2015 12th Working Conference on Mining Software Repositories

Lessons Learned from Building and Deploying a Code Review Analytics Platform

Christian Bird, Trevor Carnahan, Michaela Greiler
Microsoft, Redmond, WA, USA
{cbird, trevorc, mgreiler}@microsoft.com

***Abstract*— Tool-based code review is growing in popularity and has become a standard part of the development process at Microsoft. Adoption of these tools makes it possible to mine data from code reviews and provide access to it. In this paper, we present an experience report for CodeFlow Analytics, a system that collects code review data, generates metrics from this data, and provides a number of ways for development teams to access the metrics and data. We discuss the design, design decisions and challenges that we encountered when building CodeFlow Analytics. We contacted teams that used CodeFlow Analytics over the past two years and discuss what prompted them to use CodeFlow Analytics, how they have used it, and what the impact has been. Further, we survey research that has been enabled by using the CodeFlow Analytics platform. We provide a series of lessons learned from this experience to help others embarking on a task of building an analytics platform in an enterprise setting.**

I. Introduction

Code Review is an important practice at Microsoft and in other companies and open source projects [1] [2], as it has been shown to improve software quality, increase awareness, and disseminate knowledge [3] [4]. While there has been active research leveraging the data that results from development tasks such as developing and checking in source code changes [5], executing builds [6], and fixing defects [7], until the past few years [3] [8] [9] [10] code review remained a software engineering task that was largely unexplored from a mining perspective. Similar to these other software development tasks, tool-based code review leaves behind traces of activity. As the MSR community has demonstrated, activity traces provide an opportunity for data to be mined and analyzed, enabling teams to measure themselves and derive insight [11].

Microsoft has developed an in-house code review tool, CodeFlow, which has grown in popularity to become the predominant code review tool in all product groups. While CodeFlow provides a mechanism to examine data about an individual review, there is no convenient way to mine and analyze data from code reviews. As development teams at Microsoft have sought to become more data driven, there have been repeated requests for the ability to easily aggregate, analyze, and monitor data about their code review process. Two years ago, in an effort to address this need, we developed and deployed a real-time code review data gathering and metric computing platform named *CodeFlow Analytics* (CFA). In the time since then, teams across Microsoft have discovered, used, and begun to rely on CFA as a source of information and insight. In this paper, we describe our experiences with building, deploying, and supporting CodeFlow Analytics. Some aspects of the success of CFA have come as a result of intentional design decisions early on, while others were more the result of luck. At the same time, we also made mistakes along the way and encountered unanticipated challenges. We have continually adapted CFA as reviewing tools have evolved, as resources at Microsoft have changed, and as teams have requested additional functionality.

We make the following contributions in this paper:

1. We describe the design of CodeFlow Analytics, including the types of data that it collects, the metrics it computes, and the ways by which users can access it.
2. We report insights such as adoption patterns, frequent usage scenarios, and challenges during adoption from interviews with nine development teams across Microsoft that have used CodeFlow Analytics.
3. We discuss the ways that CodeFlow Analytics has supported research in the code review space.

Our hope is that others can learn from our experience as they attempt to build, deploy, and support software development data platforms in their own organizations. To that end, herein we have highlighted lessons learned from CFA that we believe are useful outside of our own specific context. We have also shared the reasons and ways that development teams have used CFA, as we maintain that they are reflective of how teams think about using data from their development processes. We stress that this is a practice paper, not a research paper. As such, we will not be presenting novel algorithms. Nor will we provide low level concrete details about our system, many of which are specific to Microsoft and would doubtless be of little use to most readers. More details regarding data collection and storage of software engineering data at Microsoft can be found in the work of Czerwonka et al. [12].

II. Description of CodeFlow Analytics

Most teams at Microsoft have adopted code review as a standard practice to manage software quality and aid team awareness. The primary tool that teams use to conduct reviews is CodeFlow, an in-house developed collaborative code review tool that allows users to directly annotate source code and interact with review participants in a live chat model [13]. The functionality of CodeFlow is similar to other review tools such as Google's Mondrian [14], Facebook's Phabricator [15] or open-source Gerrit [16]. Developers who want their code to be reviewed create a package with the source code changes, select

978-0-7695-5594-2/15 $31.00 © 2015 IEEE
DOI 10.1109/MSR.2015.25

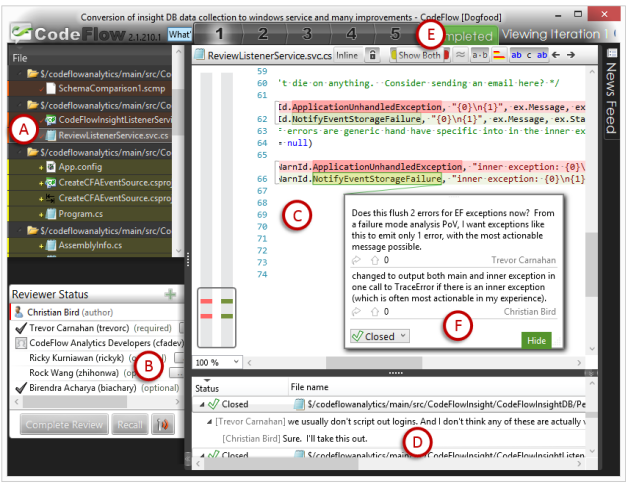

Fig. 1 Screenshot of CodeFlow Review Tool

reviewers, provide a description of the change to be reviewed, and submit the review to the CodeFlow service. CodeFlow then notifies the selected reviewers about the incoming task via email. The developer that creates the code change and submits it for review is the review *Author,* and those who receive the review, provide feedback on the change, and can sign off, are termed the *Reviewers.*

To understand the data that CFA collects, it is first important to understand how CodeFlow is used. Once reviewers open a CodeFlow review, they interact with it via a single desktop window (Figure 1). On the top left **(A)**, they see the list of files changed in the code change submitted for review, plus a "description.txt" file, which contains a textual explanation of the change, written by the author. On bottom left, CodeFlow shows the list of reviewers and their status **(B)**. In this example we see that Christian is the review author and Trevor, Ricky, Rock, and Birendra are the reviewers ("CodeFlow Analytics Developers" is a mailing list that the review is sent to, as well). Trevor and Birendra have signed off on the changes while Ricky and Rock have yet to do so. CodeFlow's main view **(C)** shows the differences in the currently selected file. Both the reviewers and the author can highlight portions of the code and add comments inline **(F)**. These comments can start threads of discussion and are the interaction points for the people involved in the review. Each user viewing the same review in CodeFlow sees events as they happen. Thus, if an author and reviewer are working on the review at the same time, the communication is synchronous and comment threads act similar to instant messaging. The comments are persisted so that if they work at different times, the communication becomes asynchronous. The bottom right pane **(D)** shows the summary of all the comments in the review along with their status (discussions begin in an "active" state and can be set to "resolved," "won't fix," "pending," or "closed," similar to bug reports). The author can make changes based on reviewers' feedback and submit an updated change, called an *iteration* in CodeFlow parlance. In the example above, there are five iterations of the change which can be viewed by clicking on the tabs labelled "1" through "5" **(E)**. Once the reviewers are confident in the change and sign off, the author marks the review as completed and typically checks his or her change into the project's source code repository.

