Applying Data Science to Software Engineering: A summary on how data science helps software development

Shrenuj Gandhi
Computer Science
North Carolina State University
sgandhi4@ncsu.edu

Neng Jiang
Computer Science
North Carolina State University
njiang@ncsu.edu

***Abstract*— This electronic document is a live template. The various components of your paper [title, text, heads, etc.] are already defined on the style sheet, as illustrated by the portions given in this document.**

***Index Terms*— This electronic document is a live template. The various components of your paper [title, text, heads, etc.] are already defined on the style sheet, as illustrated by the portions given in this document.**

I. INTRODUCTION

Data is everywhere and with the computing power increased in the past few years, data science and analytics have become important topics of investigation. source [May, T. The New Know: Innovation Powered by Analytics. Wiley, 2009.]

Amit Valia, in his talk, mentions that we in the Data 2.0 world where we perform data warehousing and analytics on the data to extract value from it. "The world we live in, is the world of data. 25-30 years ago, we were in Data 1.0 world, where the focus was on automating business and development processes via applications. Data was just the output of those applications and there was nothing strategic about it. A few years ago we phased into Data 2.0 world, where data warehousing and analytics came along and we now extract value from that data." source [https://www.youtube.com/watch?v=EIyxdG91XBM]

SE is all data rich activities. Aspects of development generating data. decisions are relied on data. 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? While there has been significant research into information needs of developers [6][7][8], the needs of managers, those who make the important decisions about the future of projects, are not well understood.[Buse, R.P.L. and Zimmermann, T. Information needs for software development analytics. In ICSE '12: Proceedings of 34th International Conference on Software Engineering (2012), 987-996.]

The rest of the paper is organized as follows. Section II discusses related work which summarizes research done in this area during the period 2008 - 2012. Section III describes the baseline study and results conducted in the paper titled "Information Needs for Software Development Analysis". source [Buse, R.P.L. and Zimmermann, T. Information needs for software development analytics. In ICSE '12: Proceedings of 34th International Conference on Software Engineering (2012), 987-996]. Section IV presents current work in this domain covering research from 2013 to 2016. Section V describes threats to validity, and in Section VI we provide our commentary. Section VII discusses future scope in this domain and Section VIII summarizes this paper.

II. RELATED WORK

A. *C. Bird, A. Bachmann, Fair and Balanced? Bias in Bug-Fix Datasets, 2009*

Software engineers have long been interested in where and why bugs occur in code and in predicting where they might turn up next. Researchers have used bug tracking systems and code version histories as source of bug data to test their hypothesis and build statistical bug prediction models. But Bird and Bachmann have questioned the integrity of the bug fixes records/datasets in 2009 and asked are they fair representation of the full population of bug fixes?

In their research, they include hypothesis testing and the following three biases:

- Sampling Bias: When the data gathered does not represent the full population the sampling bias is said to be introduced. Sampling bias makes truly random sample impossible.
- Commit Feature Bias: It is likely that certain types of commit features are systematically over-represented or under-represented among the linked bugs. This representation is termed as commit feature bias.
- Bug Feature Bias: It is likely that the properties of linked bugs look just like the properties of all fixed bugs. The authors define bug feature bias as over or under representing the linked bugs in the sample.

The authors state processes and humans are imperfect. Historical datasets are used to test hypothesis concerning processes of bug introduction and also to build statistical bug prediction models. The claim is that only a fraction of bugs are actually labelled in those data set and the same goes for code commits. And prediction made from samples can be wrong if the samples are not representative of the population. This threatens the validity and effectiveness of the derived hypothesis and models. The related work shows that bias is big consideration in hypothesis testing. Data quality also differs from project to project. Different projects have different methods of defect handling.

The authors have used Eclipse and AspectJ bug data set from the University of Saarland. Apart from this, they gathered data for five projects: Eclipse, NetBeans, the Apache Webserver, OpenOffice and GNOME. They also mined bud databases: Bugzilla and Issuezilla.

