Information Needs for Software Development Analytics

Raymond P.L. Buse and Thomas Zimmermann

***Abstract*—Software development is a data rich activity with many sophisticated metrics. Yet engineers often lack the tools and techniques necessary to leverage these potentially powerful information resources toward decision making. In this paper, we present the *data and analysis needs of professional software engineers*, which we identified among 110 developers and managers in a survey. We asked about their decision making process, their needs for artifacts and indicators, and scenarios in which they would use analytics.**

The survey responses lead us to propose several guidelines for analytics tools in software development including: Engineers do not necessarily have much expertise in data analysis; thus tools should be *easy to use, fast, and produce concise output*. Engineers have diverse analysis needs and consider most indicators to be important; thus tools should at the same time *support many different types of artifacts and many indicators*. In addition, engineers want to *drill down into data* based on time, organizational structure, and system architecture. We validated our guidelines with a proof-of-concept implementation of an analytics tool, which we used to solicit additional feedback from engineers on how future analytics tools should be designed.

I. Introduction

Software engineering is a data rich activity. Many aspects of development, from code repositories to testing frameworks to bug databases, can be measured with a high degree of automation, efficiency, and granularity. Projects can be measured throughout their life-cycle: from specification to maintenance. Numerous metrics and models for complexity, maintainability, readability, failure propensity and many other important aspects of software quality and development process health (e.g., [15], [37]) have been proposed.

Despite this abundance of data and metrics, development continues to be difficult to predict and risky to conduct. It is not unusual for major software projects to fail or be delayed [1]. Moreover, software defects cost the US economy many billions of dollars each year [43]. Together, these observations imply that there continues to be a substantial disconnect between the information needed by project managers to make good decisions and the information currently delivered by existing tools.

When information needs are not met, either because tools are unavailable, too difficult to use, too difficult to interpret, or they simply do not present useful or actionable information, managers must primarily rely on past experience and intuition for critical decision making. Such intuition-based decisions can sometimes work out well, but often they are unnecessarily suboptimal or even destructive [54]. As software projects continue to grow in size and complexity, decision making will likely only become more difficult.

Analytics describes application of analysis, data, and systematic reasoning to make decisions. Analytics is especially useful for helping users move from only answering questions of information like "What happened?" to also answering questions of insight like "How did it happen and why?" Instead of just considering data or metrics directly, one can gather more complete insights by layering different kinds of analyses that allow for summarizing, filtering, modeling, and experimenting; typically with the help of automated tools.

The key to applying analytics to a new domain is understanding the link between available data and the information needed to make good decisions, as well as the analyses that are appropriate to facilitate decision making. While there has been significant research into information needs of developers (e.g., [7], [31], [51]), the needs of project managers, those who make the important decisions about the future of projects, are not well understood.

In this paper, we present a quantitative and qualitative study of the information needs of 110 developers and managers at Microsoft. We discuss the decision scenarios they face, the metrics and artifacts they find important, and the analyses they would most like to employ. We distill from our study a set of key guidelines that should be considered when designing an analytics tool. We find for example that managers must be able to "drill-down" from high level summaries all the way to the artifacts being measured (e.g., change records). This is essential to permit both discovery of the insights as well as a concrete basis for action. To validate our guidelines we additionally present a proof-of-concept analytics tool which exercises many of the key findings of our study with a number of novel features including *drill-down*, *summarization* based on topic analysis, and *anomaly detection*. We present both the tool itself and feedback we solicited from managers at Microsoft.

A. Contributions

The main contributions of this paper are:

- A study of the information needs of 110 professional software engineers and managers. In particular, our

R. Buse is with The University of Virginia, USA. He was an intern at Microsoft Research when this paper was written. Email: buse@cs.virginia.edu

T. Zimmermann is with Microsoft Research, Redmond, USA. Email: tzimmer@microsoft.com

Microsoft Research. Technical Report MSR-TR-2011-8.

study focuses on analytical decision making (Section V).
- A set of guidelines for software analytics. We present a characterization of analytics problems in software development based on our study (Section VI).
- A proof-of-concept analytics tool. We present a tool with novel features based on multi-dimensional filtering, latent Dirichlet allocation (LDA) topic analysis of commit messages, and anomaly detection (Section VII). Our tool serves as a validation of our guidelines and means of soliciting additional feedback on analytics which we present in Section VII-B.

We begin by discussing previous research related to project management (Section II) and measurement (Section III).

II. Project Management

Software project management is a complex and broadly defined position. Project managers monitor and guide the work of designers, developers and testers of software while sometimes participating in these activities themselves. Where engineers focus on code, architecture, and performance, managers focus on high level concerns: the direction of the project, allocation of resources, the feature set, and the user experience. Managers work to simultaneously satisfy (potentially conflicting) constraints imposted by customers, developers, testers, maintainers, and management. Steven Sinofsky, a manager at Microsoft, notes that "it is almost impossible to document a complete list of the responsibilities of program managers" [53].

The complexity of the job of a manager contributes to the difficulty of designing and evaluating tools by the research community. In particular, the information needs of managers are not well understood. Boehm and Ross proposed a theory of project management which included the "top 10 primary sources of software project risk and the most effective approaches for resolving them" [12]. While top ten lists like this can be instructive, the problem is that the management techniques presented (e.g., benchmarking, organization analysis, technical analysis, etc.) aren't specific enough to be directly applied. Many critical questions remain unanswered: Which of these can be performed automatically? Which are most important? How should the results be presented? What decisions can they lead to? How does one evaluate success?

More recently, in an effort to begin answering some of these questions, Wallace *et al.* conducted a survey of 507 project managers [58]. Cluster analysis was used in an attempt to identify risk factors in projects. That study found, for example, that "even low risk projects have a high level of complexity." The study did not produce a practical answer to the question of what software indicators managers should be concerned with.

Some studies exists that identified information needs for managers for specific decision-making tasks. Jedlitschka [26] identified the needs of managers when assessing alternative technologies. He empirically showed the importance and impact on costs, quality, and schedule, and limitations of the technology for a manager's decision. Vegas *et al.* [57] identified questions that decision makers have when choosing among different testing techniques. Punter [47] investigated what information software managers would expect from a software engineering portal and found that all the information he included in his study was expected to be found by most of the respondents. Similarly, we found in our study that most indicators are important for engineers, however some indicators are more important than others (such as failure information and bug reports).

Komi-Sirvio *et al.* noted that software managers are typically too busy with their day-to-day duties to spend much time performing measurement activities [32]. Typically data-driven tasks are relegated to secondary work. Rainer *et al.* [48] found that software managers prefer information from colleagues and do not consider empirical evidence as comparably relevant. We believe that this should be changed and results from our study actually suggest that managers do recognize the importance of data for decision-making (Section V-B).

Goal-oriented approaches use goals, objectives, strategies, and other mechanisms to guide the choice of data to be collected and analyzed. For example, the Goal/Question/-Metric (GQM) paradigm [4] proposes a top-down approach to define measurement: goals lead to questions, which are then answered by metrics. Other well-known approaches are GQM+ which adds business alignment to GQM [5], Balanced Scorecard (BSC) [28], and Practical Software Measurement [45]. We believe that software development analytics complements existing goal-oriented approaches well. The availability of dedicated analysis tools will give managers more flexibility to follow the goal-oriented approaches in their decision making. The design of such tools is informed by the study in this paper, which identified frequent goals, questions, and metrics in decision making. Also we want to emphasize that *analytics is not just limited to measurement*; qualitative analysis is equally important in analytics (see Figure 1) and can be the key to "solving the 'Why' puzzle" [29].

Basili *et al.* [3] proposed the Experience Factory, which is an organization to support a software development in collecting experiences from their projects, packaging those experiences (for example in models), and in validating and reusing experiences in future projects. Software development analytics builds on this idea and has similar goals. However, rather than having a separate organization, we ultimately want to *empower software development teams to independently gain and share insight* from their data without relying on a separate entity.

In a previous technical report we summarized some of our early ideas on software development analytics [16]. With this paper we make several novel contributions that are not in the technical report: a study of the information needs of software engineers and managers (Section V), guidelines for software analytics (Section VI), which have been validated through a prototype tool and supplemented with feedback from software engineers (Section VII).

III. Existing Tools

There are a number of existing tools designed to support management. For example, `PROM` [52] and `Hackystat` [27] are both capable of monitoring and reporting a great number of software project statistics. However, after seven and ten years of development respectively, neither have seen significant levels of adoption outside of academia [23]. One explanation might be that these tools simply do not answer the right questions [19]. Each of these tools focus primarily on data collection, which while a challenging problem in of itself is probably no longer the most critical concern.

More recent tools, which have focused on data presentation rather than collection have met with some adoption. Microsoft's `Team Foundation Server` [39] and IBM's `Jazz` developer environment [25], for example, provide *dashboard* views designed to keep developers up-to-date on the status of various events like modifications, bugs, and build results. `Ohloh.Net` is an open-source community with the goal to connect people through the software they use and create. The web-site shows in dashboard-like views how the software was developed in terms of lines of code, contributors, and commits. However, a recent study concluded that while integrated tooling of this type could help support the development process, "the distinction between high-level and low-level awareness is often unclear" [55].

