Understanding and Improving Software Build Teams

Shaun Phillips
University of Calgary
Dept. of Computer Science
phillist@ucalgary.ca

Thomas Zimmermann
Microsoft Research
Redmond, WA, USA
tzimmer@microsoft.com

Christian Bird
Microsoft Research
Redmond, WA, USA
cbird@microsoft.com

ABSTRACT
Build, creating software from source code, is a fundamental activity in software development. Build teams manage this process and ensure builds are produced reliably and efficiently. This paper presents an exploration into the nature of build teams—how they form, work, and relate to other teams—through three multi-method studies conducted at Microsoft. We also consider build team effectiveness and find that many challenges are social, not technical: role ambiguity, knowledge sharing, communication, trust, and conflict. Our findings validate theories from group dynamics and organization science, and using a cross-discipline approach, we apply learnings from these fields to inform the design of engineering tools and practices to improve build team effectiveness.

Keywords
software; build; teams; effectiveness; group dynamics; organization science; communication; conflict; trust.

Categories and Subject Descriptors
H.5.3. [**Group and Organization Interfaces**]: Computer-supported cooperative work

General Terms
Human Factors; Management.

1. INTRODUCTION
Defining Build

Build, the process of creating software from source code, is an essential part of software development. Generally speaking, a build process consists of two phases: *compilation*, where source code written in high-level computer languages (e.g., C++) is transformed into machine code that can be executed by a computer, and *packaging*, where the machine code is bundled into an installation package so that it can be deployed to users.

In real-world scenarios, a build process will likely consist of many more complex tasks. For instance, there may be scheduling of build machines, build metric collection (such as time or power consumption), security checks, so-called "smoke" tests to ensure some baseline level of quality, language localization, reporting of build results, and triaging of build problems. All together, the collection of scripts, programs, and services that comprise and automate a build process is called a *build system*.

The build process is cyclical. Development teams write code which at some point is processed by the build system. The output, the build, is deployed back to the development teams to verify the changes [19] and the cycle begins again. This cycle is critical to delivering high-quality software on-time and on-budget [9, 22]. As stated by one of our interview participants, build is "the heartbeat of an organization." However, the cycle is fragile: building will fail if developers write incorrect code, and developers are unable to verify their work unless their code is built.

While building the quintessential "Hello World" program can be done in seconds, in large-scale software development building can be very complex and take many hours to complete [28]; at Microsoft, some large projects require up to six hours of build-time. If a long build process fails (e.g., from a compilation error), developers must wait much longer than expected for the build to be available, which delays their change verification and puts pressure on the development schedule. An unreliable, failure-prone build process is analogous to an irregular organizational heartbeat.

The Formation of Build Teams

Build systems change at about the same rate as the source code they build [31], and over time build systems can become very complex. Our studies show that the "builder" role can arise in an organization when it is inefficient to have many developers learn and manage a complex build process. As one interview participant quipped, builders are a "response to irritation." To illustrate how complicated build systems can become, one senior builder noted that new build team members need "about three months of experience before they can confidently manage the build process on their own."

Build teams tend to form organically in response to changes in organization size and build complexity—they are emergent groups [12]. This formation is different than other teams in a software development organization, which usually have their work scoped and staffing needs met before the development process begins. In our studies, we found that early builders at Microsoft were developers thrust into the role by necessity and the perception that they understood the build system better than others. While the role is now more formalized, how build teams form has led to role ambiguity, which we discuss in our findings.

It is difficult to estimate how many organizations have build teams as they are not a traditional topic of study. However, based on information on their blogs and websites we know that build teams exist in Microsoft [9], Facebook[1], Google[2], Gentoo Linux[3], and Eclipse[4]. In addition, at the 2013 International Workshop on Release Engineering [2] (see proceedings for individual papers and talk abstracts), presentations from NetFlix, Mozilla, LinkedIn, and GNOME (as well as workshop attendees from companies including Qualcomm and VMWare) all indicated that build was a complicated, evolving, and time consuming process that required dedicated teams in their organizations. Thus, we can reasonably assume that build teams are not uncommon in large software development organizations.

The formation of a build team creates an environment where those who primarily write source code are distinct from those who primarily build source code. Build complexity is abstracted away from other teams and onto the build team. For example, developers may build a part of the project on their own machines, but rely on a build team to build the entire project, for all supported languages, hardware architectures, platforms, and SKUs on multiple machines in parallel.

Build Team Effectiveness

We define build team effectiveness as how reliably and efficiently the team produces builds. To frame our initial explorations into build team effectiveness, we used Cohen and Bailey's framework for analyzing team effectiveness [7], which gives four characteristics to consider: *environment*, *task design*, *internal processes*, and *group psychological traits*. Our interviews, discussed in the Section 3, were divided into four parts to examine each characteristic, albiet in the context of build.

Our findings show that many impediments to build team effectiveness are social rather than technical. The challenges are similar to problems studied in group dynamics and organization science: role ambiguity, knowledge sharing, and intergroup communication, trust, and conflict [12]. Subsequently, we use theories from these fields to help explain our findings and drive our analysis. A primary contribution of this paper is the validation and application of these theories, not typically seen in software engineering research, to inform the design of tools and practices that can improve team effectiveness in software development organizations.

This paper is organized in the following manner: Section 2 discusses relevant software engineering, organization science, and group dynamics research; Section 3 describes our three studies conducted at Microsoft; Section 4 reports on our study observations; Section 5 synthesizes our findings into tools and practices and provides practitioner feedback; in Section 6 we examine the broader context of our work; and finally, the Section 7 highlights the main contributions this paper.

2. RELATED WORK

There is a large body of work on the technologies underlying the build process. Software compilation tools and processes, as well as parallel and distributed application development, have many dedicated international venues. For software deployment, technologies to make builds available for use (e.g., setup packages), Dearle provides a high-level overview of current work, approaches, and challenges [10].