CodeFlow's simplicity, low barrier for feedback, and flexible support for all of Microsoft's disparate engineering systems (it currently supports Git, Team Foundation Server, Perforce, and an internal source code management system) has driven its adoption since its inception five years ago. In aggregate, CodeFlow is used by engineers in every division to perform 100k to 150k code reviews every month, and 4.9 million code reviews to date spanning every shipping product at Microsoft, as well as a host of internal projects.

While CodeFlow has achieved widespread use, convenient access to the raw data to drive deep understanding and aggregate data to compute and track team, project, and organizational metrics around the code review process was missing. The data was stored in xml files, one per review, on disk behind the middle-tier services. The only access pattern was to retrieve a single code review at a time.

CodeFlow Analytics addresses this problem by continually monitoring activity on reviews, storing data about each review when it becomes available, processing this data into metrics and dimensions (described later), and making it easily available to teams. The architecture for CodeFlow Analytics is a Kimball-styled data warehouse [17] using business intelligence tools from Microsoft SQL Server running on Microsoft Azure (Microsoft's cloud computing platform [18]). For data acquisition, CFA uses a Windows service to poll the central CodeFlow server on a regular basis. We have found that activity on a review occurs in bursts (as reviewers open a review and provide feedback throughout the change under review) and thus the service polls for a list of the ID's of added or modified reviews every ten minutes, but waits to gather the data for a review until it has remained dormant for at least ten minutes. This design decision cuts down the number of requests for CodeFlow data from the server and the number of updates to our database by over 40%, a large performance boost. This raw review data is stored directly into a simple relational database. Every two hours, it is processed into a dimensional model, or star schema [17], via a T-SQL based "Extract, Transform, and Load" (ETL) process [17], and then processed into an Analysis Services tabular cube [19], an in-memory database that supports both traditional relational data storage (where raw review data is stored) as well as a dimensional model. A dimensional model consists of facts (computed metrics) and dimensions, which provide the ability to slice or filter the facts. As a concrete example, one might be interested in the average number of files in a review for reviews conducted in 2014 in Windows. In this case, 2014 is a filter in the time dimension and "Windows" is a filter in the product dimension, while average number of files is a computed fact based on those dimensions. Using such a tabular model allows CFA to be as responsive as we need, as an on-disk relational model would be prohibitively expensive in terms of time (CFA not only houses a large amount of data, but must be able to serve requests from many teams across the company concurrently).

Entities in the raw database along with their attributes include: aspects of the code review (e.g., project, author, description), reviewers (e.g., their current status, when they were as-

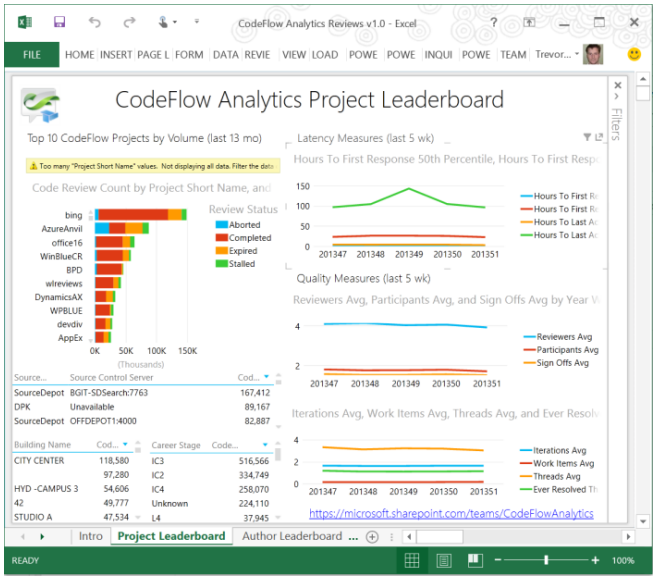

Fig. 2 Excel Power View Charts generated from CFA

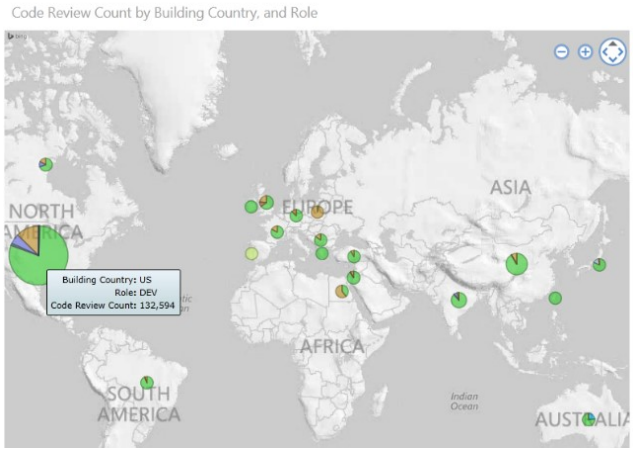

Fig. 3 Power Q&A showing reviews completed by country and role. Area of pie chart proportional to review count, green portion is by developers, orange is by testers.

signed), the files in the review (e.g. file path, repository location), feedback threads and comments (e.g., content of the comments, who made them and when, where in each file the comment is located), and related work items (references to check-ins, tasks, and defect database entries). We also track when and how each entity changes. For instance, the database may show that a particular review currently has the status "completed," but it also contains data regarding prior states, so that one can determine that its previous status was "active" and find the specific time that the status changed.

In the relational data warehouse, the data in the raw database is merged with data from the Microsoft employee database, allowing analysis by organizational hierarchy, role, or even geography.