The authors have described patterns for data retrieval and preprocessing. They extracted change histories from source code management system (SCM) commit logs. Regarding bugs, they extracted information related to opening, closing, reopening, assignment, severity, comments, time and initiator of every event. The authors also adopted Fisher et al.s technique of finding links between commit and bug report. The technique involves searching the commit log messages for valid bug report references. But the authors made several changes to decrease the number of false negative links. (see steps mentioned on page 5) To validate the data retrieved, the authors manually scanned a random sample for false positives and false negatives.

Result and Conclusion:

- Experiment on BugCache, a defect prediction model, shows how a biased training set can affect the prediction. As a baseline the authors performed training and evaluating w.r.t. bug severity on the entire set of linked bugs. This gave them a recall of around 90% for all categories. But when they train BugCache on a baised training set the evaluations (recall value) are skewed.
- Thus severity bug feature bias is affecting the performance of BugCache in this case.

As a conclusion of the paper, predictions made from samples can be wrong, if the samples are not representative of the population. Data generation process, in this case defect handling, varies among different projects.

B. ***V. Basili, M. Lindvall, Linking Software Development and Business Strategy through Measurement, 2010***

The author mainly talks about using the GQM+ (goals, question, metric) paradigm to measure the success of goals and strategies on all organizational levels.

This strategy helps to decide when and how to transform goals into operations and how to evaluate the success of strategies with respect to those goals.

Steps to use this model:

- The starting point of GQM+ Strategies is a business goal. Along with this, it requires explicit documentation of relevant context factors and assumptions necessary for understanding and evaluating goals. Use the business goal template for documentation.
- The next step is to ask questions necessary to interpret the goal.
- The questions will yield metrics required to measure the goal and an interpretation that provides the information whether the goal is achieved or not.
- Finally, list all the possible strategies and select one for implementation.
- At the next level down in the model, a software development goal is derived from the strategy.
- Steps 2-4 are repeated for this level.
- Similarly, project level goal is derived and Steps 2-4 are followed

1) Keywords:

- Business Alignment: This phrase is used to talk about aligning the business goal with software development strategies.
- Capability Maturity Model Integration (CMMI): CMMI is a process improvement training and appraisal program. CMMI models provide guidance for developing or improving processes that meet the business goals of an organization.
- MoSCoW: It is a prioritization approach for requirements and release planning. The term MoSCoW is derived from the first letter of each of four prioritization categories Must have, Should have, Could have, Would like but wont get.
- COCOMO: Cost Constructive Model is a procedural software cost estimation model which uses a basic regression formula with parameters that are derived from historic project data and current as well as future project characteristics. Basic COCOMO computes software development effort (and cost) as a function of program size (in KLOC).

2) Features:

- Motivational Statement: Organizations constantly need alignment of business goals with development strategies and justifying of cost and resources for software and system development. But we do not have effective methods for linking business goals and software related effort. GQM+ tries to bridge the gap between business strategies and their project level implementation.
- Related Work: The paper talks about related measurement programs like GQM, BSC, and PSM. GQM approach provides a method for an organization or a project to define goals, refine those goals into specifications of the data to be collected, and then analyze and interpret the resulting data in the light of the original goals. BSC (Balanced Scorecard) approach links strategic objectives and measures through a scorecard which consists of four perspectives: financial, customer, internal business processes and learning and growth. PSM (Practical Software Measurement) approach offers detailed guidance on software measurement, including providing a catalog of specific measures and information on how an organization can apply those measures in projects.
- Informative Visualizations: The paper has a list of figures and tables explaining GQM+ strategies and their ongoing applications. The following figure depicts the eight conceptual components, that form basis for constructing a consistent model, and how these components interrelate. The eight components are business goals, context factors, assumptions, strategies, level i goals, interpretation models, goal+ Strategies elements and GQM graph.

- Tutorial Materials: The paper takes on a sample application and teaches how to apply the GQM+ Strategy. The authors start with a business goal of increasing profit from software service usage. They build assumptions, questions, metrics and interpretation models around it. They then derive lower level goals and perform the same task. Finally relate how the lower tasks will help determine the success of the higher level goals.