Studies on the overall build process and its organizational impact are less common, but there has been recent progress. Suvorov et al. examined successful and failed migrations of build processes in Linux and KDE and outlined four main challenges faced by projects attempting to migrate from one build process to another [38]. McIntosh et al. looked at the effort required to maintain build systems by mining and creating models from the source control histories of software projects [31]. Phillips et al. addressed the topic in their study on integration decisions [32]. In their work, failures in the build process are an outcome of failed or poorly-planned source code integrations (also referred to as merges between branches), and they are shown to be a contributing factor to release delays.

To proactively detect these integration-related build failures, Brun et al. developed a tool to continuously build and provide desktop alerts to developers [6]. In terms of predicting build failures, Hassan and Zhang created predictive models from historic project information [15]. Similarly, Wolf et al. looked at predicting build failures through social factors, e.g., by who commented on work items in a source code repository [42]. Our work is somplementary to these studies by examining the people who would use the information and tools they propose.

Team effectiveness is a central concept in management science. In the introduction we discussed Cohen and Bailey's highly-influential analysis framework [7]. In 2008, Mathieu et al. produced a review of team effectiveness research, highlighting major work and trends from "literally hundreds of primary studies" [30]. Of these studies, the work of Levesque et al. is probably most relevant to this paper, where they examine shared mental models, e.g., knowledge structures all team member possess, in software developments teams [26]. Essentially, the authors found that divergence in mental models can lead to conflict and loss of efficiency—a scenario that can occur between build and development teams, as discussed in the Section 4.

In this paper, we address concepts common to the field of group dynamics: *communication*, *trust*, and *conflict*. These concepts also have a rich history in software engineering research, particularly in the Computer Supported Cooperative Work and CHI communities. Often, research studies will include more than one of these concepts. For example, Bos et al. examined the effects of computer-mediated communication on trust [5], which we observed as the preferred communication method between development and build teams, even when co-located. Wilson et al. also conducted a study on computer-mediated communication [41], where the authors show that computer-mediated relationships can achieve the same levels of trust as in-person communication, albeit at a slower rate.

Jarvenpaa and Leider's paper on the challenges of maintaining trust in global distributed teams [21] and Bird et al.'s discussion on the dynamics of the distributed teams in Windows Vista [4] are particularly relevant to our work, as many Microsoft build teams are globally distributed themselves, or must collaborate with development teams in different countries. Similar challenges are described in Herbsleb and Grinter's work on coordination problems in software development [17] and in Hinds and Mortensen's paper on conflicts in geographically distributed teams [20], both of which discuss how conflicts can occur when communication channels break down be-

[1] `http://www.facebook.com/Engineering`
[2] `http://google-engtools.blogspot.com/`
[3] `http://www.gentoo.org/proj/en/releng/`
[4] `http://www.eclipse.org/eclipse/platform-releng/`

tween teams.

Many studies have looked at how to manage conflict between teams. For example, Rocco discussed how in-person communication can potentially both repair and prevent trust breakdown [35]. This phenomenon is more deeply examined in Gaertner et al.'s work on reducing intergroup conflict [13], which is itself an analysis of Sherif et al.'s classic Robbers Cave study on conflict and cooperation [36]. We integrate findings from these studies into our analysis to help prevent the conflicts that can occur between builders and developers.

3. METHODOLOGY

This paper reports on three studies into build team effectiveness: interview, survey, and focus group. The studies were conducted sequentially at the Microsoft Redmond campus.

The first study, builder interviews, derived its structure from Cohen and Bailey's effectiveness framework [7]. The second study, a survey of the Microsoft build population, explored key themes discovered in the interviews. The third study, a focus group, refined and evaluated our proposed tools and practices that were designed using the analysis of the two preceding studies.

Interviews

Seven engineers (P1...P7), with build engineering experience at Microsoft ranging from 8 to 16 years, participated in our interview study. Participants were recruited through contacts in the Microsoft Office, Windows, Visual Studio, and Xbox product groups. Our intent was to capture a diverse set of experiences as these groups develop very different products and employ their own, often differing, engineering practices and policies. Moreover, four of the participants have also worked as developers, giving them a dual perspective. Table 1 shows the current product group and Microsoft build experience of the participants.

Table 1: Interview participant demographics. Experience is measured in the number of years on build teams at Microsoft. Ranges are used to help protect participant anonymity.

Identifier	Product Group	MS Exp. (yrs)
P1	Visual Studio	15–20
P2	Windows	10–15
P3	Office	10–15
P4	Visual Studio	5–10
P5	Office	10–15
P6	Office	5–10
P7	XBox	15–20

The interviews were semi-structured, audio-recorded, and 1-2 hours in length. The questions were designed to be consistent with the characteristics from Cohen and Bailey's often-referenced effectiveness framework, which allows our findings to be comparable to other team effectiveness studies in management science. The first author's three years of enterprise build experience helped ensure the questions were phrased appropriately.

The exploratory nature of this study prompted the use of grounded theory as the methodological approach [8]. The interviews were transcribed verbatim and open-coded to identify key concepts of build team effectiveness; 50 codes were organized into 12 categories, and the categories into four themes:

- **Role ambiguity**: diverse builder tasks and responsibilities and the impact on performance evaluation.
- **Knowledge sharing**: the transfer of information and experience within build teams and to development teams.
- **Intergroup dynamics**: communication, trust, and conflict between build and development teams.
- **Build failure management**: investigating build failures and the decisions around resolving the failures.

The themes are discussed at-depth in Section 4. The exception is build failure management, which has a large amount of data on build-specific tooling (e.g., source control tools) and is somewhat disparate from the human-aspects of build engineering epitomized by the other themes. Thus, we intend to present our findings on build failure management in a separate paper.

Survey

Themes from the interview study raised several questions that required a broader set of data to answer:

- What activities are builders involved in?
- What information do builders need to share?
- What problems are perceived to be in the build process?

