Information Needs for Software Development Analytics

Raymond P.L. Buse
The University of Virginia, USA
buse@cs.virginia.edu

Thomas Zimmermann
Microsoft Research
tzimmer@microsoft.com

***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.

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., [1], [2]) 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 [3]. Moreover, software defects cost the US economy many billions of dollars each year [4]. A likely contributing factor is that there continues to be a substantial disconnect between the information and insights needed by project managers to make good decisions and that which is typically available to them.

When information needs are not met, either because tools are unavailable, too difficult to use, too difficult to interpret, or 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 [5]. 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., [6], [7], [8]), 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.

The main contributions of this paper are:

- A study of the information needs of 110 professional software engineers and managers in the context of analytical decision making (Section IV).
- A set of guidelines for software analytics. We present a characterization of analytics problems in software development based on our study (Section VII).

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

II. Related work

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.

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" [9]. 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 [10]. 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." Some other studies exist that identified information needs for managers for specific decision-making tasks. Jedlitschka [11] 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.* [12] identified questions that decision makers have when choosing among different testing techniques. Punter [13] 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. Komi-Sirvio *et al.* noted that software managers are often too busy with their day-to-day duties to spend much time performing measurement activities [14]. Typically data-driven tasks are relegated to secondary work. Rainer *et al.* [15] 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 IV-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 [16] 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 [17], Balanced Scorecard (BSC) [18], and Practical Software Measurement [19]. 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 and can be the key to "solving the 'Why' puzzle" [20].

Basili *et al.* [21] 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.

There are a number of existing tools designed to support management. For example, *PROM* [22] and *Hackystat* [23] are both capable of monitoring and reporting a great number of software project statistics. Microsoft's *Team Foundation Server* [24], IBM's *Jazz* developer environment [25], and the open-source community *Ohloh.Net* for example, provide *dashboard* views designed to keep developers up-to-date on the status of various events like modifications, bugs, and build results. 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 often use them.

In a previous position paper we summarized some of our early ideas on software development analytics [26]. With this paper we make several novel contributions that are not in the position paper: a study of the information needs of software engineers and managers (Section IV and V), guidelines for software analytics (Section VII).

III. Software Analytics

Analytics has revolutionized decision making across many fields [27]. 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 [20]. 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 managers ask about their projects. The goal of analytics to assist decision makers in extracting important information and insights that would otherwise be hidden.

When using flat measurements only, it's difficult for managers to glean more than sparse insights. However, 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 (see our technical report [28] for a full discussion of the qualities): data rich; labor intensive; dependence on timing, consistency and control as well as distributed decision making; low success rate; large variability.

There are also many potential advantages to the application of analytics to software project management (again see our technical report [28]). Analytics can help: monitor a project; know what's really working; improve efficiency; manage risk; anticipate changes; evaluate past decisions.

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.

IV. 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 VII we characterize the results of the study in terms of an information needs spectrum and describe corresponding analytical solutions.

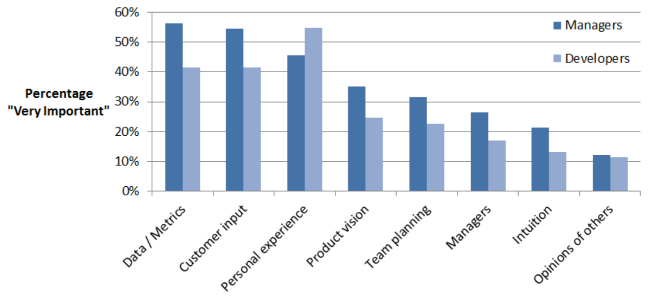

Figure 1. Reported importance of factors to decision making amongst developers and managers.

A. Methodology

For the design of the survey, we followed Kitchenham and Pfleeger's guidelines for personal opinion surveys [29]. 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% of managers.

B. Analytical Questions

What factors 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

[1]The questions in the paper are edited for brevity. The original questions are more specific and listed in a technical report [28, Appendix 2].