3) Improvements:

- Absence of backup plan: The users are supposed to make assumptions and select strategies that work best. But the authors did not mention what to do if at some point during the lifecycle, the assumption backfires or goals are not met completely.
- Effectiveness of GQM+ approach: The paper offers a list of industries that have adopted the GQM+ paradigm but it might make more impact on the readers if the authors had provided supporting data of the effectiveness of the method.
- User lacking knowledge of linking goals and strategies: The paper also fails to point out the importance of determining goals and strategies. Mentioning a list of heuristics or defining who should be responsible for coming up with the correct and effective goals and strategies would help new engineers to pick up this method. The model is more likely to fail, when a person taking charge has a weak understanding of the links between the goals and strategies.

C. An Empirical Investigation into the Role of API-Level Refactorings during Software Evolution

Refactoring is the process of changing a programs design structure without modifying its external functional behavior in order to improve program readability, maintainability and extensibility.

Relevant studies show different results between refactoring and following bug fixes.

The goal of this paper is to systematically investigate the role of refactorings during the software evolution by examining the relationships between refactorings, bug fixes, the time to resolve bugs and release cycles using fine-grained version histories.

1) Keywords:

- API level Refactoring: API-Level refactoring are rename refactorings at the level of packages/classes/methods, add/delete parameter refactorings, move refactoring at the level of packages/classes/methods and changes to the return type of a method.
- Floss Refactoring: There are two different occasions when programmers refactor. The first kind occurs interweaved with normal program development, arising whenever and wherever design problems arise. For example, if a programmer introduces (or is about to introduce) duplication when adding a feature, then the programmer removes that duplication. This kind of refactoring, done frequently to maintain healthy software, is called floss refactoring.
- Change Distilling: It is a tree differencing algorithm for fine-grained source code change extraction. It was published in IEEE transactions to Software Engineering.
- Hunks: List of consecutive source code lines that were added or deleted.

2) Features:

- Motivational Statement: In theory, refactoring improves software quality and programmers productivity. However, in practice there are trends that show increase in bug reports after refactoring. This motivates the authors to dig deeper and systematically investigate the role of refactoring. The idea is to study and establish a relationship between refactorings, bug fixes, the time to resolve bugs and release cycles.
- Related Work: Weissgerber and Diehl found that a high ratio of refactorings is sometimes followed by an increasing ratio of bug reports. In contradiction, Ratzinger et al. found that the number of defects decreases if the number of refactorings increases in the preceding time period. Dig et al. found that 80% of the changes that break client applications are API-level refactorings. Another study showed that large commits, which tend to include refactorings, have a higher chance of including bugs. Moreover, studies have also shown that there are many bugs in refactoring tools and these tools do a poor job of communicating errors. All these relate to the outcome of the empirical investigation of the authors.
- Study Instruments and Commentary: The authors applied M. Kim et al.s refactorings reconstruction technique to find revisions that underwent refactoring and S. Kim et al.s bug history extraction technique to identify bug fix revisions. The refactorings are used for this study are from change logs of Eclipse JDT, jEdit and Columba.
- Results: Results of relationship between refactorings and bug reports include in-depth knowledge regarding the following questions
  - Are there more bug fixes after API-level refactorings? Yes, there is a short-term increase in the number of bug fixes after refactorings.
  - Do API-level refactorings improve developer productivity? Yes, when it comes to fixing bugs introduced near the time of refactorings, the average fix time tends to decrease after refactorings.
  - Do API-level refactoring facilitate bug fixes? Yes, Fixes and refactoring often appear in the same revision. Furthermore, it is more likely for a refactoring revision to be followed by related bug fixes than for a non-refactoring revision.
  - Are there relatively fewer API-level refactorings before major releases? No, there are more refactorings and bug fixes prior to major version release.

3) Improvements:

- Generalization of results: The authors have closely

studied API-level refactorings but in their results they have generalized these to normal refactorings. To a reader perusing the results, it might mislead to believe that general refactorings have the same trend as API-level refactorings.
- Results and type of refactorings: The results generated talk about API-level refactorings and its trends with bug-fixes and release cycles. It would have been more interesting to know what type of refactorings, manual or automatic, conform to these results.
- Excluding Software Life Cycle Activity: The authors did not take into account the fact that their subject might practice the activity of refactoring the code before a major release. This fact might skew the last result.