The first and second questions were intended to complement and refine the themes of role ambiguity and knowledge sharing, respectively. The two questions had both scaled (with predefined lists generated from the interviews) and open-ended components. The third question, which was solely open-ended, was used to verify that the interviews captured the major build-related concerns. To collect this data, we engaged the build population at Microsoft through an online survey study.

Recruitment emails were sent to the 367 employees who work, in some capacity, in the build-space at Microsoft; 132 responses were received (S1...S132), a 36% response rate. Our results are thus generally representative of those working in the build-space at Microsoft, and quantitative results are accurate within $\pm 6.8\%$ with 95% confidence. Table 2 illustrates the distribution of respondents by primary working division.

Table 2: Survey respondents by primary Microsoft division.

Microsoft Division	# Respondents
Business	25
Entertainment and Devices	13
Online Services	11
Server and Tools	35
Windows and Windows Live	33
Other	15

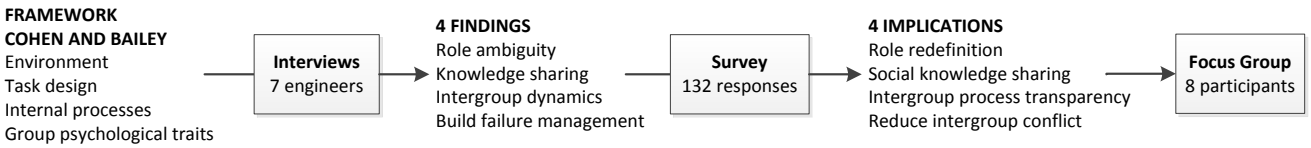

Figure 1: Summary of our research methodology.

Responses to the "build problems" question were coded in the same manner as the interview transcripts. No additional codes were identified, increasing our confidence that the interviews identified the significant build issues. The results from the survey are reported alongside the interview findings, either in tabular form or as supporting quotations, in the Section 4.

Focus Group

We synthesized the findings from the interviews and survey to preliminary design tools and practices that can be used to improve build team effectiveness. A focus group study enabled us to refine and gather feedback on the ideas in a time-efficient manner.

The focus group was intended to be *analytical*, to gather practitioner feedback and test initial feasibility, and *evaluative*, to support our interpretation of the findings [24]. The study design was based on recommended practices in sociology [25], namely, we had 8 participants, used audio-recording devices, had two researchers present (moderator and note-taker), and used short, clear, and conversational questions.

The focus group participants were selected from a pool of 64 Microsoft employees that was created from an opt-in at the end of the survey. We reduced the candidate pool to 20 by examining their responses to the open-ended survey questions, and selecting those who provided insightful answers. We then selected 8 participants (F1...F8) under the criteria of representing several divisions at Microsoft, having both build and development experience, and having worked for various amounts of time at Microsoft. Table 3 depicts the participant demographics.

Table 3: Focus group participant demographics. Experience is measured by the number of years working at Microsoft. Ranges are used to help protect participant anonymity.

Identifier	Microsoft Division	MS Exp. (yrs)
F1	Business	25–30
F2	Server and Tools	5–10
F3	Entertainment and Devices	10–15
F4	Windows and Windows Live	10–15
F5	Server and Tools	10–15
F6	Business	5–10
F7	Windows and Windows Live	5–10
F8	Windows and Windows Live	10–15

The focus group was conducted as follows: present a challenge identified in the earlier studies along with some supporting data (usually quotations from the interviews or survey); facilitate a discussion on the problem; present our potential solution to the problem through a mock-up or descriptive slide; and then continue the discussion with an emphasis on refining and evaluating the idea. The proposed tools and practices, and the results from the focus group, are discussed in Section 5.

Limitations

Efforts were made to increase the validity of our findings in the broader scope of "team effectiveness" by shaping the interviews with an established and well-accepted framework. Additionally, our studies captured the experiences of a diverse set of participants from a variety of product groups; however, the participants are still bound by the same overarching Microsoft organizational culture.

As noted in previous research (e.g., [3, 4]), there is a great amount of functional diversity between Microsoft product groups, for example, in their size, engineering practices, and release processes which mitigates some of the bias associated with conducting research at a single company. Nevertheless, there is a possibility that the challenges faced by build teams at Microsoft do not occur in other organizations, or do not occur to the same degree.

4. FINDINGS

In this section, our findings are partitioned by the themes identified in the interview study: role ambiguity, knowledge sharing, and intergroup dynamics. The themes represent factors that can influence the effectiveness of build teams. We use these findings to inform the design of tools and practices in the following section.

4.1 Role Ambiguity

Build teams "generally grow organically" (P6) in response to organizational growth, either in the number of developers or the size of the codebase, and evolving build requirements, such as supporting additional languages, hardware architectures, platforms, or SKUs. In group dynamics, build teams are classified as emergent [12] as they are a reaction to a changing environment and not pre-planned for a fixed amount of work, as are many development teams.

Emergence can cause each builder to "have a different shape" (P6)—their roles are molded to fit the changing needs of their organization. All of the interview participants noted this diversity, calling builders "generalists" (P1) or "jack-of-all-trades" (P2). For example, we found that some builders write and maintain build verification tests, while others do no testing at all.

Thus, the role "builder" has different definitions in different organizations. Furthermore, there is uncertainty around whether builders should be evaluated as "developers, testers, or project managers" (P2). For example, coordinating code flow (i.e., source code integrations) between teams is a project management task; maintaining a build system a development task; and testing falls in the domain of quality assurance.

