.

– Senior UX Designer, Web Applications

Finally, interviewees felt that great engineers worked to clarify understanding, identifying missing elements or inconsistencies in designs. As designers and engineers may have different interpretations of the problem, great engineers—often using face-to-face meetings—worked to avoid divergent efforts:

…there were a couple of developers that really understood how to pull information out of you. So if they didn't understand something, they would drill in deeper and get more clarity to the point where there wasn't any ambiguity so you both knew exactly what was expected for the best outcome… And that's probably one of the most important aspects of the process, is just being able to come together and sit down and talk through things. And the more you can sit down and get clarity upfront, the more successful the outcome is going to be at the end.

– Senior UX Designer, Web Applications

E. Data Scientists

Data scientists existed in many engineering teams across Microsoft, doing disparate tasks. This is likely because 'data' pervades software engineering: the software feature itself, logs for usage analysis, or the target of software features. There was no simple grouping or explanation of the 'data scientist' role within Microsoft. We discuss the three kinds of data scientists we interviewed separately.

Some data scientists at Microsoft were essentially engineers; our interviewee explained that his team was converted to be data scientists because their features involved extensive 'experimentation'. Rather than a 'build to last' mentality, their team had a 'fail quickly' mentality—getting to the best answer quickly by iterating through variations. This may reflect emerging trends within the software engineering domain to better leverage data [19]. All members of his team, both data scientists and software engineers, performed the same set of tasks, which expedited development and reduced 'lost in translation' problems:

I'm used to our model, where data scientists also are engineers themselves. I think that works better… There's no handoff. Right? There's no interpretation… I think if you have scientists who can actually implement code and ship it, that's useful.

– Principal Applied Sciences Manager, Web Applications

In discussing engagement with software engineers, our interviewee referred to working with platform teams on infrastructural improvements. The interviewee singled out one attribute—data-driven—as not given enough attention by software engineers:

I think data-driven is very low on the list, which surprises me… So I guess there's an opinion that measuring the software outcomes is not important, but I think that's extremely important. I think a lot of work you do needs to be data-driven. You can't just say, "Well, I have a feeling this will work."

– Principal Applied Sciences Manager, Web Applications

A second kind of data scientist were those who prototyped data features that software engineers then implemented:

My neighbor is a software engineer and my neighbor's neighbor is a software engineer…We do models then we kind of close the gap between business and engineers. We develop strategies and then we figure out how we want to do [them]. Then software engineers, they help realize our wishes, so we work really close.

– Principal Data Scientist, Web Applications

Our interviewee felt that, in his domain, great software engineers need to be detail-oriented with a full technical understanding of the software system. Since his team dealt with financial transactions, even minor issues could be costly. Great software engineers fully understood the risks and consequences of their choices, since 'I don't know why' was not acceptable when problems resulted in loss of money.

Our interviewee further described great engineers as flexible and fast, since issues involving money needed to be fixed immediately, often outside of regularly planned development cycles. Our interviewee appreciated engineers who quickly fixed (or at least temporarily patched) issues:

When you are working on business, you actually impact the customers in real time. You cannot ask the customer, "Okay. We know that there's a bug. Wait for three days, we're going to fix the bug." It's not going to work… Sometimes it can be short term solution, but need to pay immediate attention.

– Principal Data Scientist, Web Applications

The third type of data scientists produced software that reported on product use: monitoring systems that leveraged logs to report on the status of the software product. These were 'shadow' systems; their value was in providing information about the actual product and would not exist without the original. Our interviewees believed that insights about the software system was essential for improvement, allowing teams to track progress, assess outcomes, and identify new opportunities.

In this process, our interviewees felt that data scientists were *consultants* to engineers. Data scientists helped to clarify vague concepts (e.g. success and failure) and to instantiate and track them with metrics. Data scientist and software engineers

collaborated to analyze the data iteratively and quickly improve the software product:

…it's actually more that the data scientist is more on the engineer side to help improve the system…

.. analyzing the data, provide a daily scorecard. I provide a metrics that the developer can come and see whether their changes improved, with what they have done actually had made the system better. But at the same time data is used to improve our system automatically, programmatically, interacting [sic] with developer.

– Principal Data Science Manager, Web Applications

In addition to being data-driven and improvement-minded, our interviewees also wanted engineers to be intimately familiar with their software products, making error-free changes, and to proactively notify data scientists when changes could affect reporting:

Data, it's very hard to be accurate. Data is very hard to be correct. I get garbage data almost all the time…

However, I work with some developers that are just incredible. I get mail from them. "Hey, I'm changing this today because of this. I realize the data I feed to you can be better. I make these changes." That's the best experience I've ever had.