III. Baseline Study

In the paper "Information Needs for Software Development Analysis", authors Buse and Zimmermann have presented the data and analysis needs of professional software engineers. They conducted a survey among 110 Microsoft developers and managers, in which they asked about their decision making process, their needs for artifacts and indicators, and scenarios in which they would use analytics. source [Buse, R.P.L. and Zimmermann, T. Information needs for software development analytics. In ICSE '12: Proceedings of 34th International Conference on Software Engineering (2012), 987-996]

A. Methodology

The authors advertised a survey consisting of 28 questions to random, equal-sized pools of engineers and lead engineers(managers). It consisted questions from 4 categories [1]

- **Analytical**: What factors influence your decision making process?
- **Importance and Difficulty**: What questions are important or difficult to answer?
- **Artifact**: What artifacts are important to measure?
- **Indicator**: What indicators do you currently use? What would you like to use?
- **Decision Scenarios**: What decisions could analytics help with?

A total of 110 (57 managers and 53 developers) responded. The authors also recorded both the importance and difficulty of answering the questions in each domain. Importance-related questions were rated on a 4-point scale of Not-important; Somewhat-important; Important; Very-Important. Similarly difficulty-related questions were rated on a 4-point scale of Not-Difficult; Somewhat-Difficult; Difficult; Very-Difficult.

[1]The questions in the paper are edited for brevity. The original questions are more specific and listed in a technical report [R. Buse and T. Zimmermann, Information needs for software development analytics, Microsoft Research, Tech. Rep.MSR-TR-2011-8, 2010, Appendix 2].

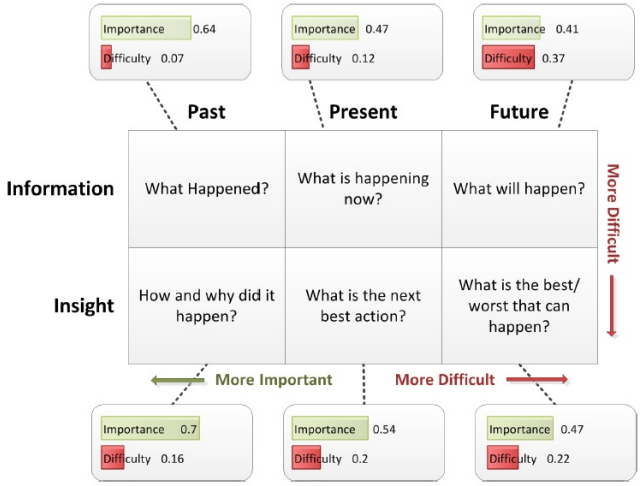

Fig. 1. Analytics Questions. The authors distinguish between questions of information which can be directly measured, from questions of insight which arise from a careful analytic analysis and provide managers with a basis for action.

B. Survey Results

In case of analytics, 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.

While rating importance and difficulty, participants responded, Figure 1, 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;

Analyzing the individual features of a project was deemed most important artifact by both developers and managers as many important decision 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.

In addition to artifacts, participants rated on the use of indicators. Figure 2, shows, 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.

For the decision scenarios, the survey participants described a total of 102 total scenarios. Some of the common themes emerged were: Targeting Testing, Targeting Refactoring, Release Planning, Understanding Customers, Judging Stability, Targeting Training, and Targeting Inspection.

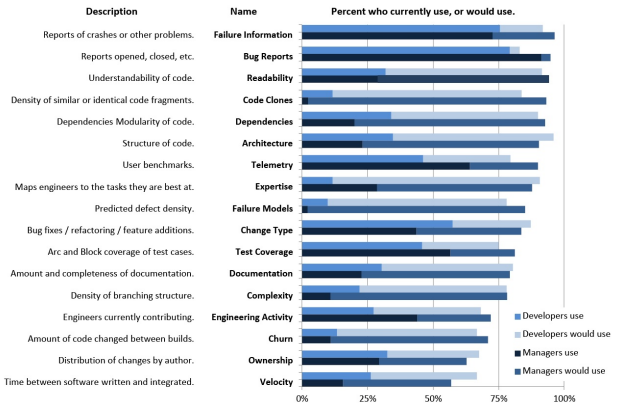

Fig. 2. Percent of managers and developers who reported that they either use or would use (if available) each of the given indicators in making decisions relevant to their engineering process.