The output of processing is a set of fact tables containing metrics and dimension tables, which in turn contain attributes useful for filtering and slicing the metrics. Examples of the metrics include: time to first response, time to sign-off, time to completion, number of threads by final state, number of comments, number of iterations, number of files, number of reviewers, number of sign-offs, as well as variants of these that have been requested by teams. Examples of dimensions are projects, months, and countries. Users can select the facts that they are interested in, filter and slice by chosen dimensions, and then aggregate in various ways. For example, a user may want to see the average number of sign-offs in his or her project per month. In this case, specifying the project dimension will filter reviews to the project of interest, while slicing along the month dimension yields a time series. The aggregation used is "average" on the fact "sign-offs." As another example, a developer might look at the 75th percentile number of reviewers in his team. In total, CFA provides over 200 facts and dimensions. We provide the median, average, and percentiles of the metrics to characterize distributions and identify outliers.

Data consumption scenarios are a point of emphasis for CFA. One goal is to optimize the "time to insight," or the time between having a curious question about code reviews and getting an empirical answer from the data. Another goal was to broadly enable many ways to experience the data. We intentionally did not create our own user interface for CFA, instead providing many ways to access the data so that customers could use the data how they see fit and leverage it in their existing tools

For the **casual user** with no material data experience who wants quick insights into his or her team's code review process, CFA provides Excel templates that query the cube and expose standard metrics. Figure 2 shows such a template that displays various metrics, including review outcome by project (top left), average hours to first and last activity and signoff by week (top right), participation (middle right), and feedback activity (bottom right).

CFA also allows **casual users** who are curious and want to ask ad-hoc questions to use Power Q&A [20]. Power Q&A enables users to ask natural language queries about data and presents the results in intuitive visualizations. As a simple example, entering the query "show the code review count by country by role in 2014 in Bing" results in the map visualization shown in Figure 3. The experience is interactive and allows users to quickly explore the dataset and answer questions. As another example Figure 4 shows how easy it is to inquire about the size of the review (in terms of files and iterations), the feedback received, and whether those metrics are dependent on role.

The natural language capabilities did require additional work to provide natural language synonyms for raw data, metrics, and dimensions to help the query processor, but this was a one-time cost for us that has been beneficial to users.

More advanced consumption experiences are available to the **analyst** users who want to dive deeper in the data to derive new insights and metrics.

The two primary data sources made available are the Analysis Services Tabular cube (facts and dimensions) and the SQL Server relational data warehouse (raw review data). Making curated data available in these two formats gives the analyst users a full spectrum of query and analysis tools that analysts are likely

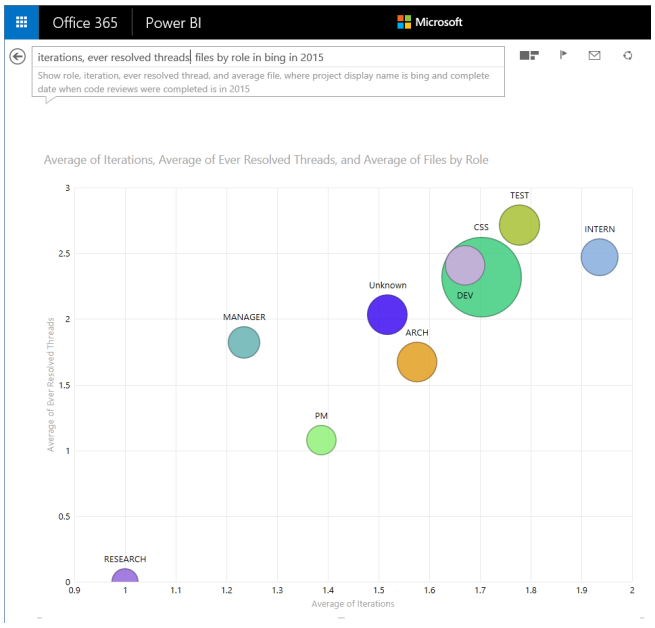

Fig. 4 Power Q&A showing average number of resolved threads of discussion (y-axis), average number of iterations (x-axis), average number of files per review (area of circle) by author role (color and label of circle) in Bing in 2015.

to already be comfortable with, such as Excel, Power BI, Tableau, SQL Server Management Studio, and R, as well as the ability to write custom code in any language that has SQL drivers available (e.g. C#, Java, or Python).

Another query interface developed for **developers** is a REST web API over the Analysis Services Tabular Model. The REST API makes it simple to include code review data into virtually any application with a simple JavaScript or web call. This has enabled developers to build web sites with dashboards, incorporate data in issue trackers, and incorporate review data into tools such as Visual Studio.

The lingua franca for data is SQL for data hounds and web APIs for app developers. Those less experienced with data analysis or wanting to easily explore data should have easy entry points. These should all be available from the beginning.

A. Challenges

There are a number of challenges that CFA has had that we have had to overcome. Here are some, along with our solutions.

Branch agnostic data collection and analysis.

Several version control systems are used within Microsoft. Most of them, with the exception of Git, are not branch agnostic. This means that the branch path is reported as a part of the file path, similar to the way branches are part of the directory structure in subversion. For example, consider a file `foo.c` in directory *bar*. Then the path of this file would be `/bar/foo.c`. Now, let us assume this file is part of the code base currently available in two branches, `main` and `release`. Since CodeFlow records the full path of each file submitted in a review, a review with changes to `/bar/foo.c` on either of these branches would contain either `/main/bar/foo.c` or `/release/bar/foo.c`. Even though these are in fact separate files, logically we might want to recognize them as one "entity" that is edited in two different branches. This example is trivial, but in practice, the portion of the path that is part of the branch can be arbitrarily deep. For example, the creation of a branch `rc_1` from the `release` branch would result in the creation of the file `/release/rc_1/bar/foo.c`. Branching systems of such large-scale software systems as Microsoft Windows or Microsoft Office tend to be extremely complex (sometimes up to seven levels deep) to allow parallel development by thousands of engineers. In theory, is it possible to obtain the list of all branches for all projects that use CodeFlow, which would make solving this problem fairly straightforward by simply comparing each path to all branches and looking for the longest common prefix.

In practice, the number of projects and permissions issues make such a solution impractical. Therefore, we had to implement heuristics that can take care of extracting the branch path from the file path. As we know that the complete path (i.e., branch path and file path) always starts with the branch path, we designed a heuristic to identify the branch portion of the path. If we do not extract enough of the path into the branch portion (e.g., if we split the aforementioned path into `/release` for branch path and `/rc_1/bar/foo.c` for the file path), we run into the problem that the same file in different branches will be seen as several distinct files. On the other hand, if we remove too many segments from the complete path (e.g., if we split the above path into /`release/rc_1/bar` for the branch portion and `/foo.c` as the file path) then two files that should be treated separately could be conflated. We did a thorough analysis of the false positive and false negatives rates for several segment length heuristics and settled for an approach that increases the file path with the length of the complete path. This is necessary because the chance of having distinct files with the same file path parts increases the deeper the directory structure is. We try to ensure a minimal length of two segments for the file path and a minimal branch path of one segment, given that the complete path is long enough. This simple approach outperformed many more complex approaches, such as finding common prefixes of branch paths. Our validation against multiple product codebases showed that we correctly split over 97% of the paths.