…And they make my life much better. And the worse thing is, sometimes I don't even know the data is wrong, and I publish the data. I make a big business decision based on the data, and it can hurt. It can be millions of dollars because the data is wrong. So, yes, pay attention to detail!

– Principal Data Science Manager, Web Applications

Since data validation and data cleansing are commonly the most expensive and time-consuming parts of data analyses [10], our interviewees' perspectives likely reflect a desire for software engineers to help ameliorate a pain point.

F. Content Developers

Our content developers were generally technical writers (one a writer by profession, the other two were software engineers prior to transitioning to content development). All viewed content development as *bridging engineering intent and customer needs*. Our experts worked with software engineers to produce a wide variety of content to help customers understand the software product (e.g. display string in dialog boxes and 'how to' information), as well as content to address post-release concerns from feedback channels (e.g. MSDN, Customer Service and Support).

While acknowledging that content was ancillary, our writers felt that great software engineers treated them (and their writing tasks) with respect. Great software engineers did not ignore or put-off requests from technical writers, which was especially important for products with fast shipping cycles (e.g. online services):

It's continuous because there are things coming out. [We] ship something every day… you have to go in there and understand how something is going to ship, have an idea of that sort of thing, and then you put up something quickly…

– Senior Content Developer, Enterprise

When shown the full list of attributes, all our interviewees commented that *technical competency* was critical. They had assumed the attribute was a given:

…if the code is broken, it doesn't matter what word I put.

– Senior Content Developer, Enterprise

Software engineers' technical competency was also important in avoiding problematic decisions (e.g. breaking existing workflows) and to understand potential pitfalls. These allowed engineers and writers to work together to create appropriate explanations and guides for customers.

Finally, our interviewees felt that great engineers valued customer needs. They accept that many customers may less technically savvy with low computing self-efficacy. Therefore, they undertook constructive actions to understand customer problems and to address the underlying confusion.

V. Discussion and Implications

We observed four overall themes in attributes that software engineers needed to excel as collaborators in interdisciplinary teams. First, our interviewees expected software engineers to *be experts of their own technical domain*. Our experts recognized that engineers performed a critical task—writing code—without which the product would not exist. This expertise also extended to knowledge about options, trade-offs, and workarounds, based on the 'truth in code' and possible future technical developments. Non-software-engineers rarely had this knowledge (nor were they expected to); therefore, for the team to be successful, great engineers were expected, foremost, to excel in their own domain.

Second, software engineers were expected to *proactively communicate information to their team*. This included, during the planning phase, helping to scope and innovate, as well as updating timelines and expectations during development. Great engineers *proactively* ensured that teammates had information they needed to make decisions.

Third, our interviewees overwhelmingly wanted software engineers to recognize that they were *not* experts on all aspects of the product, staying *open-minded to the input of other kinds of expertise*.

Finally, underlying all previous attributes was our interviewees' desire for engineers to *see the big picture of how pieces fit together across disciplines*. Great engineers understood how all experts and the tasks they performed contributed to the success of a product, enabling them to optimally leverage their expertise.

Generally, when software engineers lacked these attributes, interviewees described engagements that fueled resentment and led to poor software products. Consequently, software engineers should, at minimum, strive for proficiency in these areas. Nonetheless, contexts for engagements greatly influenced the importance of various attributes, as we discuss in the next two sections.

A. Conditions for Equality

Some experts felt that software engineers did not view them (and their discipline) as equals, reflecting themes reported in past studies (e.g. [4], [14], [20]). While we did hear some evidence of these dysfunctional collaborations, most experts across our sample at Microsoft expressed feeling of overall equality in their collaborations, including Artists and Electrical Engineers. Better understanding actions and conditions that lead to equality—real or perceived—may be valuable future work.

B. Challenging Engineering Processes

Conflicts in engineering processes also produced collaboration challenges, mirroring prior studies of specific software domains. For instance, game development is highly-competitive [21] and overly ambitious scope is prevalent. A survey of 20 game projects (including Diablo II, Unreal Tournament, and Resident Evil 2) reported it occurring in 79% of teams [22]. This engineering process is likely detrimental to code quality [23] and software engineers dislike being *expected* to do extra work [15]. Easy solutions may not exist, as any additional resources would probably be put towards more features (to 'push the envelope'), instead of reducing the strain on the team.

Consumer electronics was another problematic area. Due to the interconnected nature of consumer electronics, great software engineers needed to think through implications of decisions (upfront) as well as comprehend and communicate decisions and implications with experts in other domains. However, asking software engineers to have expertise in *multiple separate engineering disciplines* may not be realistic; research into new engineering processes that address these structural problems may be needed.