To more broadly understand and quantify the builder role, we asked the survey respondents what frequency they perform the tasks men-

Table 4: Survey results of builder task frequency. The questions were scaled from 1–5 (a response of 1 was "never" and 5 was "very frequently"). Responses were dichotomized and the frequency percentage is the proportion of respondents that answered "4, frequently" or "5, very frequently"

Task	Examples	% Frequent
Run build operations	Start and monitor build processes; manage build failures	71%
Develop build automation	Create and maintain distributed and parallel build scripts	68%
Manage code flow	Coordinate integrations between teams; resolve conflicting changes	52%
Desktop build support	Assisting developers with their local build failures	49%
Develop desktop build tools	Change verification or check-in system development	47%
Product development	Feature development; fixing bugs in product code	30%
Hardware management	Purchase and maintain build machines; configure networks	18%
Testing	Create build verification tests; maintain test execution scripts	9%

tioned during the interviews. The task descriptions and proportion of respondents that frequently perform the tasks, derived from response dichotomization [23], are listed in Table 4. Our results confirm that builders perform as testers, and more frequently, as developers and project managers.

A particular concern among interview participants was "unrealistic expectations" (P4). Builders may be involved in different types of tasks, but will likely not perform as well as a specialist in those tasks. Moreover, we found that "the jack-of-all-trades mentality can be abused, especially on smaller teams" (P2), where builders will likely to do a variety of tasks outside of the build-space.

Thus, the "builder" role is somewhat ambiguous, both in job responsibility and evaluation. Prior research has shown that role ambiguity is negatively correlated with job satisfaction and performance [1, 39], although weakly in the latter case. We found similar results; there was no indication of decreased job performance, but there were concerns about retention. Several participants noted that dissatisfaction with "how to quantify their contributions" (P1) has led some builders to move to purely development or project management roles.

4.2 Knowledge Sharing

Build teams are not "tied to a specific feature [under development]" (P4) and thus have a "200 ft. view" of the project (P1). This view gives them a broad perspective on the different development teams' current tasks, schedules, and challenges. This perspective is unique because builders are also deeply involved with the codebase and can still "pick out all the low-level details" (P1).

The build team's perspective makes them particularly well-suited to coordinating code flow between teams and at performing the actual source code integrations. Indeed, as shown in Table 4, managing code flow is a task likely to fall under the scope of build team responsibilities.

Working with development teams across their organization, coupled with their build expertise, gives build teams context on project-wide, best practices for building. For example, how to avoid changes that cause "build problems that can be routed back to some decision that looked like a good idea to a particular team in isolation" (P1). The interviews highlighted the importance of sharing this knowledge both *intragroup*, within build teams, and *intergroup*, to the development teams.

Intragroup Knowledge Sharing

Concerns about intragroup knowledge sharing tended to involve *branching* and the speed at which knowledge is transferred. Branching is a common engineering practice [33] where development teams use "copies" of the codebase until they are ready to integrate their work. Nondeterministic build problems can propagate between branches during integrations before being identified and fixed. However, "duplicate investigations" (S27) may occur in different branches until it is discovered that certain problems have the same root cause, which may not be readily apparent.

To better understand what information builders are using during these failure investigations, we asked survey respondents how useful, and how frequently they accessed, different information sources that were discussed in the interviews. The results are displayed in Table 5.

So-called "tribal knowledge" (P3), i.e., undocumented build experiences, was indicated as one of the most useful and frequently accessed information sources. Thus, because internalized information is important to builders when managing failures, team effectiveness is influenced by how well senior builders communicate their experiences with new team members. A survey respondent described how build automation created by senior builders, intended to simplify the build process, can be a barrier to this information flow:

> "As awesome as automation is, it isolates the builder from the easy tasks the help them understand the build process...when the senior builder moves on and the difficult tasks break, the rookie is at a huge disadvantage while they try to gain tribal knowledge." (S28)

Intergroup Knowledge Sharing

When sharing knowledge across group boundaries, a major challenge is the large amount of build information to communicate:

> "If I was to put together a document that described every do and do-not of [building] source code across a 40GB codebase, you would never find the pebble of information that is pertinent to what you are trying to do at this moment." (P4)

There were no reported problems with incentives [37] for inter-

Table 5: Survey results on the usefulness and access frequency of information sources when investigating build failures. The questions were scaled from 1–5 (e.g., a response of 5 was "very useful" or "very frequently"). Responses were dichotomized and the usefulness and frequency percentages are the proportion of respondents that answered "4, useful" or "5, very useful," and "4, frequently" or "5, very frequently," respectively.

Information Source	Description	% Useful	% Frequent
Build logs	Output from build processes	92%	92%
Tribal knowledge	Undocumented team/personal experiences	89%	81%
Source control	Change history of source code files	82%	78%
New changes	List of new changes in broken builds	74%	67%
Check-in systems	Results from change verification	43%	26%
External context	Email, hallway conversations, etc.	42%	32%
Loop/rolling builds	Results from continuous compilation	40%	25%

group knowledge sharing; on the contrary, the interview participants spoke of how preventing bad practices as far "upstream" (P6) as possible, e.g., on the developer desktop, greatly reduced their workload. Rather, the concern was how to make the best practices easily discoverable.

Wikis were described as a common approach to knowledge sharing with developers; however, the general attitude was that, as noted above, there is a large amount of information and "developers were expected to pull that information" (P4) without knowing what was applicable to their current task. The lack of perceived benefit has led to the abandonment of many build wikis, a situation similar to what has been reported in earlier research [14].

An alternative to wikis is to "have information pushed to developers" (P4), where they do not have to actively seek details on best practices. Push approaches have had mixed success:

> "You need somebody saying that 'this is a best practice.' But I tried telling people about them, I tried yelling, I tried emailing...the only thing that has made a difference was putting a warning when they compile." (P1)

The participant's successful experience involved writing rules into a desktop build tool that displays warnings when a developer compiles non-conforming code. The goal is to push targeted information from the build team at relevant times. It was noted that the warning does not necessarily have to explain how to implement the best practice, it can be sufficient to simply initiate a conversation, "that is the biggest thing" (P1).