Linking multiple data sources.

Even though CFA provides a rich set of data to explore code review behavior, we quickly found that users wanted to link code review data with other artefacts created during software development. In particular, users wanted to "link" code reviews to the actual checkins of the code change in the software repository. Such linkages allow users to track, for example, how long it took from submitting a code change for review to the final checkin in the code base; a metric frequently asked for by development teams using code review.

As reviews and checkins happen in completely different systems, there is no easy way to find this relationship. We developed a machine learning model using logistic regression that uses features such as the proportion of file paths that a review and checkin have in common, the identities of the review author

and checkin author, the difference in time between code review signoff date and checkin date, and the textual similarity between review description and the commit message. Our model achieves precision and recall values above 95%, levels high enough that we have confidence making the review-check in linkage data available to developers.

III. Use of CodeFlow Analytics by Product Teams

As CFA has been available for two years, a number of teams have begun to use it. In an effort to understand why and how they use it, what the impact has been, and how we can improve CFA, we contacted teams to ask them about their experiences.

A. Methodology for interviews

When CFA was initially created, we created two mailing lists, "CodeFlow Analytics Discussion" and "CodeFlow Analytics Support" where people interested in using and asking questions about CFA could send messages. Traffic on these mailing lists give an indication of who is using CFA, what they are doing with it, and the difficulties they are running into. We solicited fifteen people who had previously participated in these mailing lists for a short fifteen minute interview and were able conduct nine interviews with a total of eleven people.

The interviews were conducted online and were recorded so that we could refer to them later. Interviews lasted between fifteen and thirty minutes. We chose a semi-structured interview style as that allowed us to obtain information about our general areas of interest while still allowing flexibility to pursue interesting avenues of discussion as the interviewee brought them up [21]. We asked about the following topics during our interviews:

- The background and current role of the interviewee.
- How the interviewee found CFA.
- What the intended purpose of using CFA was.
- How the interviewee used CFA.
- What the impact of their work using CFA was.
- What went well and what challenges they encountered using CFA.

For analysis, we took notes during the interview and also listened to each interview again and took additional notes relative to each area of questioning. We then met together to discuss our notes and common themes that emerged from the interviews. The following subsections contain a discussion of what we learned regarding the topics that we asked about. In cases where we provide quotes, we have removed disfluencies (e.g. "um" and "ah"). We stress that our goal was to understand how teams are using CFA, what they are accomplishing, what went well, and what their pain points are. This investigation was purely qualitative and does not attempt to achieve statistical significance.

B. Background and Role

All of those interviewed were either managers of development teams or individual developers, though one interviewee was a tester when he originally began using CFA. We observe that nearly all activity on the CFA mailing lists come from development managers and developers, indicating that they are the most interested in using data regarding code reviews. Very little discussion comes from program managers or upper management. One interview was with a vendor (a contractor working for Microsoft), while the others were all full time Microsoft employees. The projects that the interviewees currently work on are were intentionally diverse so as to give a broad perspective [22], and included Visio, Bing, OneNote, Xbox, Excel, Office Internationalization, Visual Studio Online, Microsoft Dynamics, and an internal IT ticketing system. This diverse set of teams gives us confidence that if broad consensus for certain topics in our interviews emerged, they are likely representative of CFA users. Geographically, most interviewees are in Redmond, WA, with four others coming from Brazil, North Carolina, and India (2).

C. How is CodeFlow Analytics discovered?

In a company as large and diverse as Microsoft, we have observed that it is not always easy to find something, even if you know that it exists. Conversely, it is hard to publicize a new project or tool to the right people across the entire company. Nonetheless, CFA has achieved a fairly broad adoption. In an effort to understand how people came to know of CFA, we asked them how they found out about it. The answers we were given were diverse and there was no clear consensus that emerged. Some interviewees were told about CFA from friends or coworkers who had used or heard about it. When we first made CFA available, we publicized it by making informal presentations to engineering teams and making announcements on the "CodeFlow Users" mailing list. Some of the interviewees saw the announcement at that time and *"decided to try it out."*

One finding that we had was that those people with a concrete purpose and a good idea of what they wanted to do were more deliberate in trying to find code review data when they discovered CFA. They either asked on the CodeFlow users' mailing list (we are on this list and respond to such requests), or they searched for code review data on CodeBox, an internal SourceForge-like site for hosting community side-projects at Microsoft, and found CFA.

> *Just because you build it, doesn't mean they will come.*
>
> *Matching the right producers to the right consumers is a problem of large organizations. People's habits when looking for systems like CFA are diverse, therefore creators should pursue several avenues including informal talks, announcements on mailing lists and forums, and web locations (internet and/or intranet) to promote their service.*

D. How is starting with CodeFlow Analytics experienced?

Most people started with either the Excel template or with direct access to the database. We observed that most exploratory people used Excel, whereby people with a clear goal used either the database or Excel. Interestingly, those that used the direct database access tended to not use Excel.

Two people said they started by looking at very simple things such as how many reviews and how many comments their team had recently completed so that they could get accustomed to the data. One aspect of this was developing trust in the system and the data. We observed a common pattern of starting simple, then moving on to more sophisticated queries, and finally pursing the actual goal. Several engineers mentioned that the CFA web page was very helpful in getting started.

People felt that it was easy to get started. Several interviewees reported that they were up and running quickly, whether they used Excel or the database entry point. One interviewee told us *"The vendor on my team just grabbed the thing and he got it working in, I think, the matter of an hour."* We believe this has to do with the fact that CFA data can be accessed via multiple interfaces, allowing people to use whatever means is most comfortable and familiar to them. Many engineers, and especially the ones that used direct database access on the raw data, explained that the data schema was intuitive and easy to work with. As one person expressed it: *"The way of organizing the data is very natural and self-explanatory. I don't need to read a lot of documents and I can understand most of them."* We believe that this was the case because the raw collected data maps clearly to the concrete entities and activity in the CodeFlow tool, such as comments, files, and reviewers.

A few engineers that used Excel as a basis for their investigations and used the dimensional model which included the facts (computed metrics) and various dimensions as filters, mentioned that some of the names of facts were not self-explanatory, and that they would like a more explicit definition for the indicators they see.

Provide a clear set of definitions for both the data and the metrics. For some users it will be essential, others can use it to gain confidence in their intuitive understanding.