While modern tools can present a large amount of data from varied sources, most either focus on data collection or on developer *awareness*; because they don't have a good model for the needs of real product managers, real product managers do not generally use them. Furthermore, managers may be too busy or may simply lack the quantitative skills or analytic expertise to fully leverage advanced analytical applications which may range from trend analysis, classification algorithms, predictive modeling, statistical modeling, optimization and simulation, and data- and text-mining [20].

IV. Software Analytics

Analytics has revolutionized decision making across many fields [20]. Web analytics, for example, leverages large volumes of click-stream data to help website managers make informed decisions about many aspects of their business from advertising to content layout to investment. Today, large websites not only thoroughly test all proposed changes in the traditional sense, but they also undertake detailed analytic experiments aimed to precisely quantify the net benefit of any proposed change [29]. Analytics has had a profound effect on businesses ranging from technology to retail to finance.

In the context of software engineering we hypothesize that analytics can help answer important questions mangers ask about their projects. The goal of analytics to assist decision makers in extracting important information and insights that would otherwise be hidden.

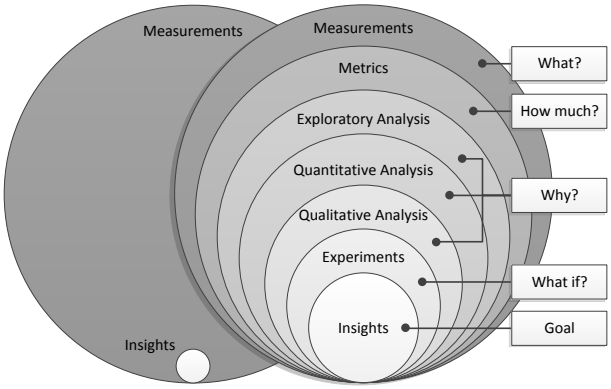

Fig. 1. Paradigm of Analytics (adapted from [29]): Analytics is based on the composition of many types analyses to formulate more complete insights.

When using flat measurements only, it's difficult for managers to glean more than sparse insights. Figure 1, which we adapt from Kaushik [29], illustrates how layering many types of analyses can greatly increase the net yield of useful information. For example, measuring the defect rate of a project to be 10% doesn't provide much insight. However, if this measurement is 2% higher than a month ago, or higher than other similar projects at this phase of the release cycle, it might be cause for concern. To act on that information it is the important to "Drill-down" further: Why is the defect rate high? Which areas of the project are most prone to defects? Did they appear over time or all at once? Which authors are responsible? Do the defect rates correlate with complexity? with some other metric? What would happen if we increased test coverage?

The overarching goal of analytics is to help managers move beyond information and toward insight. However, such a transition isn't easy. Insight necessarily requires knowledge of the domain coupled with the ability to identify patterns involving multiple indicators. Analytical techniques can help managers quickly find important needles in massive haystacks of data.

Software engineering has many qualities that suggest a business process that lends itself well to analytics:

- **Data-rich.** Analytics operates best when large amounts of data are available for analysis.
- **Labor intensive.** Analytics enable leverage of expertise especially where talent supply is short, demand is cyclical, and training times are lengthy [20]. Software engineering is especially labor intensive. Furthermore, numerous studies have found an order of magnitude productivity gap between developers [11], [56].
- **Timing dependent.** Analytics can be helpful in cases where business products must meet specific schedules; analytics enable decision makers to look both upstream and downstream.
- **Dependent on consistency and control.** Analytics help enable consistent decision making even under unusual circumstances [20].
- **Dependent on distributed decision making.** Analyt-

ics can help decision makers understand the collective state of projects even in the face of geographic distribution. Many software projects, especially open source, exhibit highly distributed development activity.
- **Low average success rate.** Domains with a high failure rate are the most likely to benefit from analytics. Software projects fail as much as 33% of the time [1].

There are many potential advantages to the application of analytics to software project management (inspired by [20]). Analytics can help mangers:

- **Monitor a project.** Analytics provides tools to help managers understand the dynamics of a complex project.
- **Know what's really working.** Analytics can help evaluate the effectiveness of a change to the development process. For example, it can measure whether some effect is statistically significant.
- **Improve efficiency.** Techniques based on analytics can help mangers allocate resources and talent to where it is most needed and recognize when under utilization occurs.
- **Manage risk.** The effectiveness of risk models can be improved with increases in the quantity and precision of data. Analytics can provide both a data channel into risk models and a vehicle for delivering the output of such models in an intuitive way.
- **Anticipate changes.** Analytics can help managers to detect and forecast trends in data.
- **Evaluate past decisions.** Logical and consistent decision making based on data is much more amenable to later review and assessment than decisions based on intuition.

Analytics helps describe a reasoning framework which we've observed has the potential to fit well with software engineering. However, to realize that potential it is obvious that studies of the information needs of decision makers are needed. In the next section we discuss our study of managers and developers at Microsoft.

V. Analytics Study

In order to adapt analytical techniques to software engineering, by prescribing process changes or building tools, it is important to first understand the information needs of managers, those who make decisions about the direction of a project. In this section we present a study of a large group of managers and, for comparison, developers who work at Microsoft. We describe the administration of the study as well as analyze the results. In Section VI we characterize the results of the study in terms of an information needs spectrum and describe corresponding analytical solutions.

A. Methodology

We advertised a survey consisting of 28 questions over email to a number of software engineers at Microsoft. We sampled randomly from equal-sized pools of engineers and lead engineers. A lead engineer at Microsoft directly manages a development team. He or she has responsibilities including defining scheduling commitments, establishing priorities (especially in consideration of customers), and developing effective metrics. The lead engineers we surveyed do not spend all of their time working in a strictly management capacity. On the contrary, they typically spend about 60% of their time working on engineering and testing tasks, and the other 40% managing. Clearly, there is not always a clear distinction between managers and developers. However, for simplicity, in this paper we refer to engineers as *developers* and lead engineers as *managers*.

A total of 110 individuals participated in the survey, 57 managers and 53 developers. The participants work in a diverse set of project domains; they include entertainment, on-line services, business applications, and systems. Despite this variety of participation less than 4% of responders felt the survey was not relevant to them. Over 20% thought the survey was "very relevant" including 52% managers.

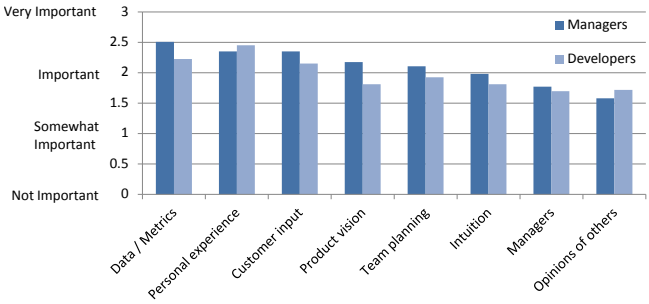

Fig. 2. Reported importance of factors to decision making amongst developers and managers.

B. Analytical Questions

What factors most influence your decision making process?[1]

Analytics is about forming insights and making decisions based on data. Nonetheless, data is not the only reasonable basis for decision making. Managers and developers make use of their own experience and intuition as well as the input of others when making decisions related to their work. We asked survey participants to rate the importance of a number of such factors on a 4-point scale {Not Important=0; Somewhat Important=1; Important=2; Very Important=3}. The average *importance* score assigned to each factor is presented in Figure 2.

Interesting to note is that managers rated *Data and Metrics* as the most important factor to their decision making. Developers, on the other hand, rated their personal experience as most important; this, despite the fact that they have about 6 years less experience to draw from on average. In absolute terms, both pools agreed that data is an important factor, however managers felt it is more important (T-test significance level ¡ 0.02). One possible implication is that managers have learned to rely less on their experience

[1]The questions in the paper are edited for brevity. The original questions are more specific and listed in Appendix B.

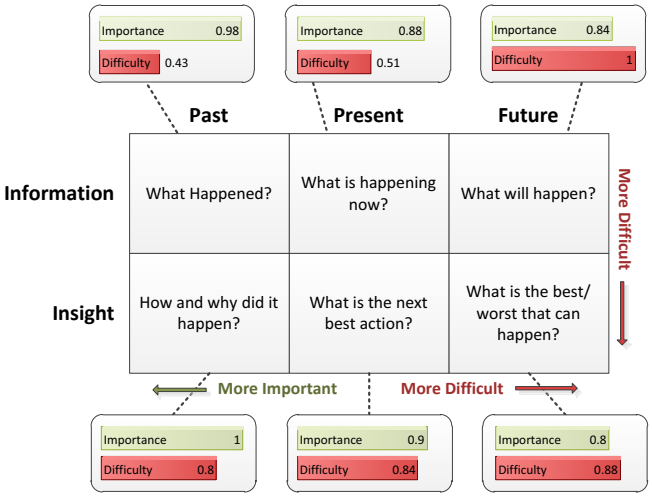

Fig. 3. Analytical Questions (adapted from [20]): We distinguish between questions of *information* which can be directly measured, from questions of *insight* which arise from a careful analytic analysis and provide managers with a basis for action.