4.3 Intergroup Dynamics

How an emerged build team fits socially within its organization, and in particular, how it relates with development teams, will influence how well it operates. The term "organic" (P6) was often used to describe how these relationships formed, and there was little mention of any explicit actions to guide their growth. Our findings from both the interviews and survey, presented in this section, reveal much about the nature of the intergroup builder/developer relationship as well as the areas that can be improved to promote team effectiveness.

Communication

The "email culture" [40] within Microsoft makes computer-mediated interactions the norm. We found this culture accentuated in build teams. All interview participants preferred to communicate with developers through email, even when co-located. Many build tasks are managed entirely in a virtual-space: from the "you have broken the build" (P6) messages to the end-of-day summary and status updates, "it is all kept track in emails" (P3). Many products developed at Microsoft have globally distributed development teams—an increasingly common industry practice [16]. These global development environments have made computer-mediated communication even more likely between builders and developers.

Most participants had experienced difficulties using these computer-mediated, global communication channels [17, 5], such as accommodating cultural and language differences through text. One participant discussed how they accounted for these differences by "send[ing] emails that have been worded by project managers" so that they "are very gentle" when giving negative feedback (P6).

Conflict and Trust

A frequent point of discussion in the interviews was the builder/developer relationship. In many cases, build problems were attributed to the dynamics of the two groups: their perceptions, motivations, and interactions. The notions of trust and conflict tend to be central to these discussions.

The emergence of a build team promotes organizational efficiency by abstracting the complexity of the build process. In other words, to "present what is a very complicated system" (P2) in a simplified manner. This simplification, somewhat ironically, was often perceived to be a significant problem. Of the 97 responses to the open-ended survey question on "build problems," 27 included some variation of "developer misunderstanding of overall build engineering" (S30). As stated by an interview participant:

> "Developers have their own opinion of what build is; 'you compile and copy, that is all you do'...they do not see the whole picture." (P7)

Conflict can occur when developers fail to understand that their desktop build experience is different than the build team's experience. Builders view the codebase holistically and build for all supported platforms, hardware architectures, SKUs, and languages. This type of misunderstanding can erode trust and cause a "suspicion that builders are breaking your build" (P6) because "it worked on my machine" (S19).

There are situations, however, where conflict is not rooted in process misunderstanding, but in competing motivations. Intuitively, build and development teams are working towards the same goal—the release of their software product. While this goal is superordinate, the individual group motivations can differ; for example:

> "Developers get forced into the position where they feel they need to cut corners or cheat to avoid processes that are there for a good reason." (P4)

> "Some teams are behind so they feel a sense of urgency and cut corners...but they are taking a huge risk and can introduce a failure rate that slows everyone down." (P3)

These experiences describe situations where developers, due to time constraints, attempt to have a greater code velocity at the expense of confidence that the build will not fail—the overhead of fully vetting changes for build-stability "outweighs the perceived benefit" (P4). In these situations, developers believe that it will take less time to fix the build if it happens to break, than to go through a full change verification prior to submitting their changes. However, a build team is likely more concerned with build reliability for the entire organization, and when it feels a subset of the organization is not appropriately verifying its changes, it can be a source of frustration. The problem is not with developers needing more agility, but builders feeling not "in the loop" (P7) when the requirements change and are not communicated.

Peer Monitoring

Although each interview participant could recall conflicts between builders and developers, it was generally felt that "most people feel bad when they break the build and they do not want to do it again" (P6). Yet, for repeat offenders, a question that generated a lot of discussion was how they can be "incentivized [*sic*] to not do it again," because "you do not want to let them off the hook" (P6).

Historical approaches to build-break accountability varied from the humourous, such as wearing a funny hat for the day, to the somber, reporting to "ship room" (P7) and account for the break in front of one's peers. Today, accountability tends to be computer-mediated, e.g., broadcasts to email distribution lists describing what changes broke the build. Recent work has called these types of actions *peer monitoring* [27], where coworkers attempt to deter disruptive behaviour. The term one participant used was "public shaming" (P4).

The feeling was that public shaming does promote caution as it makes people "petrified of breaking the build" (P7); however, there was no evidence that it reduced the overall number of build failures. On the contrary, some felt that the "same amount of failures still occur" (P6), which indicates that careless submissions, while frustrating, may not be very common.

Nevertheless, all participants stated that computer-mediated public shaming is something they have used, and will likely use in the future—it was described as "a valuable tool...but you need to know when to use it" (P1). The variance stemmed from frequency: use it often, occasionally, or only as a "last resort for the most severe offenses" (P4).

5. IMPLICATIONS FOR ORGANIZATIONS

In the previous section, we presented our findings on the characteristics and dynamics of build teams that can influence their effectiveness. In this section, we synthesize the findings into tools and practices that can potentially be used to improve build team effectiveness. As the challenges we have described are mostly social, we leverage theories from group dynamics and organization science, some of which have decades of supporting research, in our designs.

While the ideas we present here are grounded in qualitative data and established theories, there is a risk that they are impractical, ineffective, or otherwise have little value to practitioners. To mitigate this risk, we conducted a focus group study with Microsoft employees to evaluate, refine, and assess the feasibility and value of the ideas.

This section is structured around our four ideas: role redefinition, social knowledge sharing, intergroup process transparency, and reducing intergroup conflict. After each idea is described, the feedback from the focus group is presented.

5.1 Role Redefinition

Use empirical data to reduce role ambiguity.

Our findings indicate that the role of "builder" is ambiguous and the definition can vary between organizations. There is some evidence that this ambiguity can lead to job dissatisfaction and movement out of the build discipline. To prevent the loss of talented builders, our idea is to redefine the role with clear responsibilities, as recommended in other work [21, 39], but to do so based on empirical data.

Our survey data in Table 4 shows the general shape of the build role after years of organic growth. The two most frequent tasks, build operations and build automation, are different in that operations is closely aligned with project management, while automation is aligned with development. To reduce ambiguity, we propose a bucketing approach where build operations and build automation are the buckets, and the remaining tasks are sorted based on whether they primarily involve management or development duties.