On the other hand, as people progressed with their analysis and moved on to answer specific questions or reach a specific goal, analysis was experienced as more difficult as questions about the data began to pop up and they were less confident. Several engineers expressed that translating their original questions into something that could be measured was hard. We describe these challenges in a dedicated challenges section (Section III-H), as these were most often experienced after the first steps and analysis with CFA. In general, the support through direct contact and mailing lists has been crucial to helping people handle challenges and deal with the learning curve of using a new system. Interviewees expressed that they appreciated the quick responses and had confidence in the community. One developer told us, *"We work with Trevor closely and he helps a lot. Most of my requirements he takes care of so it's fine."*

Users are more likely to invest time in a solution if they know that the community is active and that they can get help when needed.

E. Why and how is CodeFlow Analytics used?

One of the most important questions we asked in interviews was why people chose to use CFA and how they were able to use it. That is, what was the impetus to search for, find, and use code review data? The answers that we received were varied and many individuals actually provided multiple reasons for using it. In this subsection we have organized their responses into categories and provided descriptions and illustrations.

Empirically Confirming Beliefs

Six of the interviewees indicated that they had beliefs about their code review practices and how well they were working, but they wanted to have empirical evidence to support their anecdotal experiences. They believed that such evidence would provide support for taking actions to change code review practices in their teams

For example, one development lead that works with two teams said that he felt like one team was much faster doing code reviews than the other, and since a change requires code review prior to checking in, their development speed had slowed down. He turned to CFA to confirm his suspicions and to help drive change by adopting the faster team's practices. In his words *"So I wanted to just gather the data to back up my hypothesis and try to inflict the change. Hey, this other team is getting things done much quicker - what are we doing here?"* When the data supported his beliefs, he began trying to understand what the faster team was doing differently so that he could help the slower team.

The manager of the Office internationalization team felt that the distributed nature of his team (some in Russia, some in Brazil, etc.) was hurting productivity because it would take eight hours or longer for an issue to be discussed or resolved (e.g., one person would be sleeping while the other was working and vice versa). His team initially didn't believe there was an issue, but they were more receptive once they saw the data. *"So I just gathered the data to see if I was the crazy one or if my team was off."*

Education

Five interviewees told us that they used data from code reviews to find areas that they could improve through team education. For example, one of the teams looks through all of the comments made in code reviews every two weeks to look for patterns indicative of bad practices, poor understanding of the system, or incorrect use of tools. If there are many questions about how a component works or comments point out errors in an author's change to a component, then the topic of the next training session is teaching about that component. In one instance, they saw many comments about issues in a change that would have been easily caught by StyleCop, a tool to enforce style and consistency rules that the team is supposed to use. As a result, they conducted training on the use of StyleCop so that authors would use it prior to review, saving reviewers' time.

Another developer from Visio manually inspected thousands of comments and created a taxonomy of the types of things to look for during code review. He developed a set of patterns and anti-patterns and wrote a paper for Microsoft's ThinkWeek[1], a forum for any employee to submit ideas around topics that impact the future of the company.

Another individual looked at behavior in code reviews to understand the communication style of the team. This helped uncover the team dynamics and how various members of the team provide and respond to feedback so that they could address problems.

Reports and Dashboards

Almost all of the subjects we interviewed used the data in CFA to generate reports of some kind, including dashboards. The audience for these fell into two main categories. The first

[1] http://research.microsoft.com/en-us/projects/thinkweek/

category were reports intended for the actual teams or individuals participating in code review, so that they could see how they were doing and if they were meeting goals. One member of Bing created a dashboard web site where developers could go to see how they were doing with respect to various metrics of code review. Others created dashboards to actively drive behavior like increased participation rates.

The second category of reporting was reports that were generated for management. As one developer told us, there were monthly meetings with the leadership team where they would talk about how development was doing and what they could do to improve. In these meetings he would *"try to show the metrics to them and see if they can believe that's useful and move the metrics and take action to move the metrics."*

We also found that interviewees fell into two camps regarding how frequently they use CFA. A few individuals mentioned that they really only used CFA once as a sanity check and don't plan on using it regularly. For example, the Microsoft Dynamics Engineering Fundamentals team used CFA when they were doing a self-evaluation of many of their software engineering processes (build, test, code review, etc.) and found that compared to other areas, their code review was going quite well. They said that they felt like code review was *"going ok," "but the CFA stats support that assertion and helped us determine that other things are more pressing."* They used this in conjunction with evidence from other sources to conclude that their effort at improvement would be better used in areas other than code review.

Other teams are generating these reports on a regular basis (most often, teams indicated every two weeks or once per month) to monitor whether they are improving and *"moving the metrics"* (as one developer said) in the right direction, e.g., faster turnaround times from submitting a code review to signoff. One team developed a "leaderboard" of three key code review metrics that is updated on a daily basis.

Increase Code Review Use

Four interviewees told us that they used CFA to measure and try to increase participation in code reviews. One developer worked in two teams in Office and created a dashboard to track participation of testers in the review process. *"We have created a dashboard so that everybody [referring to testers] will see that and they'll basically be motivated to do more code reviews so that their participation is seen there."*

Another developer told us that a team within Bing had a goal of having all non-trivial changes signed-off in code review prior to check in. However, they were not tracking this measure. When they discovered CFA, they used it to examine how close they were to that goal and found that their sign-off rate was approximately 60%. They were able to point this out to upper management, in hopes of creating concrete goals.

Still another developer used it to demonstrate to a contracted remote team that they were missing reviews that they could be participating in.

Common Metrics

Although CFA calculates and makes available over two hundred metrics and filterable attributes, there were a few that emerged quite clearly as the most used among all people interviewed. Teams are most interested in productivity, effectiveness, and participation and they therefore have tried to identify metrics that best approximate these.

Because most teams require code reviews to be completed before a change is checked into the source code repository (and subsequently available to everyone else on the team), the time taken to perform code reviews has an impact on the productivity of the team. Every person that we talked to that used any of the metrics in CFA was interested in *time to first response* and/or *time to completion*. *Time to first response* is a measure of how much time passes between when the author submits a code review and when the first comment or sign-off from a reviewer occurs. *Time to completion* is the time between when the author submits the review to when it has been marked as completed (usually indicating that the change can be checked in).

Many also indicated that they were interested in participation. The developer mentioned in the previous subsection was concerned that a contracted team was only reviewing a small proportion of the changes that they were invited to review. A team in Excel felt that including testers in code review was important. They and other teams measured this by examining the *participation rate*, the number of reviews that a person participates in (provides comments and/or signs off) divided by the total number of reviews that person is invited to.