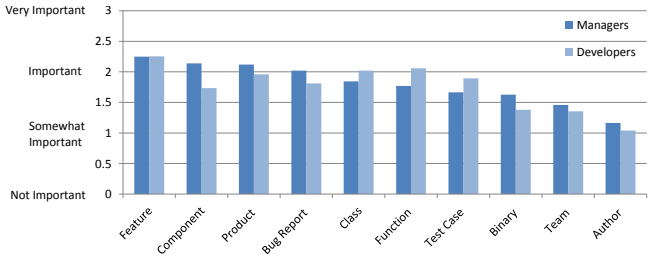

Fig. 4. Reported importance of measuring the given types of artifacts amongst developers and managers.

over the course of their career. In any case, the great interest in data as compared to powerful factors like product vision, planning, and management suggests that many in the software development field, and particularly managers, would be open to more analytical tools and techniques if they could be made to fit their needs.

What questions are important or difficult to answer?

Analytics can help distinguish questions of *information* which are available through a number of existing tools (e.g., how many bugs are in the bug database?), from questions of *insight* which provide managers with an understanding of a project's dynamics and a basis on which to make decisions (e.g., why is the project delayed?). Davenport *et al.* [20] identify six question areas analytics can help answer organized by time-frame and by information vs. insight. We asked survey participants to rate both the importance and difficulty of answering questions in each domain on the same 4-point scale we used earlier. We average and normalize the results to facilitate comparison in Figure 3.

Unsurprisingly, participants noted that questions of insight are generally more difficult to answer than of information, and furthermore that they become even more difficult if they pertain to the future as compared to the past or the present. Surprisingly, questions about the future were rated as progressively less important (though still important in absolute terms). In other words, both developers and managers find it more important to understand the past than try to predict the future; echoing George Santayana, "those who cannot remember the past are condemned to repeat it."

One potential explanation is that managers sometimes distrust predictive models. Such models lack transparency, so to make a decision based on one is to make a decision without fully understanding why the decision needs to be made. When it comes to critical decisions, transparency is important.

Furthermore, this finding was alluded to by a number of responders in a free-form response field. Consider hypothesis testing: for example, a manager might suspect that a feature is prone to bugs because it lacks clear ownership. In that case the manager might attempt to test that hypothesis by inspecting some amount of relevant data. If the hypothesis is supported, the manager might then insist that in the future all features have well-defined ownership. In this way, the analysis question relates primarily to the past and present, but the effect is felt in the future. New analytical tools, like those which we describe in Section VI might assist managers not only in formulating hypotheses, but also in conducting more principled analyses and experiments.

C. Indicators and Artifacts

What artifacts are important to measure?

In the context of software engineering, there exists a wide variety of artifacts suitable for analysis. For a given indicator, complexity for example, one can evaluate it on individual function, a class, all code associated with a given feature, written by a certain author, touched by a test suite, or even for an entire product provided these mappings can be established. We asked those who participated in our survey to rate the importance of a number of artifacts on a 4 point scale, Figure 4 shows how they responded on average.

Overall, it is clear that many artifacts provide unique and valuable information. Analyzing the individual features of a project, however, was deemed most important by both developers and managers. This makes sense considering that many important decisions relate directly to features (e.g., release planning). We discuss the importance of features in more depth in Section VII-B. Other important artifacts include components, entire products, and bug reports; each of these high-level artifacts are most important to managers. On the other hand, lower level constructs like classes, functions, and test cases are most important to developers.

We conclude that not only are each of these artifacts important, but they are important *simultaneously*. While a manager may wish to measure say, code churn for the project, it is just as important to "drill down" and determine which individual classes, functions, or teams are connected to that churn in order to take action as needed. We elaborate on this idea in Section VII-B.

What indicators do you currently use?
What would you like to use?

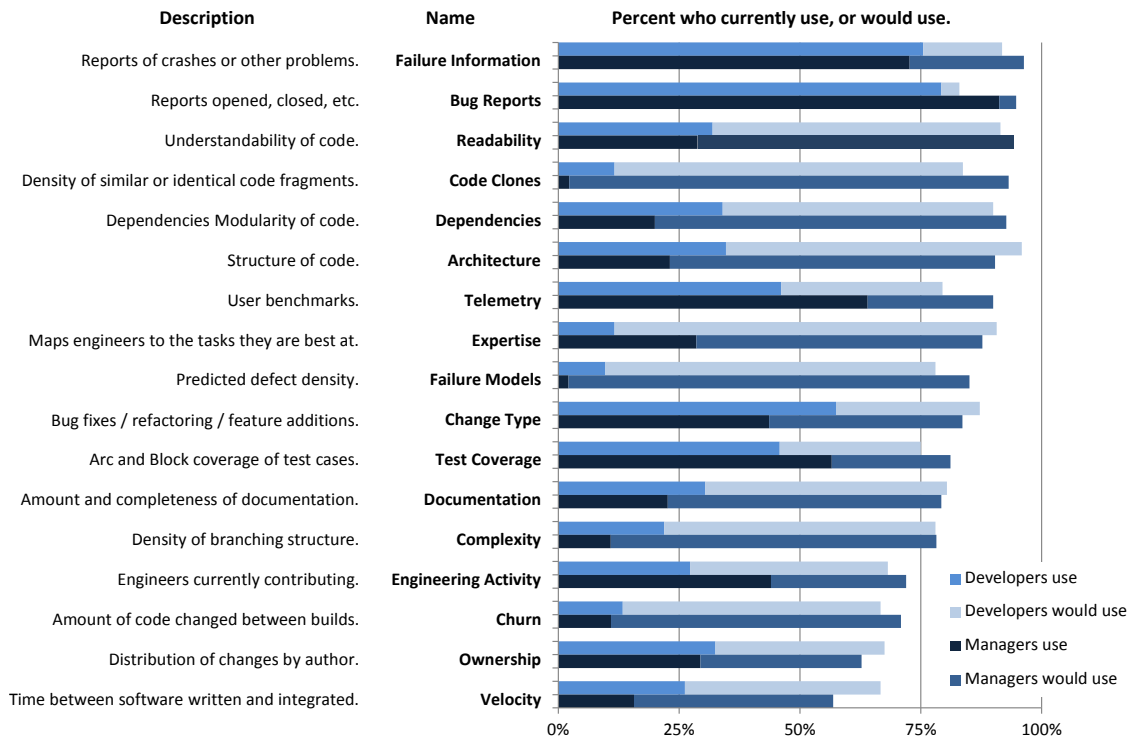

Fig. 5. Percent of managers and developers who reported that they either use or would use (if available) each of the given indicators in making decisions relevant to their engineering process.

In addition to artifacts, we asked study participants about a number of popular indicators (metrics); we asked whether each indicator was available and whether they currently use it, or if they would use it if it was made available. In Figure 5 we show, for each indicator, the fraction of respondents who reported they use or would use the indicator.

Failure information was ranked most important and bug reports second most overall. While failure information refers to crash reports or other information from end users, bug reports are most often created by developers and testers within the development team. Both developers and managers make extensive use of both types of information. However, the next several metrics in order of importance are much less common. Readability, code clones, and dependencies are infrequently used, but 90% of survey participants responded that they would use such indicators if they were available. Furthermore, except for test coverage, change type, and engineering activity, all of the indicators could be used by twice as many developers and managers if they were made available.

Comparing the responses of developers to managers, several indicators suggest important differences in concerns. Managers, for example, are more likely to use indicators related to failures, telemetry, and testing which reveals a stronger focus on customers. Developers, on the other hand, are more interested in code metrics like velocity, churn, readability, and complexity. However, as was the case with artifacts, few indicators would seem to be wholly unimportant to either group; every indicator would be used by at least half of the respondents if available.

While strong interest clearly exists, the difficulty of incorporating such metrics in practice remains. We conjecture that at least part of this difficultly is rooted in disconnect between available tools and real-world decision making. In an effort to understand and ultimately bridge this disconnect, in the next section we develop taxonomy of decision scenarios: circumstances under which these and other indicators might be used in a real development setting.

D. Decision Scenarios

What decisions could analytics help with?

The landscape of artifacts and indicators in software engineering is well known, but the concrete decisions they might support are not. We conjecture that understanding *how* managers and developers can make use of information is critical to understanding *what* information should be delivered to them. To our the best of our knowledge, no previous study has produced a broad-based taxonomy of decision scenarios sufficient for guiding tool design or other research in this area.