Based on the bucketing, the builder role should be partitioned into the roles of build *operator* and build *engineer*. Build operators will manage the build operations and team coordination, while build engineers will create and maintain build automation and tools as specialized developers. This measured approach to role definition accounts for an organization's actual needs by broadly examining how an ambiguous role has evolved over time.

Feedback: The focus group confirmed our findings of role ambiguity by describing a builder as "whatever it needs to be" (F5) and that "you cannot have one definition for a place as diverse as Microsoft" (F6). Support for redefining the builder role along our proposed boundaries was strong, as there was agreement that build engineering and build operations are "separate disciplines" (F8).

It was felt that, under our proposal, build engineers could be more fairly evaluated against developers as they would have a clear focus on development tasks. A limitation, however, was that build operators might not receive the same benefit:

> "There are huge amounts of obfuscation in the build process. If you are good, no one ever knows about it." (F1)

The concern is that build operators would lose visibility if they were no longer involved in development tasks, so they should have some avenue to report their achievements.

5.2 Social Knowledge Sharing

Promote intergroup communication with social build tools.

Our interview participants discussed several challenges with sharing knowledge across group boundaries; in particular, the large amount data to share and the discoverability of relevant information. A solution from the Microsoft Visual Studio build team had the most enthusiastic support, where custom build rules are verified at desktop compilation; for example, when a developer creates a dependency between two projects that, while technically correct, crosses an architectural boundary that could break parallelism in the build system. In this case, a warning or error message is displayed to the developer.

Our idea is to enhance this solution by incorporating a social component. Majchrzak et al.'s findings also show the value of transferring data "fragments," but they additionally advocate for continual intergroup engagement [29]. Applying their findings to our context, each build best practice will have an owner and their contact information will be included in the corresponding error message. Developers will then know exactly who to contact on the build team for guidance, and will be encouraged, or possibly required, to do so.

Feedback: The focus group, before being presented with our idea, discussed how they have written documents about build best practices, but "developers do not know what to search for in them" (F2), supporting our earlier findings on intergroup knowledge sharing.

The group suggested two refinements to improve the likelihood of our idea being adopted. First, it was noted that "tribal knowledge" can vary "between branches" (F4) and not just at the product level. In other words, certain rules will apply to some branches, but not to others (e.g., branches that contain user interface work vs. branches for database changes). The knowledge sharing tool will need to be aware of what branch the developer is working in.

Second, it was noted that the tool will probably not be able to detect all violated best practices:

> "You can never write a system that will work 100% of the time. An automated system cannot look at a bad build configuration file and say 'oh, I think this is what you meant to do."' (F3)

When automatically verifying best practices is not possible, it may be sufficient to examine which files the developer has changed, and display warnings if those files are related to particular best practices. Moreover, builder contact information will still be displayed for when the developer is unsure if they are violating the best practice in question.

5.3 Intergroup Process Transparency

Do not make the build process a black box.

We found that some build failures are perceived to be caused by developers misunderstanding the build process, or not understanding the benefit of following the process and they avoid it or cut-corners. The problem appears to be the abstraction of build system complexity away from development teams. If the build team provides a view of the build process that is too simple, trust can erode between the two groups. Our idea is to balance abstraction with process transparency.

This idea is grounded in Pirson and Malhotra's analysis, where perceptions of transparency were the only significant predictors of trust for deeply interdependent relationships [34], which is the case for build and development teams. We propose providing developers with a desktop tool that clearly describes build process requirements based on changes they are currently making. For example, what parts of the project they must build, what tests they need to run, or whether they should perform any special verification. But most importantly, the tool will clearly describe why they are being asked to perform each task as well as the benefits.

The goal is to give developers a glimpse into the "200 ft. view" held by the build team and allow them to see that their time is not being wasted with ineffectual overhead, and that the build team has their best interests in mind.

Feedback: Of our four ideas, process transparency received the most negative reaction. It was felt that the idea "would work" (F4) by preventing some failures, but there were concerns about underlying concept of transparency:

> "They will care less about when things break—they will say, 'I followed your process, why did it not prevent me from breaking the build?" (F3)

There was a reluctance among the participants to lower group boundaries and offer process transparency out of concern that the tool would be used against them. In other words, that the tool would lower accountability for build breaks and require them to establish a perfect set of build process requirements, which would not be possible.

To implement this tool, it is clear that the descriptions of the tasks, justifications, and benefits should be carefully worded. For example, it should be made clear that the requirements are guidelines provided by the build team to decrease the likelihood of build breaks, but they are not complete and do not remove personal accountability.

5.4 Reduce Intergroup Conflict

Meet early to establish trust before conflicts occur.

The categorization of build and development teams is evident by the frequent use of the term "they" when referring to developers [13], which can be observed in the quotations we have throughout this paper. In the Section 4, we identified several situations where conflict can occur between build and development teams, such as impeding goals and with public shaming. We have also shown these conflicts are similar to those observed in other studies, often in

fields outside of software engineering, such as group dynamics and organization science. Our idea is to use results from these studies, altered to fit the build engineering context, to prevent or reduce the likelihood of conflicts between builders and developers.

Rocco found that early face-to-face contact can establish trust in computer-mediated relationships [35]. Similarly, we propose that builders and developers meet twice, in-person, at the beginning of a product release cycle.

The first meeting (i.e., the "hand-shake" meeting) should not focus on upcoming work, but on personal interests to promote decategorization [13]. The second meeting, held shortly afterward, will revolve around mutual intergroup differentiation—the appreciation of their differences [13]. The build team can talk about changes to the build process, which also aids transparency, and the development teams can talk about their upcoming features and general time-lines.

Our idea is relatively simple, but there is a large body of research supporting these methods of reducing intergroup conflict. In terms of improving build team effectiveness, ensuring a healthy relationship between build and development teams facilitates the smooth operation of interdependent tasks.