VI. Threats to Validity

As with any empirical study, our study has many threats to validity. Our study's construct validity is threatened by our interviewees' understanding of the attributes of software engineering expertise from our previous study [15]. This was partially mitigated by our selection of experts that worked with software engineers at Microsoft. Also, we did not attempt to pigeonhole attributes described by our interviewees—we examined each role separately, retaining their contexts.

Threats to internal validity come from our interpretation of the data; other researchers may have interpreted the data differently. Our familiarity of Microsoft's engineering processes (having shipped engineering features in Windows) and of the domain (having published numerous research papers), may have improved our interpretation.

Our study is qualitative and specific to Microsoft. In smaller organizations, experts may take on multiple roles, in both technical and non-technical areas. Replicating this study by other researchers and in other contexts can strengthen and expand on our findings.

VII. Conclusion

As software is increasingly embedded into our social and physical worlds, software engineering is becoming increasingly interdisciplinary. This paper suggests that with this change, great software engineers need to be more than masters of code, but also masters of interdisciplinary communication and systems-level thinking. Future work should explore the implications of this findings, investigating how these skills impact software teams and ultimately how to effectively teach these skills to future software engineers.

References

[1] M. Hewner and M. Guzdial, "What game developers look for in a new graduate: interviews and surveys at one game company," in *Proc. SIGCSE*, 2010.

[2] A. Begel and T. Zimmermann, "Analyze this! 145 questions for data scientists in software engineering," in *Proc. ICSE,* 2014.

[3] H. R. Beyer and K. Holtzblatt, "Apprenticing with the customer," *Commun. ACM*, vol. 38, no. 5, May 1995.

[4] B. Mehlenbacher, "Technical writer/subject-matter expert interaction: the writer's perspective, the organizational challenge," *Tech. Commun.*, vol. 47, no. 4, 2000.

[5] J. Aranda and G. Venolia, "The secret life of bugs: going past the errors and omissions in software repositories," in *Proc. ICSE*, 2009.

[6] Joint Task Force on Computing Curricula, "Software Engineering 2014: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering," 2014.

[7] A. J. Ko, R. DeLine, and G. Venolia, "Information needs in collocated software development teams," in *Proc. ICSE*, 2007.

[8] E. Brechner, "Things they would not teach me of in college: what Microsoft developers learn later," in *Proc. OOPSLA*, 2003.

[9] A. Trifonova, S. U. Ahmed, and L. Jaccheri, "SArt: towards innovation at the intersection of software engineering and art," *Inf. Syst. Dev.*, 2009.

[10] D. Fisher, R. DeLine, M. Czerwinski, and S. Drucker, "Interactions with big data analytics," *interactions*, vol. 19, no. 3, May 2012.

[11] B. W. Boehm, "Software risk management: principles and practices," *IEEE Softw.*, vol. 8, no. 1, 1991.

[12] J. Ropponen and K. Lyytinen, "Components of software development risk: how to address them? a project manager survey," *IEEE Trans. Softw. Eng.*, vol. 26, no. 2, 2000.

[13] D. Walkowski, "Working successfully with technical experts—from their perspective," *Tech. Commun.*, vol. 38, no. 1, 1991.

[14] A. Cooper, *The Inmates Are Running the Asylum*. Sams - Pearson Education, 1999.

[15] P. L. Li, A. J. Ko, and J. Zhu, "What Makes A Great Software Engineer?," in *Proc. ICSE,* 2015.

[16] K. Stol, P. Ralph, and B. Fitzgerald, "Grounded Theory in Software Engineering Research: A Critical Review and Guidelines," in *Proc. ICSE, 2016.*

[17] P. L. Li, "What Makes a Great Software Engineer," Ph.D. thesis, University of Washington, 2016.

[18] R. C. Schank, T. R. Berman, and K. A. Macpherson, "Learning by doing," in *Instructional-design Theories and Models: A New Paradigm of Instructional Theory*, 1999.

[19] Economist, "Data, data everywhere," *Economist*, Feb-2010.

[20] G. P. Zachary, *Showstopper!: The Breakneck Race to Create Windows NT and the Next Generation at Microsoft*. Free Press, 1994.

[21] J. Blow, "Game Development: Harder Than You Think," *ACM Queue*, vol. 1, no. 10, 2004.

[22] F. Petrillo, M. Pimenta, F. Trindade, and C. Dietrich, "Houston , we have a problem ...: A Survey of Actual Problems in Computer Games Development," in *Proc. SAC*, 2008.

[23] N. Nagappan and T. Ball, "Use of relative code churn measures to predict system defect density," in *Proc. Int'l Conf. on SWE*, 2005.