Because no preexisting classification exists, we asked each participant to describe up to three scenarios illustrating the actual or potential effect of analytics on their decisions; highlighting the *questions* analytics could answer and *decisions* it could support. Survey participants described a total of 102 total scenarios. For each scenario we identified which subset of the past, present, or future it related to. We enumerated each type of artifact and indicator mentioned, and we categorized the types of decisions each analytical

question was said to assist with. Some common themes emerged:

Targeting Testing. Allocating testing resources is one of the most powerful tools at the disposal of project managers for responding to a variety of conditions. Testing is the go-to solution when it comes to essentially all software correctness issues; it's straightforward to deploy, measurable, and generally effective. Several managers commented on the information needs for test allocation:

> *"Targeting testing of a product needs information on the code that changed from build to build and as a result of bug fixes so we could more easily map out what features and what other code requires re-examination."*
>
> *"More information about the code dependency and churn due to a change or a set of changes also affects the decision made. It goes back to the tradeoff of making the change versus what it means to both the testing team in terms of churn and coverage plus any potential breaks or failures or performance effects the change might have on unforseen components and features."*

Much research exists into software testing, especially into automated tool support (e.g., [17], [22]). Comparatively little is known, however, about how testing effort should be deployed over the life-cycle of a large project. It is not clear whether re-targeting testing based on indicators like code churn, readability, and others is truly effective. In a future where project data is detailed, complete, and ubiquitous an opportunity exists for testing effort to become highly-targeted (e.g., test bug fixes contributed by a given developer and relating to specific feature). More research is needed before the implications of these trends can be fully understood.

Targeting Refactoring. While testing is the primary means for revealing functional defects, refactoring refers to the modification of code that does not impact external functional behavior. Rather, the goals is to improve attributes relating to readability, maintainability, extensibility, and other important non-functional software attributes. Refactoring can be thought of as investment [30]: spending resources now to avoid larger costs in the future. Often indicators related to architectural complexity and code clones are often cited as relevant to targeting refactoring effort [38].

> *"The number of bug reports for a certain feature area helps us decide whether that feature area is mature for a refactoring."*
>
> *"Telemetry allows us to prioritize investment for code cleanup and bug fixing in a way that has substantial customer impact."*

Release Planning. Commercial software projects are under constant pressure to be released quickly. Managers told us that among the most important reasons to monitor a project is to plan such releases and to anticipate risks. Planning consists of determining what features should be included in upcoming releases and when those releases should actually occur [49]. Relevant factors include testing and development progress for each feature, feature dependencies, outstanding defects, as well as external factors like market conditions. Several managers noted the importance of release planning.

> *"Identify features that put ship date and quality at risk to develop mitigation plans."*

Yet, effective release planning involves more than just understanding the progress of a project, it also demands that developers understand their customers.

Understanding Customers. In any business endeavor, understanding customers is important; software development is no exception. Many technologies exist for collecting data from customers: from crash reports [21] to telemetry to surveys [13], [33], [35]. Several decision scenarios described the importance of leveraging information about customer behavior when making decisions about the direction of a project.

> *"Analytics help us understand how a user is using our product. Are they performing tasks we expect? Performing tasks we didn't anticipate? We can determine effectiveness of features, as well."*

Making customer data actionable implies directly relating it to development effort. Not only must we know which features are valuable or problematic, it must also be possible to identify these features in the source code, to track their progress, and to employ customer feedback to guide specific aspects of development and maintenance.

Judging Stability. Stability is a key concept in software development. Many modern software projects are long-lived as evidenced by the observation that maintenance (defined as change subsequent to release) will typically consume as much as 90% of the total life-cycle budget of a project [46], [50]. Yet, not all parts of a project change together or in the same way. Many managers and developers indicated that monitoring the stability of various parts of a project is important. Understanding stability can help managers anticipate future change and can ultimately lead to key decisions about the fate of a system or one of it's components; form the observation that it's time to release to the decision that it must be abandoned.

> *"Using bug / fix / regression metrics over time to understand what areas are not yet stable and how long it might take to make them stable."*

Targeting Training. Despite its technical nature, software development remains primarily a human endeavor. Many aspects of a project can benefit from explicitly considering it as a human artifact, the result of a potentially large-scale collaboration of individuals or teams. Throughout a project's life-cycle some developers leave a project and new ones are added. Managing the intellectual capitol associated with labor turnover is a key concern [40], yet monitoring this resource can be very difficult.

We believe that much data exists in software repositories that can help managers to precisely characterize these

intellectual resources. Suppose a certain developer is about to leave the project. With the proper analytical tools a manager should be able to understand which parts of the projects (features, files, other artifacts, certain skills) may be compromised. Suppose, for example, the leaving developer has found most of the security bugs in the past. The manager can use this information to ensure proper knowledge transfer, recruiting, or institute necessary training so that future security bugs do not go unaddressed.

Targeting Inspection. Finally, it is not uncommon for organizations to make use of code reviews or inspections [18]. Yet such inspection are invariably expensive, involving several developers and often a great deal of time. While under some development regimes changes to certain system components must always be inspected (e.g., mission critical sections), several survey participants discussed how it is often difficult to decide when inspection are needed in the rest of the project.

> *"For our checkin process, if I had decent metrics for software readability, dependency changes, component-level changes, etc., I could help drive decisions on what levels of code review we would force. I'd also like to make comparisons between the results of the analytics and the actual number of issues found during code review (in order to tune the analytics)."*

Analytics techniques and tools hold out the promise of identifying such situations. For one development team, for example, the occurrence of code clones may indicate the need for inspections, while for another reports of security defects may be the most important consideration. Such techniques could even be used to evaluate the success of different types of inspection strategies.

E. Frequency of Decision Scenarios

Figure 6 shows the percentage of scenarios mentioning each decision type. Many managers and developers described how various combinations of metrics and artifacts could be useful for targeting testing effort. Developers were also in particular agreement over the relevance of analytics to targeting refactoring, while managers mentioned a somewhat wider variety of decision types.

Approximately 89% of the scenarios described concern either the past or present, a finding which underscores our earlier observation that participants believe questions pertaining to the past are the most important to answer. The artifacts most often mentioned were features, code artifacts, and change records. Managers discussed features more often than any other artifact: approximately 35% of the scenarios they described implied a mapping between features and code (about twice as often as developers). Similarly, developers described scenarios pertaining to measuring code about twice as often as managers. As for indicators, managers most frequently mentioned bugs, telemetry, test coverage, regressions and churn. Developers mentioned complexity and dependencies more often. This confirms similar findings on the importance of features presented in Section V-C.

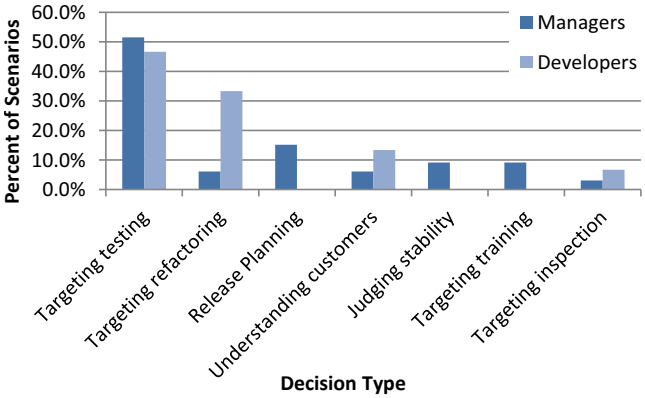

Fig. 6. Percent of scenarios described in study that pertain to each type of decision.

F. Threats to Validity

We now consider briefly whether the results of our study are likely to be valid and generally applicable.

A potential threat to generality is that this study was conducted exclusively with engineers at Microsoft. We believe this threat is mitigated by a number of factors. First, our study is broad in the sense that the participants' work spans many product areas including systems, mobile devices, web applications, and games. Some Microsoft projects are large (as many as 2,000+ engineers) and others are significantly smaller. Specific development methodologies also vary throughout the company. Second, our study is large (involving 110 participants) and diverse (participants worked for companies other than Microsoft in the past for five years on average).

Another consideration is survey bias. To mitigate this threat, our survey was conducted on a random sample of engineers and all responses were kept anonymous. Furthermore, we confirmed many of the survey findings during later face-to-face meetings with managers (Section VII-B).

VI. Software Analytics Guidelines

A large majority of survey participants agreed or strongly agreed with the statement "The difficulty of interpreting data is a significant barrier to the use of analytics today." This implies a need for a new class of tools that specifically target the information needs of managers. Another intriguing possibility is the addition of an analytic professional to the software development team, we discuss this in Section VI-D.

A. Tool Guidelines

Here we employ the insights gathered from our survey to characterize a set of important characteristics for any analytics tool. Furthermore we enumerate a set of analyses targeted to real-world management needs.

First, analytics tool should ...

- Be easy to use. Managers who don't necessarily have expertise in analysis.
- Be fast and produce concise or summary output. Managers have significant time constraints.

	Past	Present	Future
Exploration Find important conditions.	**Trends** Quantifies how an artifact is changing. Useful for understanding the direction of a project. ▪ Regression analysis.	**Alerts** Reports unusual changes in artifacts when they happen. Helps users respond quickly to events. ▪ Anomaly detection.	**Forecasting** Predicts events based on current trends. Helps users make pro-active decisions. ▪ Extrapolation.
Analysis Explain conditions.	**Summarization** Succinctly characterizes key aspects of artifacts or groups of artifacts. Quickly maps artifacts to development activities or other project dimensions. ▪ Topic analysis.	**Overlays** Compares artifacts or development histories interactively. Helps establish guidelines. ▪ Correlation.	**Goals** Discovers how artifacts are changing with respect to goals. Provides assistance for planning. ▪ Root-cause analysis.
Experimentation Compare alternative conditions.	**Modeling** Characterizes normal development behavior. Facilitates learning from previous work. ▪ Machine learning.	**Benchmarking** Compares artifacts to established best practices. Helps with evaluation. ▪ Significance testing.	**Simulation** Tests decisions before making them. Helps when choosing between decision alternatives. ▪ What-if? analysis.