Feedback: Support for the idea was strong among the focus group participants. One participant shared a similar practice that they have found to be effective:

> "Every time I join a new organization, I meet with all of the development leads and talk about best practices...'if you do this then I'll provide you good builds in a timely manner.' You have to be proactive." (F8)

Our approach differs in that it has the individual builders and developers meet, and not just the team leads. Although, the scale of our idea scale was questioned:

> "Shaking hands...is a little unreasonable. In Windows, 5000 people cannot meet with 30 builders." (F7)

However, we do not see the approach as all-or-nothing. The build team can selectively meet with development teams; in particular, those that are planning changes that could be disruptive and potentially cause intergroup conflict.

6. DISCUSSION

To many practitioners, software build is an uninteresting step in the development process. Academically, build is a niche topic when compared to understanding, writing, and testing code. To our surprise, however, research into the understudied "build team" uncovered rich experiences and useful insight that can be applied outside the context of our work.

We found that build teams are created in response to project growth and mounting build inefficiency. In other words, they are not planned (and in some cases, are unwanted), but nevertheless emerge and grow organically. An interesting facet is that these ad hoc groups are given sole responsibility of a critical resource needed by much larger development teams.

A particularly noteworthy finding is process transparency. Participants in both the interviews and survey felt that exposing elements of the build process would reduce developer frustration and potential conflict. However, when we proposed our simple transparency-promoting tool to the focus group, it evoked strong negative reactions. When confronted with actually giving up some control of the important resource, the small group fiercely defended their domain. Anticipating this reaction can help ease the implementation of new tools and practices for these types of groups.

A critique of our ideas may be that they are too simplistic (or for some, obvious). But in practice, we found many product groups and practitioners have not attempted, or are not aware, of these types of solutions. Academically, our findings provide validation of existing social theories that have been developed through many different contexts over the last 60 years; for example, with adolescents [36], between different races [18], and in the workplace [11]. However, we are applying these theories in a software development context—a multidisciplinary approach not often seen in software engineering literature.

Simplicity also facilitates future work in the build space. Changing a build process within a company is non-trivial and can impact a very large number of developers [38] (e.g., in Windows, there are over 2000 developers who would be affected) and implementation problems can have serious financial repercussions. Research ideas that are easy to implement and carry minimal risk are much more likely to be adopted in such environments. In this regard, our use of a focus group serves to minimize risk in future work by providing initial evaluation by subject matter experts and current practitioners.

7. SUMMARY

The goal of our research was to understand and propose solutions to issues faced by software build teams. Success in this area can have a positive impact on an organization's ability to release software on-time and on-budget.

We used an established team effectiveness framework from management science to structure our initial explorations, where we found that many impediments to build team effectiveness are social, brought about by their relationships with interdependent teams.

Further, we demonstrated how theories from the social sciences can inform the design of tools and practices for software engineering. We proposed and solicited feedback on four such ideas to address the social challenges that can impede build team effectiveness. We also discussed how our approach can benefit other groups with interdependent relationships both inside and outside of software organizations.

8. ACKNOWLEDGMENTS

This work was done during a summer internship at Microsoft Research and was sponsored by Microsoft Engineering Excellence. We would like to thank our study participants for sharing their time and experience, and to Jonathan Sillito and Jeff Huang for their valuable feedback on this paper.

9. REFERENCES

[1] D. J. Abramis. Work role ambiguity, job satisfaction, and job performance: meta-analyses and review. *Psychological Reports*, 75(3f):1411–1433, 1994.

[2] B. Adams, C. Bird, F. Khomh, and K. Moir. 1st international workshop on release engineering (releng 2013). In *Proceedings of the 35th International Conference on Software Engineering*. IEEE / ACM, 2013.

[3] C. Bird, B. Murphy, N. Nagappan, and T. Zimmermann. Empirical software engineering at microsoft research. In *Proceedings of the 2011 ACM Conference on Computer Supported Cooperative Work '11*, pages 143–150. ACM, 2011.

[4] C. Bird, N. Nagappan, P. Devanbu, H. Gall, and B. Murphy. Does distributed development affect software quality?: an empirical case study of windows vista. *Commununications of the ACM*, 52(8):85–93, 2009.

[5] N. Bos, J. Olson, D. Gergle, G. Olson, and Z. Wright. Effects of four computer-mediated communications channels on trust development. In *Proceedings of the ACM CHI 2002 Conference on Human Factors in Computing Systems (CHI)*, pages 135–140. ACM, 2002.

[6] Y. Brun, R. Holmes, M. D. Ernst, and D. Notkin. Proactive detection of collaboration conflicts. In *Proceedings of the 2011 joint meeting of the European Software Engineering Conference and SigSoft Symposium on Foundations of Software Engineering*, pages 168–178. ACM, 2011.

[7] S. G. Cohen and D. E. Bailey. What makes teams work: Group effectiveness research from the shop floor to the executive suite. *Journal of Management*, 23(3):239–290, 1997.

[8] J. Corbin and A. Strauss. *Basics of Qualitative Research*. SAGE Publications, 3rd edition, 2008.

[9] M. A. Cusumano and R. W. Selby. How microsoft builds software. *Communication of the ACM*, 40(6):53–61, 1997.

[10] A. Dearle. Software deployment, past, present and future. In *Future of Software Engineering*, pages 269–284. IEEE Computer Society, 2007.

[11] C. DeDreu, C. De Dreu, and M. Gelfand. *The Psychology of Conflict and Conflict Management Organizations*. Taylor & Francis, 2007.

[12] D. Forsyth. *Group Dynamics*. Cengage Learning, 2009.

[13] S. L. Gaertner, J. F. Dovidio, B. S. Banker, M. Houlette, K. M. Johnson, and E. A. McGlynn. Reducing intergroup conflict: From superordinate goals to decategorization, recategorization, and mutual differentiation. *Group Dynamics: Theory, Research, and Practice*, 4(1):98–114, 2000.