Lastly, teams want to make sure that reviews were actually having a positive effect and used a few different metrics to try to measure this. *Average Comments* for a reviewer is the number of comments that a reviewer makes per review. If authors are not getting feedback for their changes then reviews aren't having impact. However, we have found (and many teams agree), that not all review comments are actually useful to authors [4]. As a result, some teams have also used *Ever Resolved Comments* which measures the number of comments in a review that are marked as being "resolved," which is some indication that an action was taken and potentially an issue was fixed. Two teams told us that they use both measures to assess the quality of feedback left by a reviewer, by computing the number of comments made by a reviewer that are eventually marked as resolved divided by the total number comments. These are all proxy measures, and the most commonly asked-for metric is a measure of usefulness. As one development manager told us, they want to know, *"Was this an impactful review or a useful comment on the review? [...] Did it result in a change that wouldn't have been there before?"*

Development teams will use metrics that best approximate productivity, quality, and efficiency. Focus on metrics related to those measures instead on "interesting," but ultimately less-actionable metrics.

Metrics versus Data

We observed that, due to the diverse goals that teams using CFA had, the types of information that they wanted were varied. Some only wanted metrics, as they were interested in reports or measuring themselves against quantitative goals. Others wanted the ability to start with aggregate metrics such as average number of reviewers per review and then *"drill down"* to individual data points, especially outliers, for investigation. As mentioned

earlier in this section, a few teams only performed a manual analysis of individual code reviews, such as reading through code review comments for patterns. Thus, it was beneficial to provide both computed metrics and the raw data from which they were derived.

> *Provide both the derived metrics **and** the raw data so that teams can quantitatively measure themselves, but also manually investigate the individual cases when desired.*

F. Impacts and Outcomes from using CodeFlow Analytics

After having examined the reasons that various teams have used CFA, we also asked them about what outcomes they saw and what impact it had.

Few teams actually had quantitative evidence supporting outcomes. One exception was Bing, where the percentage of checkins that had gone through code reviews rose from around 60% to over 80% and some teams are now consistently achieving 100% sign-off. They credit both a focus from upper management (as a result of reports from CFA) and visibility of sign-off metrics in web dashboards.

Four teams indicated that they had seen an improvement by watching the metrics change over time (thought they did not provide us with actual numbers) and also observed attitude shifts in the team with regard to the importance of code review. For example, Visio did see a rise in tester participation once the team began monitoring and encouraging code review activity.

For two teams, their use of CFA helped them decide that their effort would best be served in areas other than code review. So while the outcome was that they did not change their practices, they were more confident in the decision to not change because it was backed by data.

Several of those interviewed expressed a feeling that they believed it would have an impact, but that it was still too early to tell how much of an impact they would see. Some were also worried about possible negative impacts. *"I have no doubt that those analysis will lead to behavior changes, but I am concerned about that change,"* said one, who was fearful that team members might focus too much on numbers.

As a few teams had education as their impetus to use CFA, a common outcome was improved training. One interviewee travelled to Asia to provide training on code review practices to a contracted team. Another team has provided training multiple times on weaknesses found through analysis of code reviews. Still another wrote an internal paper on reviewing practices.

G. Challenges faced during use of CodeFlow Analytics

This section details some of the challenges we observed during the interviews. Some of these can be used to improve the CFA experience. Other challenges are inherent to analytics, and some really can't be fixed. Nonetheless, users of analytics systems should be aware of them.

Translating questions into metrics

Several engineers had a hard time translating their original questions into metrics that they felt reflected their initial intent. Engineers explained that they had to decide exactly what the terms they used meant. For example, several participants mentioned that they wanted to measure response time and participation rate, but initially they had no concrete definition for how a response time should be measured. Is it the time to first comment, or the time to first signoff? In general, people mentioned that they had to make tweaks to the metrics provided to make them useful for their context. They did indicate that the availability of predefined metrics in CFA supported them during the investigation and clarification process and provided a good starting point for more detailed adjustments.

Deriving proxy measures

Some engineers explained that they had to create proxy measures, as the direct measure could not be implemented. For example, several engineers wanted to measure the duration from the creation of a code review to successful checkin of this code change in the code base. As CFA data contains only review data and does not have repository checkin information, such metrics cannot be obtained from the single database. In most cases, engineers used the time to "signoff", i.e., the event that peer reviewers deemed the code change as good enough for checkin, instead of checkin time. One engineer said: *"I was able to gather the data. […] I think I have to refine the query between what's available and redefining my question to be a more correct one. But in the end […] I got close enough data so I was able to deliver the point to the team."*

> *No data is an island - The ability to "link" and join different data sources reflecting different parts of the development process (e.g., review and testing practices) is important. We should enhance our solution space and make available linkages more accessible.*

Unawareness of data availability

Some of the data points or metrics that engineers said were not in CFA were actually either directly accessible through the database interface or by querying the data in a certain way. For example, many engineers wanted to look at individual reviews and comments. Those are not surfaced in the Excel template, but they can be obtained with additional work. Some gave up on this task, while others looked manually through code reviews and comments by opening them via the normal code review dashboard—a tedious manual task. One potential fix for this is including low-level individual data in the example Excel template (e.g., last 10 recent comments). As we discuss in Section II, we do have data that allows users to "link" code reviews to checkins. However, only one participant discovered this data. Discovering such information should be easier for users, for example by discussing different usage scenarios on the CFA web site.

> *Just because users use part of your system doesn't mean they know about all of it. Always consider discoverability.*

Interpretation of data

Interpretation of analytics data is not always straight forward. In our study, we observed problems with the ability to interpret the data at hand. For example, some people didn't know about "FYI" reviews. This a practice used by some teams where they create reviews whose sole purpose is to inform others about changes and not to solicit feedback from reviewers. This affects participation rates, as team members "invited" on these reviews

never actually take any action, other than looking at the change. Some interviewees unaware of this practice were surprised to see very low participation rates for some teams. Also, some activities that are recorded in CFA are not performed by humans, but are automated tool activities. For example, in one case an engineer explained that most of the reviews contain comments that occur within the first few seconds of review creation (in this case, an automated system scans the change for common easy to detect errors). This prohibited them from looking at first response time, as they were unable to separate the tool from human activity. As another example, CodeFlow allows engineers to "like" others' review comments with a "like" button similar to Facebook. Nevertheless, some people explained they didn't know what the like button was and how to use it, nor did they know how they should interpret likes of comments.

Using data without knowing the process that created it is fraught. Different teams may use the same tool in vastly different ways.