Fig. 7. A spectrum of analyses suitable for comprising the core of an analytics tool for development activities. We describe each technique and the insights it primarily pertains to. Additionally we bullet a related technique for each.

- Measure many artifacts using many indicators. Many are important, and combining them can yield more complete insights.
- Be current and interactive. Managers want to view the most current data available, at many levels of detail, not static reports.
- Map indicators to features and dates to milestones.
- Focus on characterizing the past and present over predicting the future.
- Recognize that managers and developers have different needs and focus on information relevant to the target audience.

B. Common Analysis Types

Additionally, we identify a set of analysis types that fit well with information needs described by our study. In Figure 7 we organize these analyses by what time frame they pertains to (i.e., *Past*, *Present*, *Future*) and their general category of technique (i.e., *Exploration*, *Analysis*, and *Experimentation*). For each analysis, we briefly describe what it does and what insights it can help with. We also give an example of a related technique which might underlie such an analysis. These analyses can be instantiated with any number of metrics; whichever are most appropriate for the target scenario, and can be layered as in Figure 1.

Trends. The nominal value of an indicator is often less important than how it is changing or "trending." Many decision scenarios describe intervening when negative trends are detected. An analytics tool should have the capacity to differentiate significant trends from spurious ones.

Alerts. During the course of a project's life-cycle, it's not unusual for certain events to occur suddenly which should be addressed a quickly as possible; influxes of crash reports, sudden and rapid changes in key indicators, large changes to sensitive system components are all rare but important to address when discovered. The size and complexity of many projects make it important that managers have automated support for detecting these events.

Forecasting. As significant trends and correlations are identified, it can often be useful to project them into the future. Such a tool can help engineers predict when tasks will reach important thresholds (e.g., number of known defects is no longer decreasing). A good analytics tool should help the user understand potential sources of error and confidence bounds for any such projections.

Summarization. Software engineering artifacts can be numerous and complex; inspecting hundreds of change records to discover what they have in common, for example, isn't practical. Summarization techniques like topic analysis can be employed to automate these tasks and help managers and developers focus on gathering high-level insights.

Overlays. The idea of overlays is to present multiple views of a dataset simultaneously, typically with a strong component of interactivity. Overlaying architecture with code clones, for example, might reveal hidden insights about why and how the clones originate. Overlays can also be used across development histories (e.g., overlaying bug reports from last year's release with the current history could help a manager decide if bug triage is more or less effective then it used to be).

Goals. Analytics tools often explicitly represent important goals. By encoding project milestones and other goals managers and team members can explore how actions they take can influence high-level goals in the short an long term.

Modeling. Managers must maintain awareness of how the development is functioning and where it can be improved. In this context, modeling refers to the task of building a representation of the project history or of the development team itself for the purpose of comparison or assessment.

Benchmarking. Managers often expressed the importance of comparing to best practices, the idea of benchmarking is to characterize such practices so that can be referenced automatically. An analytics tool can help find any significant divergence from benchmarks.

Simulation. Managers often perform contingency planning (often cast as "What-if?" decisions), for example: "What if we release in three months?" "What if we abandon feature X?" Simulation can be used to show the eventual real effects of alternative conditions and courses of action.

C. Implications for Research

We would now like to emphasize several implications for research based on the findings from the survey.

- *Diverse information needs.* Both engineers and managers revealed a wide spectrum of information needs in the survey. The needs also change over the lifetime and maturity of the project, i.e., close to a release the needs are different from the needs in the beginning of a project.
- *Many stakeholders.* There are many different stakeholders interested in data with very different needs. For example, a manager wants to see different data than a developer or tester. A direct consequence is that tools should support different views for managers, developers, testers, etc. Probably no tool fits all.
- *One tool is not enough.* Stakeholders will likely need multiple tools. Analytics is more than just measurement and often multiple methods are needed to fully understand data (see Figure 1).

Often teams have very unique questions, which are impossible to anticipate and require experience with data analysis. In the next section, we argue for a dedicated software analyst who is responsible to support data-driven decision making within a project—similar like a build manager is responsible for a successful build process.

D. Software Analysts

An *analyst* is an expert in data creation, collection, interpretation and use, but is also trained in the workings of the business process in question. The role of the analyst is to use quantitative skill and domain knowledge to combine many types of quantitative and qualitative information and form the most complete insights.

Software analysts could be enlisted to perform studies too involved for managers or even for sophisticated tools. For example, Bird *et al.* found that distributed development did not appreciably effect software quality in the implementation of Windows Vista [9]. Another study by Mockus *et al.* enumerated a set of metrics which may be highly predictive of defects [41]. An analyst could carry out similar studies and prescribe corrective action based on the results. Furthermore, many findings in software engineering research depend on large numbers of context variables [6]. As such, these findings may not generalize [42], [59]. A software analyst is critical for determining which important results offered by the research community apply to a specific project.

We believe that analysts could be trained as part of a Software Engineering curriculum modified to emphasize quantitative skills, or through a Master of Analytics program like the one recently launched at North Carolina State University.[2]

In the next section, we return focus on the guidelines established in Section VI by presenting an analytics tool based on those guidelines. A fully fledged software analyst would likely require a different set of tools entirely which is beyond the scope of our study.

VII. Proof-of-Concept Analytics Tool

In this section we describe a proof-of-concept analytics tool which we use to (a) validate the application of the guidelines from Section VI, and (b) demonstrate our ideas to candidate users for the purpose of soliciting additional feedback. The tool was built to concisely explain software development activity over some explicit time window. The tool can be instantiated in the context of whatever indicators or artifacts are available. We would like to emphasize that the **purpose of the tool is to solicit additional feedback on information needs for software development analytics.**

The successful execution of many management responsibilities, from resource allocation to release planning hinges on the ability of a manager to maintain a detailed and complete understanding of the state and direction of their project.

A. Description

An annotated screen shot of our tool is shown in Figure 8. At the core of the tool is an interactive chart view which displays the value of the selected indicator for each date in the current time window. By clicking in the chart the user can zoom in on a particular time frame of interest. The main features of the tool are (1) *Drill-Down* which allows the user to target the analysis across many dimensions, (2) *Summarization* based on latent Dirichlet allocation (LDA) topic analysis of commit messages, and (3) *Anomaly Detection* which identifies artifacts that exhibit unusual behavior during the selected time period. In this subsection we discuss each feature by describing a typical use scenario; technical details are given in Appendix A.

- *Drill-down.* A manager is planning the next release and wants to know which features should be included. She first uses the tool to filter the project history to consider only the part of the project she is responsible for: changes by her direct and indirect reports. The tool then allows her to quickly inspect progress on several features undergoing active development, looking at the chart she can tell which ones have become stable, as they only show a small amount of churn recently. One feature in particular shows several recent peaks. These peaks lead her to believe that this feature may not

[2] http://analytics.ncsu.edu

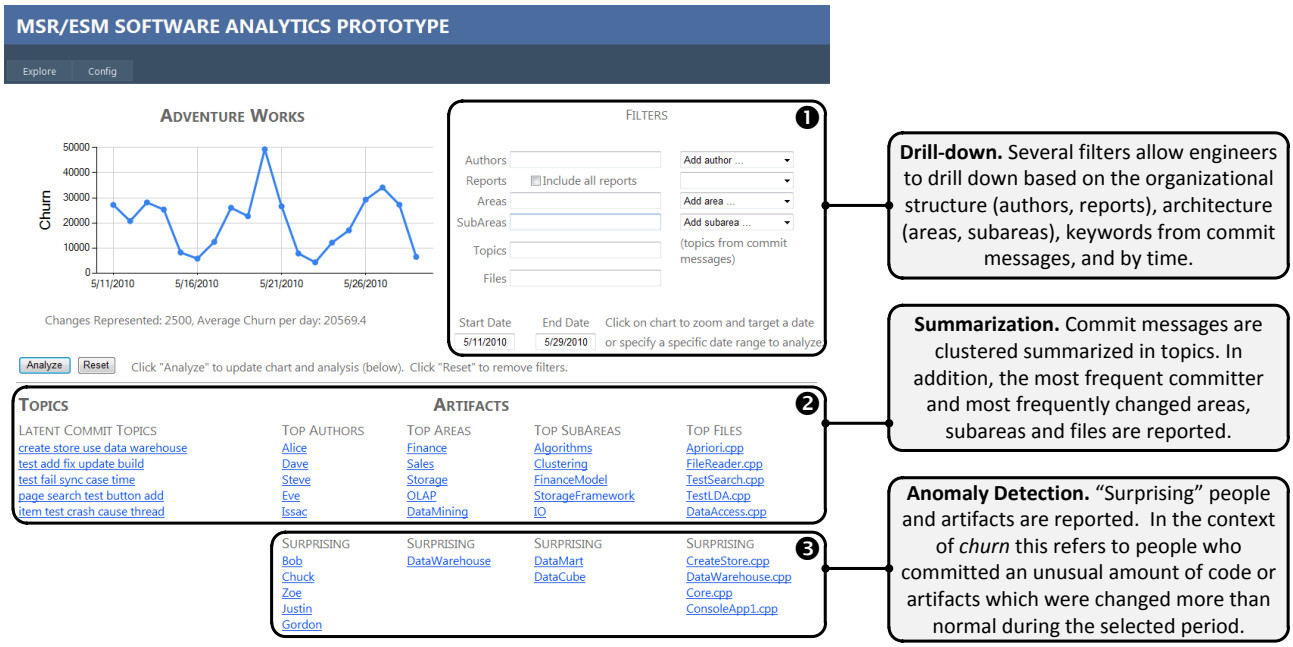

Fig. 8. Our proof of concept analytics tool which features Drill-down, Summarization, and Anomaly Detection.

be very stable. Not only should this feature not be included she decides to focus some additional testing effort on this feature and a few other features that show a similar history.