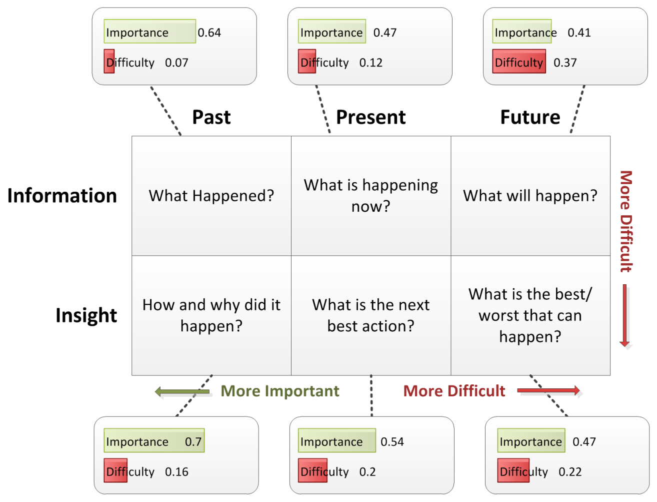

Figure 2. Analytical Questions (adapted from Davenport et al. [27]): We distinguish between questions of *information* which can be directly measured, from questions of *insight* which arise from a careful analytic analysis and provide managers with a basis for action.

of such factors on a 4-point scale {Not Important; Somewhat Important; Important; Very Important}. For the analysis of the responses, we followed the advice by Kitchenham and Pfleeger [29] and dichotomized the ordinal scale to avoid any scale violations. More specifically, we report the percentage of the response “Very Important” among all responses for each factor (excluding responses that had no opinion) in Figure 1.

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 p-value < 0.02). One possible implication is that managers have learned to rely less on their experience 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.* [27] identify six question areas analytics can help answer organized by time-frame and by information vs. insight. We

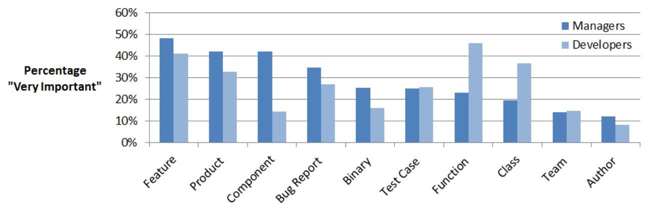

Figure 3. Reported importance of measuring the given types of artifacts amongst developers and managers.

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. Again following the advice by Kitchenham and Pfleeger [29], we report the percentage of the response “Very Important” (for importance) and “Very Difficult” (for difficult) among all responses (excluding no opinion) in Figure 2.

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.”

A 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. Transparency is important for critical decisions.

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 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

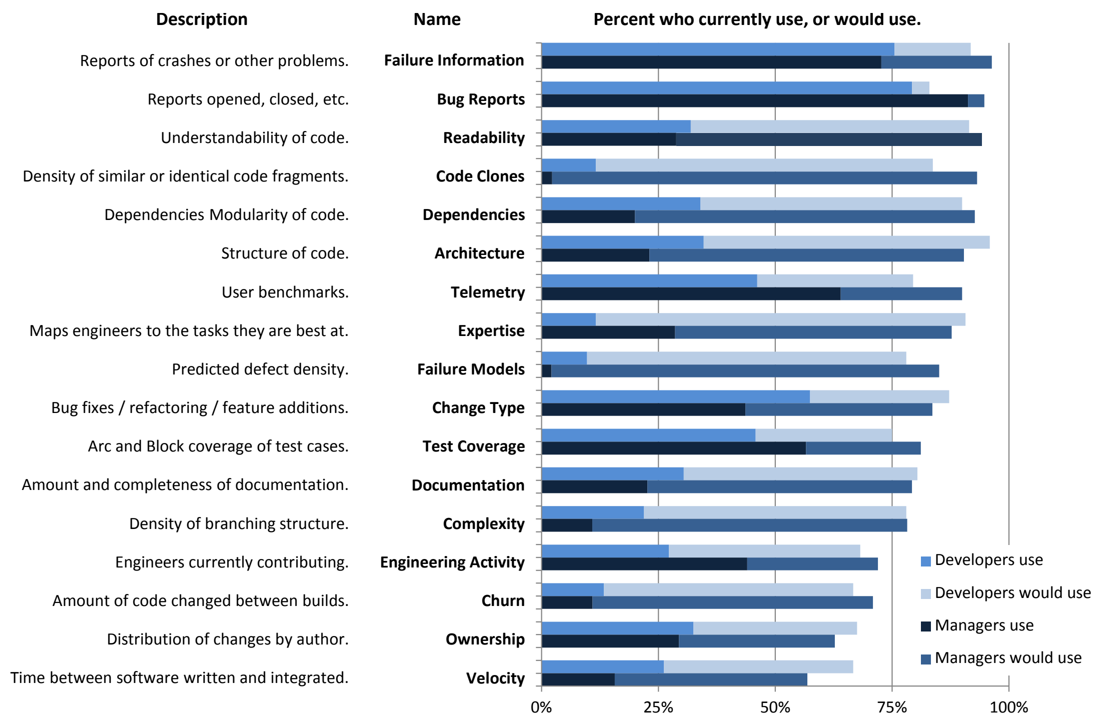

Figure 4. Percent of managers and developers who reported that they either use or would use (if available) each of the given indicators in making decisions relevant to their engineering process.

to rate the importance of a number of artifacts on a 4-point scale, Figure 3 shows how they responded on average. Again we report the percentage of the response "Very Important" among all responses (excluding no opinion).