Deriving and interpreting useful indicators

Several engineers explained that some of the indicators were straightforward to derive and interpret (e.g., average number of reviews authored or reviewed), whereby more sophisticated indicators were difficult to derive. The difficulty here did not lie in previously described points, but in the design and interpretation of the indicator itself: it was unclear whether the signal observed from the indicator is a good or a bad sign. One engineer formulated it like this: *"A lot of metrics we want to use like an inquiry methodology. It's like, 'why does the data look like that?' We don't know if it's good or bad, but it would be good to take a look at that. A perfect example is signoff time. If we look at signoff time, it could take two hours, it could take 2 days, it could take 20 days, but which one is better, we don't know. It could be signoff one second blindly, which is not the behavior that we want to encourage, but a long signoff with a very simple code review is also something I don't want to encourage, so that's the kind of thing where we show some data, show some distribution and see the outliers and have people look at that."*

Many people understand that looking at metrics does not explain the causality for a phenomena or why they see a certain value. Several explained that they engaged in more thorough investigations for the outliers they saw or to understand why certain values occurred. In general, we observed that people were cautious about abusing metrics and wanted to avoid driving incorrect behaviors.

Getting to actionable outcomes

Even though some people explained how they actively use the derived metrics and report to drive behavior, several explained that making the results actionable is difficult. This challenge is not specific to CFA, but applies in general to the analytics field [23]. On the other hand, during the interviews people explained that they had ideas of how to make this easier. First and foremost, they asked for a list of frequently asked questions and answers that detail common mistakes and also common uses/metrics/indicators. Another frequent suggestion was to make it easier for users to share their insights and metrics with others.

Unfamiliar with analytics

Not everybody that wanted to use the data was experienced or familiar with data analytics. Some people explained that distributions and percentiles were missing, and that they had a hard time deriving them themselves. Also, normalization of the data was experienced as a challenge. They said that having better predefined Excel templates or other kinds of examples or queries would be really helpful. The optimal solution for them would be free analytics that didn't need any (or not much) customization. In general, if a user didn't already have analysis skills, the required time investment to make CFA useful for their purpose was problematic, as the main responsibility of those using CFA was development and not data analysis.

Explicit analysts that take over consulting functionality could help product teams with various challenges described, e.g., to derive meaningful indicators, drive desired process improvements, or share insights and lessons learned from working with other teams.

Access and permissions

Gaining access and permissions to use the CFA data was a problem for some people. In addition, several explained that it was unclear how they could share the data with a broader set of people (e.g., their entire team or management) if they used the direct database access. How could the larger team use it? In a few cases, people explained that they ran into permission issues when sharing the Excel files (e.g., others couldn't open them because they didn't have permissions). One person explained that initially he tried to access CFA, but because of permission problems, he gave up. As a result of early feedback from users, we modified our permissions policies to make access available to all Microsoft employees. A few months later he found himself trying again and by that time we had made the changes necessary, so he was able to access what he needed. However, it is unclear how many potential users tried, failed, and are not aware of the access changes.

Access and permission problems can quickly drive users away. Efforts should be taken to make the entry point as painless as possible.

Not everyone is a data expert

One engineer explained that upper management had problems with the visualization in Excel, as column names might be truncated, and they would not know how to enlarge the columns to see the whole metrics name. Further he explained that the reports got so complex that he ran out of screen real estate (i.e., he could not add any more columns without creating the need for the users of his report to scroll within the Excel sheet). While this sounds trivial, this is actually a real problem. People should be able to create reports and pass them around to others that are not experts in analytics or our system.

It should be easy for someone using your data and analytics platform create a report that a non-expert can read and understand.

Requests for the future

In the interviews, we also asked each participant what features they felt were missing in CFA, or if they had future requests for what they would like to see implemented. Most of the engineers had no direct or immediate requests for future features in CFA. On the other hand, a few mentioned issues that we also heard indirectly from others, during their explanations of their approaches - and especially the challenges - they faced (e.g., more pre-defined indicators, more fine granular data). One common request was an automatic way to classify and assess the usefulness of comments. This is a separate research endeavor unto itself.

IV. Research that Leverages CodeFlow Analytics

CodeFlow Analytics has not just enabled development teams to monitor themselves and find ways to improve; another important outcome from CFA has been that ability to use it as a platform for research. In this section we survey studies and tools that are enabled by CFA. Our purpose is not to discuss the techniques or conclusions of each piece of research, but rather to indicate the ways in which CFA is used to support.

In 2012, we conducted a study on code review at Microsoft with the intent of understanding why teams were motivated to perform code reviews, what they hoped to get out of doing code reviews, and what the actual outcomes were [4]. We used CFA to identify over 1,000 participants for the study, as we were interested in finding both developers and managers of teams that used CodeFlow to involve in observations, interviews and a broad survey. We also manually inspected and categorized 570 code review comments.

In 2013, we performed an empirical study of code review practices at Microsoft, Google, and AMD in an effort to determine what differed and what was similar across these companies [1]. We used CFA to gather and analyze metrics from Microsoft, including the time between review submission by the author and first activity by a reviewer, the total time to complete a review, the number of reviews per month, the number of lines modified and files changed per review, the number of comments per review, and distributions of the number of people participating per code review. We found that there are a number of practices and characteristics of code review that are very consistent across all companies included in the study.

Based on feedback from our earlier observations of code reviews, and prior research of others that indicated that small cohesive changes are easier to understand and elicit higher quality feedback [24] [25], we developed ClusterChanges, a system for decomposing a change under review into relatively distinct "partitions" that can be reviewed independently [26]. We initially used CFA during the implementation of ClusterChanges so that we could test it on real reviews that fulfilled some criteria (e.g. at least four source files, made up mostly of C# source code). When it came time to perform a user study of ClusterChanges, we used CFA each morning to identify authors of reviews created the previous day, so that the review was fresh in their mind, and contacted them to invite them to try ClusterChanges. We also used CFA to randomly select 1000 reviews that fit our criteria that we applied ClusterChanges to, to gather distributions of decomposition metrics.

Allamanis et al. have been working on a line of research that applies natural language probability models to source code [27]. We have collaborated with them in an effort to use these language models to infer code conventions, find team coding convention violations, and provide suggested changes to improve convention adherence [28]. As part of that work, we used CFA to examine 1000 code review comments made by reviewers in 169 reviews, to see how often those reviews include feedback about following conventions, and found that 38% of them did.