- *Summarization.* A manager is interested in the progress of a new team member. She begins by filtering changes to that author only. Looking at the analysis generated by the tool she notices that although the new team member has been working in only one area of the product, his work is associated with a very wide variety of topics which suggests to her that this developer needs to have a more focused assignment.
- *Anomaly Detection.* A developer is concerned about development progress relating to a certain feature, he uses the tool to filter changes to this feature only. He then notices that the amount of churn has been increasing in recent weeks and wants to know why. To do that he clicks on the chart and zooms into the time period exhibiting the greatest increase. The tool then shows that changes to a particular sub-area are unusually high and a variety of topics are generated. Clicking on this sub-areas allows him to discover that several files in particular seem to be changing very often. This leads him to target refactoring effort in these files.

B. Feedback

Over the course of two weeks we demoed our tool, and presented our survey results to several Microsoft project managers. We asked them to discuss their information needs and thoughts on tool support. Our study results and interactive tool demo severed the purpose of stimulating discussion on both topics. In this section we present some of the key ideas.

1) Information Needs / Metrics: A number of participants confirmed our finding that, while some metrics are probably more valuable than others on average (Section V-C), most every common metric can be quite useful. Their key observation, however, is that utility hinges on context; information needs can vary quite widely depending on time, schedule, project, and other factors but especially on the user's position within the organizational hierarchy. While developers and their direct managers can make a great use of detailed metrics, managers higher in the hierarchy may find the same metrics to be significantly less actionable. In general, higher level managers prefer more aggregated metrics with opportunities for comparison across development teams and projects.

Furthermore, the value of a metric is determined not just by how often it can inform critical decisions directly, but also by how often it can be used for tracking projects. Code velocity, which is typically used as a proxy for productivity, for example, was described as a metric that doesn't typically trigger decisions, but is of great use for assessing development effort over relatively long periods. This observation alludes to *overlays* and *benchmarking*. Participants described to us a typical use case for some metrics: establish suggested parameters based on past experience (e.g., outstanding bugs should be generally decreasing 3 months before release and should fall within a given range) and then take steps to ensure the project stays on that course.

Finally, participants confirmed our observation on the importance of measuring features. They offered the obser-

vation that features span many parts of a project: they are a focus not only of developers but also managers and even customers. A number of managers noted that our notion of *topics* could serve a similar role. Next we discuss another observation that apply to analytics tools.

2) Analytics Tools: After viewing the tool, many managers reacted positively to our notion of anomaly detection. Some noted that one of the most basic tasks of analytics is simply to identify any metrics for which a given team "sticks out" in comparison to other teams. One reason for this may be that metrics are difficult to interpret in part because it's often unclear what ranges of values for a particular metric are good, and which should be concerning. This is particularly true when a long running project changes direction in some way. When this happens managers often don't know how to evaluate metrics when past experience is not longer applicable. Furthermore, this emphasizes the importance of normalizing metrics (e.g., diving by size of team or size of project) in order to allow the best comparison possible.

Also in relation to anomaly detection, managers asserted a need for detecting "unexpected behavior." An example of this is an author who commits a large change in an area where he or she does not normally work. Managers suggested it would be most useful if such an event generated an immediate alert that could be delivered to key individuals responsible for the feature. Additional customization of such reports might be possible as well, for example, a performance engineer might want to be notified of any change to a particular module that is especially performance sensitive.

Finally, managers told us that in order to best support decision making a good tool should not only summarize or otherwise explain behavior, it should also, on command, produce the actual change records or other artifact descriptions that are often necessary to take action on the analytic findings.

VIII. Conclusion

We believe that analytics holds out great promise for enhancing the ability of software engineers and their managers to make informed decisions. Analytics is a data-driven decision making paradigm that emphasizes the layering of multiple types of analyses on top of existing information resources, typically with the help of automated tools. In this paper we explained why software engineering is a good candidate for this approach. For example, software projects are highly measurable, but often unpredictable.

Realizing analytics for software development demands understanding the information needs of development managers; what their needs are, what decisions they influence, and how those needs map to analyses. Toward that end, we presented a study of the information needs of managers and developers. We then distilled from the study a set of guidelines for constructing analytics tools which we validated with a proof-of-concept implementation. Finally, we discussed feedback on the tool from managers.

We hope this paper serves as a first step toward important new tools which will support software development. Our guidelines may influence new tools, and our proof-of-concept tool can be evaluated with user studies and developed into a production tool. We also hope that inspired by our work other researchers will do additional studies to better understand information needs in software development analytics. To facilitate replication, we provide the full text of the survey in Appendix B.

Future work in the area of software development analytics should fall into the following categories.

- *Data collection.* We need to rethink how we collect data. So far, mining software repositories has had a data-focused approach (i.e., take some data and come up with great tools and insights). However, it will be important to also think in a *user-focused* way: start with the user and decide what data we need to collect to provide her with solutions to her problems.
- *Data quality.* To make decisions based on data, the quality of the data has to be high [8], [2], [44]. But it is important to notice that not all data needs to be perfect. The collection of high-quality data costs money; if data is not used for decisions, its quality matters less—why collect it in the first place.
- *Data privacy.* Data can be very dangerous when used inappropriately. It is important to put in place mechanisms that ensure only proper uses of data are possible.
- *Understanding user needs.* This paper is a first step towards understanding the data needs of software engineers and managers. A next step will be to understand how data is used in their communication. Often people justify decisions to their peers, managers, and reports.
- *User experience.* The user experience of any tool for software development analytics will be critical, e.g., what are the best ways to surface and interact with software development data and analysis.
- *Education.* Finally, as today's society and businesses become more data-driven [36], it will be important to prepare software engineers for data analysis and educate them how to use basic analysis techniques.

We are at the crossroads to become more data-driven in software development. With Web services and the Cloud the amount of data will explode, but also the opportunities gain insight. To make the right decisions during this transition it is important for us to better understand the data and analytics needs. This paper is a first step into this direction.

Acknowledgments: We thank Sunghun Kim, Daniel Liebling, Robin Moeur, Brendan Murphy, Nachi Nagappan, Dongmei Zhang, and the many managers and developers at Microsoft who volunteered their time to participate in our study, meet with us, and provide their valuable insights and feedback.

References

[1] T. Addison and S. Vallabh. Controlling software project risks: an empirical study of methods used by experienced project managers. In *SAICSIT '02*, pages 128–140, 2002.

[2] A. Bachmann, C. Bird, F. Rahman, P. Devanbu, and A. Bernstein. The missing links: Bugs and bug-fix commits. In *SIGSOFT'10/FSE-18: Proceedings of the 16th ACM SIGSOFT Symposium on Foundations of Software Engineering*. ACM, 2010.

[3] V. R. Basili. The experience factory and its relationship to other improvement paradigms. In *ESEC'93: Proceedings of the 4th European Software Engineering Conference*, volume 717 of *Lecture Notes in Computer Science*, pages 68–83. Springer, 1993.

[4] V. R. Basili, G. Caldiera, and H. D. Rombach. Goal, question, metric paradigm. In J. J. Marciniak, editor, *Encyclopedia of Software Engineering Volume 1*, pages 528–532. John Wiley & Sons, 1994.

[5] V. R. Basili, M. Lindvall, M. Regardie, C. Seaman, J. Heidrich, J. Münch, D. Rombach, and A. Trendowicz. Linking software development and business strategy through measurement. *IEEE Computer*, 43:57–65, 2010.

[6] V. R. Basili, F. Shull, and F. Lanubile. Building knowledge through families of experiments. *IEEE Transactions on Software Engineering*, 25:456–473, 1999.

[7] J. T. Biehl, M. Czerwinski, G. Smith, and G. G. Robertson. Fastdash: a visual dashboard for fostering awareness in software teams. In *CHI '07*, pages 1313–1322, 2007.

[8] C. Bird, A. Bachmann, E. Aune, J. Duffy, A. Bernstein, V. Filkov, and P. Devanbu. Fair and balanced? bias in bug-fix datasets. In *ESEC/FSE'09: Proceedings of the the Seventh joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, 2009.