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). 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.

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

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 4 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.

V. Decision Scenarios for Analytics

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.

What decisions could analytics help with?

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 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."*

Much research exists into software testing, especially into automated tool support (e.g., [30], [31]). 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 is truly effective. In a future where project data is detailed, complete, and ubiquitous an opportunity exists for testing effort to become highly-targeted.

Targeting Refactoring. Refactoring refers to the modification of code that does not impact external functional behavior; rather, the goal is to improve attributes relating to readability, maintainability, extensibility, and other important non-functional software attributes. Refactoring can be thought of as investment [32]: 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 [33].

> *"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 [34]. Relevant factors include testing and development progress for each feature, feature dependencies, outstanding defects, as well as external factors like market conditions. 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 [35] to telemetry to surveys [36], [37], [38]. Several 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 and maintenance (defined as change subsequent to release) will often consume as much as 90% of the total life-cycle budget of a project [39], [40]. 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.

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 [41], yet monitoring this resource can be very difficult.

Targeting Inspection. Finally, it is not uncommon for organizations to make use of code reviews or inspections [42]. Yet such inspections 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 is 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.

Frequency of Decision Scenarios

Figure 5 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 IV-C.

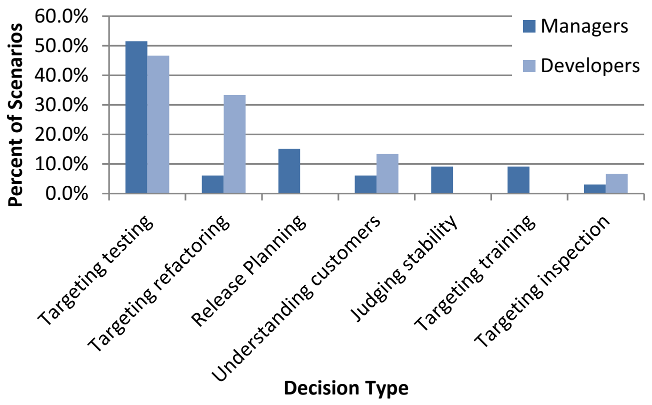

Figure 5. Percent of scenarios described in study that pertain to each type of decision.

VI. 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 (as discussed in our technical report [28]). Other considerations include the Hawthorne effect; survey responders may modify their responses simply because they are aware they are being studied. For example, they may be more eager to agree about the potential utility of new tools even though they would be unlikely to use them in practice. We mitigate this threat somewhat by asking responders to describe concrete usage scenarios (Section V).

VII. 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, which we proposed in an earlier position paper [26].

	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.

Figure 6. 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.

A. *Tool Guidelines*

Here we employ the insights gathered from our survey to characterize a set of important characteristics for any analytics tool. Analytics tools should . . .

- Be easy to use. Managers don't necessarily have expertise in analysis.
- Be fast and produce concise or summary output. Managers have significant time constraints.
- 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 6 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.

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. Manually inspecting hundreds of change records to discover what they have in common 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 could help a manager decide if bug triage is as effective then it used to be).

Goals. Analytics tools often explicitly represent important goals. By encoding project milestones and other broad concerns, managers and team members can explore how the actions they take may influence their goals.

Modeling. Managers must maintain awareness of how 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 now 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. Needs also change over the lifetime and maturity of the project.
- *Many stakeholders.* Different stakeholders may have 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 need multiple tools. Analytics is more than just measurement and often multiple methods are needed to fully understand data.

Often teams have unique questions which are impossible to anticipate and require experience with data analysis. In a previous position paper [26], we argued 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.

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.

We hope this paper serves as a first step toward important new tools which will support software development. 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 and the design of a proof-of-concept tool in a technical report [28, Appendix 2].

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 [43], [44], [45]. 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 [46], 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 data and analytics needs. This paper is a first step in 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] R. P. L. Buse and W. R. Weimer, "A metric for software readability," in *ISSTA '08*, 2008, pp. 121–130.

[2] T. J. McCabe, "A complexity measure," *IEEE Trans. Software Eng.*, vol. 2, no. 4, pp. 308–320, 1976.