[14] J. Grudin and E. S. Poole. Wikis at work: success factors and challenges for sustainability of enterprise wikis. In *Proceedings of the 2010 International Symposium on Wikis and Open Collaboration (WikiSym)*, pages 5:1–5:8. ACM, 2010.

[15] A. E. Hassan and K. Zhang. Using decision trees to predict the certification result of a build. In *Proceedings of the 2006 IEEE/ACM Conference on Automated Software Engineering*. IEEE Computer Society, 2006.

[16] J. D. Herbsleb. Global software engineering: The future of socio-technical coordination. In *Future of Software Engineering*, pages 188–198. IEEE Computer Society, 2007.

[17] J. D. Herbsleb and R. E. Grinter. Splitting the organization and integrating the code: Conway's law revisited. In *Proceedings of the International Conference on Software Engineering*, pages 85–95. ACM, 1999.

[18] M. Hewstone and R. Brown. *Contact and conflict in intergroup encounters*. Basil Blackwell, 1986.

[19] J. Highsmith and A. Cockburn. Agile software development: the business of innovation. *Computer*, 34(9):120 –127, 2001.

[20] P. J. Hinds and M. Mortensen. Understanding conflict in geographically distributed teams: The moderating effects of shared identity, shared context, and spontaneous communication. *Organizational Science*, 16(3):290–307, 2005.

[21] S. L. Jarvenpaa and D. E. Leidner. Communication and trust in global virtual teams. *Journal of Computer-Mediated Communication*, 3(4), 1998.

[22] E.-A. Karlsson, L.-G. Andersson, and P. Leion. Daily build and feature development in large distributed projects. In *Proceedings of the 2000 International Conference on Software Engineering*, pages 649–658. ACM, 2000.

[23] B. Kitchenham and S. Pfleeger. Personal opinion surveys. In F. Shull, J. Singer, and D. SjÃÿberg, editors, *Guide to Advanced Empirical Software Engineering*, pages 63–92. Springer London, 2008.

[24] J. Kontio, L. Lehtola, and J. Bragge. Using the focus group method in software engineering: Obtaining practitioner and user experiences. In *Proceedings of the International Symposium on Empirical Software Engineering*. IEEE Computer Society, 2004.

[25] R. Krueger and M. Casey. *Focus Groups: A Practical Guide for Applied Research*. Sage, 2009.

[26] L. L. Levesque, J. M. Wilson, and D. R. Wholey. Cognitive divergence and shared mental models in software development project teams. *Journal of Organizational Behavior*, 22(2):135–144, 2001.

[27] M. L. Loughry and H. L. Tosi. Performance implications of peer monitoring. *Organization Science*, 19(6):876–890, 2008.

[28] A. MacCormack. How internet companies build software. *MIT Sloan Management Review*, 42(2):75–85, 2001.

[29] A. Majchrzak and P. H. More. Transcending knowledge differences in cross-functional teams. *Organization Science*, 23(4):951–970, 2012.

[30] J. Mathieu, M. T. Maynard, T. Rapp, and L. Gilson. Team effectiveness 1997-2007: A review of recent advancements and a glimpse into the future. *Journal of Management*, 34(3):410–476, June 2008.

[31] S. McIntosh, B. Adams, T. H. Nguyen, Y. Kamei, and A. E. Hassan. An empirical study of build maintenance effort. In *Proceedings of the 2011 International Conference on Software Engineering*, pages 141–150. ACM, 2011.

[32] S. Phillips, G. Ruhe, and J. Sillito. Information needs for integration decisions in the release process of large-scale parallel development. In *Proceedings of the 2012 ACM Conference on Computer Supported Cooperative Work*, pages 1371–1380. ACM, 2012.

[33] S. Phillips, J. Sillito, and R. Walker. Branching and merging: an investigation into current version control practices. In *Proceedings of the International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, pages 9–15. ACM, 2011.

[34] M. Pirson and D. Malhotra. Foundations of organizational trust: What matters to different stakeholders? *Organization*

Science, 22(4):1087–1104, 2011.

[35] E. Rocco. Trust breaks down in electronic contexts but can be repaired by some initial face-to-face contact. In *Proceedings of the ACM CHI 1998 Conference on Human Factors in Computing Systems (CHI)*, pages 496–502. ACM Press/Addison-Wesley Publishing Co., 1998.

[36] M. Sherif. *The Robbers Cave Experiment: Intergroup Conflict and Cooperation*. Wesleyan University Press, 1988.

[37] E. Siemsen, S. Balasubramanian, and A. V. Roth. Incentives that induce task-related effort, helping, and knowledge sharing in workgroups. *Management Science*, 53(10):1533–1550, 2007.

[38] R. Suvorov, M. Nagappan, A. E. Hassan, Y. Zou, and B. Adams. An empirical study of build system migrations in practice: Case studies on kde and the linux kernel. In *Proceedings of the International Conference on Software Maintenance*, pages 160–169, 2012.

[39] T. C. Tubre and J. M. Collins. Jackson and schuler (1985) revisited: A meta-analysis of the relationships between role ambiguity, role conflict, and job performance. *Journal of Management*, 26(1):155–169, 2000.

[40] G. D. Venolia, L. Dabbish, J. Cadiz, and A. Gupta. Supporting email workflow. Technical Report MSR-TR-2001-88, Microsoft Research, 2001.

[41] J. M. Wilson, S. G. Straus, and B. McEvily. All in due time: The development of trust in computer-mediated and face-to-face teams. *Organizational Behavior and Human Decision Processes*, 99(1):16 –33, 2006.

[42] T. Wolf, A. Schroter, D. Damian, and T. Nguyen. Predicting build failures using social network analysis on developer communication. In *Proceedings of the 2009 International Conference on Software Engineering '09*, pages 1–11. IEEE Computer Society, 2009.