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

[10] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, 2003.

[11] B. Boehm. *Software Cost Estimation with Cocomo II*. Addison Wesley, Boston, MA, 2000.

[12] B. Boehm and R. Ross. Theory-w software project management principles and examples. *IEEE TSE*, 15(7):902 –916, jul 1989.

[13] T. Briggs. How does usage data improve the office user experience? http://blogs.technet.com/b/office2010/archive/2010/02/09/how-does-usage-data-improve-the-office-user-experience.aspx, Feb 2010.

[14] R. P. L. Buse and W. Weimer. Automatically documenting program changes. In *ASE '10: Proceedings of the twenty-fifth IEEE/ACM international conference on Automated software engineering*, 2010.

[15] R. P. L. Buse and W. R. Weimer. A metric for software readability. In *International Symposium on Software Testing and Analysis*, pages 121–130, 2008.

[16] R. P. L. Buse and T. Zimmermann. Analytics for software development. Technical Report MSR-TR-2010-111, Microsoft Research, 2010. 4 pages. Under submission to FSE/SDP Workshop on the Future of Software Engineering. Available at http://research.microsoft.com/apps/pubs/default.aspx?id=136301.

[17] C. Cadar, V. Ganesh, P. M. Pawlowski, D. L. Dill, and D. R. Engler. Exe: Automatically generating inputs of death. *ACM Trans. Inf. Syst. Secur.*, 12:10:1–10:38, December 2008.

[18] J. Cohen, editor. *Best Kept Secrets of Peer Code Review*. Smart Bear Inc., Austin, TX, 2006.

[19] I. D. Coman, A. Sillitti, and G. Succi. A case-study on using an automated in-process software engineering measurement and analysis system in an industrial environment. In *ICSE '09*, pages 89–99, 2009.

[20] T. Davenport, J. Harris, and R. Morison. *Analytics at Work*. Harvard Business School Publishing Corporation, Boston, MA, 2010.

[21] K. Glerum, K. Kinshumann, S. Greenberg, G. Aul, V. Orgovan, G. Nichols, D. Grant, G. Loihle, and G. Hunt. Debugging in the (very) large: ten years of implementation and experience. In *SOSP '09: Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 103–116. ACM, 2009.

[22] P. Godefroid, N. Klarlund, and K. Sen. Dart: directed automated random testing. *SIGPLAN Not.*, 40:213–223, June 2005.

[23] G. Gousios and D. Spinellis. Alitheia core: An extensible software quality monitoring platform. In *ICSE '09*, pages 579–582, 2009.

[24] A. Hindle, M. W. Godfrey, and R. C. Holt. What's hot and what's not: Windowed developer topic analysis. In *ICSM*, pages 339–348, 2009.

[25] IBM Corporation. Jazz. http://www.ibm.com/software/rational/jazz/.

[26] A. Jedlitschka. Evaluation a model of software managers' information needs – an experiment. In *ESEM'10: Proceedings of the Symposium on Empirical Software Engineering and Measurement*, September 2010.

[27] P. Johnson, H. Kou, M. Paulding, Q. Zhang, A. Kagawa, and T. Yamashita. Improving software development management through software project telemetry. *IEEE Software*, 22(4):76 – 85, july-aug. 2005.

[28] R. Kaplan and D. Norton. The balanced scorecard—measures that drive performance. *Harvard Business Review*, page 71, January/February 1992.

[29] A. Kaushik. *Web Analytics 2.0*. Wiley Publishing, 2010.

[30] M. Kim, D. Cai, and S. Kim. An empirical investigation into the role of refactorings during software evolution. In *ICSE '11: Proceedings of the 33th international conference on Software engineering*, 2011. To appear.

[31] A. J. Ko, R. DeLine, and G. Venolia. Information needs in collocated software development teams. In *ICSE'07: Proceedings of the International Conference on Software Engineering*, pages 344–353, 2007.

[32] S. Komi-Sirvi, P. Parviainen, and J. Ronkainen. Measurement automation: Methodological background and practical solutions-a multiple case study. In *IEEE International Symposium on Software Metrics*, page 306, 2001.

[33] P. Koss-Nobel. Data driven engineering: Tracking usage to make decisions. http://blogs.technet.com/b/office2010/archive/2009/11/03/data-driven-engineering-tracking-usage-to-make-decisions.aspx, Nov 2009.

[34] E. Linstead, P. Rigor, S. Bajracharya, C. Lopes, and P. Baldi. Mining concepts from code with probabilistic topic models. In *ASE '07: Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*, pages 461–464, 2007.

[35] S. Lipstein. Designing with customers in mind. http://blogs.technet.com/b/office2010/archive/2009/10/06/designing-with-customers-in-mind.aspx, Oct 2009.

[36] T. May. *The New Know: Innovation Powered by Analytics*. Wiley, 2009.

[37] T. J. McCabe. A complexity measure. *IEEE Trans. Software Eng.*, 2(4):308–320, 1976.

[38] T. Mens and T. Tourwé. A survey of software refactoring. *IEEE Trans. Software Eng.*, 30(2):126–139, 2004.

[39] Microsoft Corporation. Team foundation server. http://msdn.microsoft.com/en-us/vstudio/default.aspx.

[40] A. Mockus. Succession: Measuring transfer of code and developer productivity. In *ICSE'09: Proceedings of the 31st International Conference on Software Engineering*, pages 67–77, 2009.

[41] A. Mockus, P. Zhang, and P. L. Li. Predictors of customer perceived software quality. In *ICSE '05*, pages 225–233, 2005.

[42] I. Myrtveit, E. Stensrud, and M. Shepperd. Reliability and validity in comparative studies of software prediction models. *IEEE Trans. Softw. Eng.*, 31(5):380–391, 2005.

[43] National Institute of Standards and Technology. The economic impacts of inadequate infrastructure for software testing. Technical Report 02-3, Research Triangle Institute, May 2002.

[44] T. H. D. Nguyen, B. Adams, and A. E. Hassan. A case study of bias in bug-fix datasets. In *WCRE '10: Proceedings of the 17th Working Conference on Reverse Engineering*, 2010.

[45] U. D. of Defense and U. Army. Practical software and systems measurement: A foundation for objective project management, v.4.0c, March 2003. www.psmsc.com.

[46] T. M. Pigoski. *Practical Software Maintenance: Best Practices for Managing Your Software Investment*. John Wiley & Sons, Inc., 1996.

[47] T. Punter. What information do software engineering practitioners need? In *The Future of Empirical Studies in Software Engineering. Proceedings of the ESEIW 2003 Workshop on Empirical Studies in Software Engineering*, pages 85–95, 2003.

[48] A. Rainer, T. Hall, and N. Baddoo. Persuading developers to 'buy into' software process improvement: Local opinion and empirical evidence. In *ISESE'03: Proceedings of International Symposium on Empirical Software Engineering*, pages 326–335, 2003.

[49] G. Ruhe. *Product Release Planning: Methods, Tools and Applications*. Auerbach Publications, 2010.

[50] R. C. Seacord, D. Plakosh, and G. A. Lewis. *Modernizing Legacy Systems: Software Technologies, Engineering Process and Business Practices*. Addison-Wesley Longman, MA, USA, 2003.

[51] J. Sillito, G. C. Murphy, and K. De Volder. Questions programmers ask during software evolution tasks. In *SIGSOFT '06/FSE-14*, pages 23–34, 2006.

[52] A. Sillitti, A. Janes, G. Succi, and T. Vernazza. Collecting, integrating and analyzing software metrics and personal software process data. In *EUROMICRO*, page 336, 2003.
[53] S. Sinofsky. Steven sinofsky's microsoft techtalk / pm at microsoft. http://blogs.msdn.com/b/techtalk/archive/2005/12/16/504872.aspx, Dec. 2005.
[54] L. Strigini. Limiting the dangers of intuitive decision making. *IEEE Software*, 13:101–103, 1996.
[55] C. Treude and M.-A. Storey. Awareness 2.0: staying aware of projects, developers and tasks using dashboards and feeds. In *ICSE '10*, pages 365–374, 2010.
[56] J. D. Valett and F. E. McGarry. A summary of software measurement experiences in the software engineering laboratory. *Journal of Systems and Software*, 9(2):137 – 148, 1989.
[57] S. Vegas, N. J. Juzgado, and V. R. Basili. Packaging experiences for improving testing technique selection. *Journal of Systems and Software*, 79(11):1606–1618, 2006.
[58] L. Wallace, M. Keil, and A. Rai. Understanding software project risk: a cluster analysis. *Inf. Manage.*, 42(1):115–125, 2004.
[59] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, and B. Murphy. Cross-project defect prediction. In *Symposium on the Foundations of Software Engineering*, August 2009.

APPENDIX A
IMPLEMENTATION DETAILS

This subsection describes some of technical details of the proof-of-concept tool that we used to solicit additional feedback on information needs and guidelines for software development analytics in Section VII.

As input for the tool we used data from the version control system and bug database of a large Microsoft project.