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

[4] National Institute of Standards and Technology, "The economic impacts of inadequate infrastructure for software testing," Research Triangle Institute, Tech. Rep. 02-3, May 2002. [Online]. Available: http://www.nist.gov/director/prog-ofc/report02-3.pdf

[5] L. Strigini, "Limiting the dangers of intuitive decision making," *IEEE Software*, vol. 13, pp. 101–103, 1996.

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

[7] A. J. Ko, R. DeLine, and G. Venolia, "Information needs in collocated software development teams," in *ICSE'07*, 2007, pp. 344–353.

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

[9] B. Boehm and R. Ross, "Theory-w software project management principles and examples," *IEEE TSE*, vol. 15, no. 7, pp. 902 –916, jul 1989.

[10] L. Wallace, M. Keil, and A. Rai, "Understanding software project risk: a cluster analysis," *Inf. Manage.*, vol. 42, no. 1, pp. 115–125, 2004.

[11] A. Jedlitschka, "Evaluating a model of software managers' information needs – an experiment," in *ESEM'10*, September 2010.

[12] S. Vegas, N. J. Juzgado, and V. R. Basili, "Packaging experiences for improving testing technique selection," *Journal of Systems and Software*, vol. 79, no. 11, pp. 1606–1618, 2006.

[13] T. Punter, "What information do software engineering practitioners need?" in *Proc. of ESEIW Workshop on Empirical Studies in Software Engineering*, 2003, pp. 85–95.

[14] S. Komi-Sirvi, P. Parviainen, and J. Ronkainen, "Measurement automation: Methodological background and practical solutions-a multiple case study," in *METRICS'01: IEEE International Symposium on Software Metrics*, 2001, p. 306.

[15] A. Rainer, T. Hall, and N. Baddoo, "Persuading developers to 'buy into' software process improvement: Local opinion and empirical evidence," in *ISESE'03*, 2003, pp. 326–335.

[16] V. R. Basili, G. Caldiera, and H. D. Rombach, "Goal, question, metric paradigm," in *Encyclopedia of Software Engineering Volume 1*, J. J. Marciniak, Ed. John Wiley & Sons, 1994, pp. 528–532.

[17] 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*, vol. 43, pp. 57–65, 2010.

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

[19] 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.

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

[21] V. R. Basili, "The experience factory and its relationship to other improvement paradigms," in *ESEC'93*, 1993, pp. 68–83.

[22] A. Sillitti, A. Janes, G. Succi, and T. Vernazza, "Collecting, integrating and analyzing software metrics and personal software process data," in *EUROMICRO*, 2003, p. 336.

[23] P. Johnson, H. Kou, M. Paulding, Q. Zhang, A. Kagawa, and T. Yamashita, "Improving software development management through software project telemetry," *IEEE Software*, vol. 22, no. 4, pp. 76 – 85, july-aug. 2005.

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

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

[26] R. P. Buse and T. Zimmermann, "Analytics for software development," in *Proceedings of the FSE/SDP Workshop on the Future of Software Engineering Research*, November 2010.

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

[28] R. P. L. Buse and T. Zimmermann, "Information needs for software development analytics," Microsoft Research, Tech. Rep. MSR-TR-2011-8, 2010, available at http://research.microsoft.com/apps/pubs/default.aspx?id=144543.

[29] B. Kitchenham and S. Pfleeger, "Personal opinion surveys," in *Guide to Advanced Empirical Software Engineering*. Springer, 2007.

[30] 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.*, vol. 12, pp. 10:1–10:38, December 2008.

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

[32] M. Kim, D. Cai, and S. Kim, "An empirical investigation into the role of refactorings during software evolution," in *ICSE '11*, 2011.

[33] T. Mens and T. Tourwé, "A survey of software refactoring," *IEEE Trans. Software Eng.*, vol. 30, no. 2, pp. 126–139, 2004.

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

[35] 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: Proc. Symp. Operating Systems Principles*, 2009, pp. 103–116.

[36] 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.

[37] 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.

[38] 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.

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

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

[41] A. Mockus, "Succession: Measuring transfer of code and developer productivity," in *ICSE'09*, 2009, pp. 67–77.

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

[43] 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*, 2009.

[44] A. Bachmann, C. Bird, F. Rahman, P. Devanbu, and A. Bernstein, "The missing links: Bugs and bug-fix commits," in *SIGSOFT'10/FSE-18*. ACM, 2010.

[45] T. H. D. Nguyen, B. Adams, and A. E. Hassan, "A case study of bias in bug-fix datasets," in *WCRE '10*, 2010.

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