We recently embarked on a study to determine what makes code review comments useful to authors, what factors have a relationship with comment usefulness, and if we could build a classifier to automatically determine if a comment is useful [29]. We used features of code reviews exclusively from CFA, including content of the comment, who made the comment, the size of the review, and where code changed in the review, to train and evaluate our classification model, which is able to achieve precision and recall rates between 85% and 90%. We also investigated the effects of factors such as developer experience, types of files being reviewed, and team membership on comment usefulness.

One common challenge that many teams performing code review face is knowing who to include on the code review. A system to recommend the best reviewer is a common request from teams at Microsoft and other companies [2]. We have developed such a review recommendation system, currently in pilot phase, that recommends reviewers based on data from CFA. This considers the files a developer has authored and reviewed in the past, how many reviews a developer currently has outstanding, and the developer's median response time to reviews.

The impact of the described research may not be as direct as the impact teams can have that use CFA themselves. However, these studies do provide insight regarding the code review process of developers at Microsoft, and the tools resulting from such studies are already beginning to help teams conduct code reviews more efficiently and productively.

V. Conclusion

We have described our experiences designing, deploying, and supporting CodeFlow Analytics, a code review data analytics platform. As we had hoped, teams across Microsoft have begun using this platform, but it has not been without a few challenges. While we have been able to address some of them, such as dealing with branches and linking reviews to commits, we still need to improve in other areas. Despite these, CFA has already had a positive (and in many cases, measurable) impact on development teams. It has also enabled research on many aspects of code review. We hope that others will find our experience useful as they build their own data collection and analysis systems.

VI. Acknowledgment

We would like to thank the developers of CodeFlow as well as the development teams that let us interview them about their experience with CodeFlow Analytics.

VII. REFERENCES

[1] P. C. Rigby and C. Bird, "Convergent Software Peer Review Practices," in *Proceedings of the the joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering (ESEC/FSE)*, 2013.

[2] V. Balachandran, "Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation," in *Proceedings of the 2013 International Conference on Software Engineering*, 2013.

[3] S. McIntosh, Y. Kamei, B. Adams and A. E. Hassan, "The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, 2014.

[4] A. Bacchelli and C. Bird, "Expectations, Outcomes, and Challenges of Modern Code Review," in *Proceedings of the 35th International Conference on Software Engineering*, 2013.

[5] T. Ball, J.-M. Kim, A. A. Porter and H. P. Siy, "If your version control system could talk," in *ICSE Workshop on Process Modelling and Empirical Studies of Software Engineering*, 1997.

[6] T. Wolf, A. Schroter, D. Damian and T. Nguyen, "Predicting build failures using social network analysis on developer communication," in *Proceedings of the 31st International Conference on Software Engineering*, 2009.

[7] N. Bettenburg, S. Just, A. Schroter, C. Weiss, R. Premraj and T. Zimmermann, "What makes a good bug report?," in *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, 2008.

[8] A. Meneely, A. C. R. Tejeda, B. Spates, S. Trudeau, D. Neuberger, K. Whitlock, C. Ketant and K. Davis, "An empirical investigation of socio-technical code review metrics and security vulnerabilities," in *Proceedings of the 6th International Workshop on Social Software Engineering*, 2014.

[9] M. Beller, A. Bacchelli, A. Zaidman and E. Juergens, "Modern code reviews in open-source projects: which problems do they fix?," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, 2014.

[10] P. Thongtanunam, R. G. Kula, A. E. C. Cruz, N. Yoshida and H. Iida, "Improving code review effectiveness through reviewer recommendations," in *Proceedings of the 7th International Workshop on Cooperative and Human Aspects of Software Engineering*, 2014.

[11] K. Herzig and A. Zeller, "Mining Your Own Evidence," O'Reilly Media, Inc., 2010.

[12] J. Czerwonka, N. Nagappan, W. Schulte and B. Murphy, "CODEMINE: Building a software development data analytics platform at Microsoft," *Software, IEEE,* vol. 30, pp. 64--71, 2013.

[13] "A Bar, an Idea, and a Garage: The Story of CodeFlow," Microsoft, 5 Janjuary 2012. [Online]. Available: http://news.microsoft.com/2012/01/05/a-bar-an-idea-and-a-garage-the-story-of-codeflow/. [Accessed February 2015].

[14] N. Kennedy, "How Google does web-based code reviews with Mondrian.," 2006.

[15] A. Tsotsis, "Meet Phabricator, The Witty Code Review Tool Built Inside Facebook.," 2006.

[16] Wikipedia, "Gerrit (software)," 2012.

[17] R. Kimball and M. Ross, The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling, Wiley, 2013.

[18] "Microsoft Azure," Microsoft, [Online]. Available: http://azure.microsoft.com/en-us/overview/what-is-azure/. [Accessed February 2015].

[19] "Analysis Services," Microsoft, [Online]. Available: https://msdn.microsoft.com/en-us/library/bb522607.aspx. [Accessed February 2015].

[20] "Demystifying Power BI Q&A," 27 Feb 2014. [Online]. Available: http://blogs.msdn.com/b/powerbi/archive/2014/02/27/demystifying-power-bi-q-amp-a-part-1.aspx.

[21] T. Wengraf, Qualitative research interviewing: Biographic narrative and semi-structured methods, Sage, 2001.

[22] M. Nagappan, T. Zimmermann and C. Bird, "Diversity in Software Engineering Research," in *Proceedings of the the joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering (ESEC/FSE)*, 2013.

[23] T. H. Davenport, J. G. Harris and R. Morison, Analytics at work: Smarter decisions, better results, Harvard Business Press, 2010.

[24] Y. Tao, Y. Dang, T. Xie, D. Zhang and S. Kim, "How do software engineers understand code changes?: an exploratory study in industry," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, 2012.

[25] P. C. Rigby, B. Cleary, F. Painchaud, M. Storey and D. M. German, "Contemporary peer review in action: Lessons from open source development," *Software, IEEE,* vol. 29, pp. 56--61, 2012.

[26] M. Barnett, C. Bird, J. Brunet and S. Lahiri, "Helping Developers Help Themselves: Automatic Decomposition of Code Review Changesets," *Proceedings of 37th IEEE/ACM International Conference on Software Engineering,* 2015.

[27] M. Allamanis and C. Sutton, "Mining source code repositories at massive scale using language modeling," in *Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on*, 2013.

[28] M. Allamanis, E. T. Barr, C. Bird and C. Sutton, "Learning Natural Coding Conventions," in *Proceedings of the 22nd International Symposium on Foundations of Software Engineering*, 2014.

[29] A. Bosu, M. Greiler and C. Bird, "Characteristics of Useful Code Reviews: An Empirical Study," *Submitted to the International Conference on Mining Software Repositories,* 2015.