A. Drill-down

In addition to narrowing the time window, the tool allows the user to narrow the analysis by *authors*, *areas*, *sub-areas*, *files*, and *topics*. *Authors* refers to the individual responsible for the commit, but is additionally parametrized by *Reports* allowing the user to quickly identify all authors below a certain author in an organization hierarchy. A manager, for example, can quickly narrow analysis to exactly those individuals who report to him or her. *Areas* and *Subareas* refer to project domains which roughly correspond to product features; we obtain the information on areas and subareas from the bug reports linked to a commit. *Files* simply refer to source code files. *Topics* are terms in commit messages and correspond to LDA topic analysis which we discuss next.

B. Summarization

As a most basic, but important, form of summarization the tool displays the "Top" (i.e., most active) artifacts in the sample. However, this only shows what artifacts are changing, and not what about them is being changed. Latent Dirichlet allocation (LDA) is an existing statistical technique for clustering documents and extracting sets of common words [10], in effect describing sets of documents by displaying the most common "topics." LDA was applied to commit messages by Linstead *et al.* [34], and later the technique was refined by consider specific change windows by Hindle *et al.* [24] which is similar to our application. In these cases LDA was used to generate static reports, whereas our approach is interactive. We also incorporate word stemming and a stop word list consisting of a number of generic software engineering terms.

C. Anomaly Detection

The problem of anomaly detection here is equivalent to the question of whether some sample of change records is likely to be drawn from a particular distribution which we refer to as the context distribution. For example, if the sample consists of changes made on August 1st through 5th, the context might be all changes this year *except* those between August 1st and 5th. Intuitively, if the context has a high variance, then a sample is not likely to be surprising. On the other hand, if the context has a low variance (i.e., development activity is consistent in some dimension) then a sample distribution with a significantly different mean may be "surprising" and worth inspection.

To estimate this probability we use the standard probability density function which returns the probability that a random variable x occurs at a given point in a distribution parametrized by μ and σ. In this case, $\mu =$ context mean, $\sigma =$ context standard deviation and x is the mean of the sample.

$$Z(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2} \quad (1)$$

$$Surprise(x) = \frac{|Z(x) - Z(\mu)|}{Z(\mu)} \quad (2)$$

Equation 1 assumes the context data is normally distributed which may not be true in many cases (e.g., many software artifacts typically exhibit long-tailed power law distributions [14]). However, we are able to use it as a reasonable characterization because the final value is not used directly but instead normalized (Equation 2) and used as a ranking criteria which is not sensitive to absolute differences, only relative ones.

APPENDIX B
SURVEY

To enable replication, we provide the complete questionnaire in this section. We will be happy to share non-confidential responses on a per request basis.

This survey is anonymous.

Background Information. Analytics refers to the use of analysis, data, and systematic reasoning to make decisions. The purpose of this survey is to help us understand your needs in the context of analytics.

1) How many years have you worked in the software industry?
   Not Required - (Min Number: 0 - Max Number: 100)
2) How many years have you worked at Microsoft?
   Not Required - (Min Number: 0 - Max Number: 100)
3) Which best describes your work area? *Not Required*
   - DEV - Development
   - TEST - Test
   - PM - Program Management
   - Build
   - Design and UX
   - Documentation and Localization
   - Other
4) Which best describes your work role? *Not Required*
   - Individual Contributor
   - Lead
   - Manager
   - Other
5) About how many hours per week do you work on engineering activities?
   Not Required - (Min Number: 0 - Max Number: 168)
6) About how many hours per week do you work on testing activities? *Not Required - (Min Number: 0 - Max Number: 168)*
7) About how many hours per week do you work in a management capacity?
   Not Required - (Min Number: 0 - Max Number: 168)
8) How many people do you directly or indirectly manage?
   Not Required - (Min Number: 1 - Max Number: 100000)
9) In general, how important are these factors to your work related decision making process?
   Not Required - (Very important; Important; Somewhat important; Not important; Unsure)

- Data / Metrics
- Personal experience
- Intuition
- Customer input
- Product vision or value propositions
- Team planning
- One-on-one's with managers
- Opinions of others

Software Indicators. Software indicators include metrics that may be useful for analytical decision making.

10) Which of the following indicators do you currently use or would use, if available? (Part 1)
Not Required - (Available / Use Currently; Available / Do not use; Not Available / Would Use; Not Available / Would not use; Not Sure)
- Churn - Amount of code changed between builds
- Velocity - Time between software written and integrated into main build.
- Engineering Activity - Engineers currently contributing.
- Change Type - Bug fixes / refactoring / feature additions.
- Ownership - Distribution of changes by author.
- Expertise - Maps engineers to the tasks they are best at.

11) Which of the following indicators do you currently use or would use, if available? (Part 2)
Not Required - (Available / Use Currently; Available / Do not use; Not Available / Would Use; Not Available / Would not use; Not Sure)
- Documentation - Amount and completeness of documentation.
- Readability - Understandability of code.
- Dependencies - Modularity of code.
- Architecture - Structure of code.
- Code Clones - Density of similar or identical code fragments.
- Complexity - Density of branching structure.

12) Which of the following indicators do you currently use or would use, if available? (Part 3)
Not Required - (Available / Use Currently; Available / Do not use; Not Available / Would Use; Not Available / Would not use; Not Sure)
- Bug Reports - Reports opened, closed, etc.
- Test Coverage - Arc and Block coverage of test cases.
- Failure Information - Report of crashes or other problems.
- Predicted Defect Density - Failure models.
- Telemetry - User benchmarks (e.g., SQM, RAC, OCA, Watson).

13) What other indicators do you currently use or would you like to use? *Not Required - (Max Characters: 2000)*

14) Software analytics can be performed at many levels of granularity. How important is it for you to use analytics at each of the following artifact levels?
Not Required - (Very Important; Important; Somewhat Important; Not Important; Not Sure)
- Function level
- Class level
- Binary level
- Feature level
- Bug Report level
- Test Case level
- Author level
- Team level
- Component level
- Product level

Scenarios for Analytics. In the boxes provided, please briefly describe up to three scenarios illustrating the actual or potential effect of analytics on your decisions. Please be as specific as possible with respect to questions analytics can help you answer and decisions analytics can support.

15) Scenario 1
Not Required - (Max Characters: 2000)

16) Scenario 2
Not Required - (Max Characters: 2000)

17) Scenario 3
Not Required - (Max Characters: 2000)

Questions in Analytics Analytics distinguishes between questions of *information* and of *insight*.

We're interested in how you perceive the importance and the difficulty of answering the following set of general questions for your work; some are of information and some are of insight.

18) In the context of software projects that you work on, how *important* are the following questions to answer?
Not Required - (Very Important; Important; Somewhat Important; Not Important; Not Sure)
- What happened?
- Why did it happen?
- What is happening now?
- What's the next best action?
- What will happen?
- What's the best / worst that could happen?

19) In the context of software projects that you work on, how *difficult* are the following questions to answer?
Not Required - (Very Difficult; Difficult; Somewhat Difficult; Not Difficult; Not Sure)
- What happened?
- Why did it happen?
- What is happening now?
- What's the next best action?
- What will happen?
- What's the best / worst that could happen?

Analytic Tool Features. Please consider the follwoing four analytic tool features (questions XX - YY). Each feature presents a data view parameterized by indicators, artifacts, and models. Some examples are given.

Which features would be useful to you? Please judge the utility generally, and not just the specific examples shown.

20) Trends - Characterize changes in indicators over time.
Example: "The maintainability index of a.sys is decreasing 5% per week." Example: "The test coverage of module M is increasing 10% per day." *Not Required*
- Very Useful
- Useful
- Somewhat Useful
- Not Useful
- Not Sure

21) Forecasting - Extrapolate indicator values into the future.
Example: "In 3 months, 100% of code changes to the project will be bug fixes." Example: "In 1 week, feature F will be 50% complete." *Not Required*
- Very Useful
- Useful
- Somewhat Useful
- Not Useful
- Not Sure

22) Benchmarking - Compare indicators across artifacts.
Example: "Development Team A commits 500 lines of code per day on average; 30% more than the average team." Example: "a.sys has increased in size by 1500 lines since the last release; 150% more than the median file."
Not Required
- Very Useful
- Useful
- Somewhat Useful

- Not Useful
- Not Sure

23) Alerts - Get automatic notifications of unusual indicator values or trends.
Example: "The number of open bugs increased by 10% this week; 80% more than normal." Example: "Author A writes code with a 15% rate of code clones; 10% more than the average contributor to component C." *Not Required*
   - Very Useful
   - Useful
   - Somewhat Useful
   - Not Useful
   - Not Sure
24) Are there other features you would find useful? (Please be descriptive) *Not Required - (Max Characters: 2000)*
25) Do you agree with the following statement: "The difficulty of interpreting data is a significant barrier to the use of analytics today"? *Not Required*
   - Strongly Agree
   - Agree
   - Neutral
   - Disagree
   - Strongly Disagree
   - Not Sure

Conclusion

26) Do you have any final comments about this survey or about the topic of analytics? *Not Required - (Max Characters: 2000)*
27) Overall, how relevant was this survey to you? *Not Required*
   - Very Relevant
   - Relevant
   - Somewhat Relevant
   - Not Relevant
   - Not Sure