C. Software Analytics Guidelines

The survey responses lead the authors 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
- Engineers want to drill down into data based on time, organizational structure, and system architecture

Additionally, the authors identified a set of analysis types that fit well with information needs. In Figure 3 they 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).

IV. Current Research

A. *A. Begel, T. Zimmermann, Analyze This! 145 Questions for Data Scientist in Software Engineering, 2014*

The paper is about two surveys related to data science applied to software engineering. The first survey solicited questions that software engineers would like to ask data scientists to investigate about software, software processes and practices. This yielded 145 questions grouped in 12 categories. The second survey asked a different pool of software engineers to rate the 145 questions. Respondents favored questions that focus on how customers typically use their applications. The idea behind categorizing and cataloging 145 questions is to help researchers, practitioners, and educators to more easily focus on their efforts on topics that are important to the software industry.

1) Keywords:

- Software Analytics: It is a subfield of analytics with the focus on software data.
- Affinity Chart: The affinity diagram is a business tool used to organize ideas and data. The affinity diagram organizes ideas with following steps:
  - Record each idea on cards or notes.
  - Look for ideas that seem to be related.
  - Sort cards into groups until all cards have been used.
- Churn: Your churn rate is the amount of customers or subscribers who cut ties with your service or company during a given time period. These customers have churned. Business people say an acceptable churn rate is in the 5 7% range annually, depending upon whether it measures customers or revenue.
- Telemetry: Software Project Telemetry is a project management technique that uses software sensors to collect metrics automatically and unobtrusively. It then employs a domain-specific language to represent telemetry trends in software product and process metrics. Project management and process improvement decisions are made by detecting changes in telemetry trends and comparing trends between different periods of the same project.

2) Features:

- Motivational Statement: The data science stream is growing. All businesses want to use analytics to better reach their customers. As a result, there are keynote speeches and panel talks about how to make the software engineering community more data-driven. With this survey the authors have given the software developers a chance to express their doubts regarding to data-science.
- Related Work: With the big data boom, several research groups pushed for more use of data for decision making, and shared their experiences collaborating with industry on analytics projects. Zhang et al. emphasized the trinity oh software analytics in the form of three research topics (development process, system, users) as well as three technology pillars (information visualizations, analysis algorithms, large-scale computing). Buse and Zimmermann argued for a dedicated analyst role in software projects and presented an empirical survey on guidelines for analytics in software development.
- Patterns: The authors received 728 items in 203 responses in the first survey. In order to categorize they used the open card sort technique. Card sorting is a technique that is widely used to create mental models and derive taxonomies from data. In this case, card sorting also helps us to deduce a higher level of abstraction and identify common themes.
  Card sorting has three phases: in the preparation phase, cards for each question written by the respondents are created; in the execution phase cards are sorted into meaningful groups with a descriptive title; finally, in the analysis phase, abstract hierarchies are formed in order to deduce general categories and themes.

3) Improvements:

	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. 3. 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.

- The survey could have been both internal and external: The survey was internal to Microsoft. The authors could have reached out to a bigger audience and weighted the opinions of other software engineers around the world.
- Data scientists view on the survey results: Since the idea to help researchers, practitioners and educators improve their focus and efforts on topics that are important, the authors could have added a step where the data scientists explain how each of the top 24 questions could be analyzed.
- Taking input from data scientists: The motivation is to make software processes more data driven. Then why not include the opinion of data scientists (since they are also part of the organization and possess knowledge about the processes). The authors could have collaborated with data scientists to rate the questions. Moreover, they could also have created a reverse survey where data scientists ask the questions regarding current software practices and then told the data scientists to analyze how these practices could be transformed into data driven practices.

4) Connection to Paper 1: The authors in this paper tried to understand the questions that software engineers would like to ask to data scientists about software. To me, this paper seems to be inspired from the paper, where the authors have surveyed 110 software developers to present a spectrum of information needs and the corresponding insights gained from them. Moreover, the authors categorized those information needs into 9 categories. Interestingly, Thomas Zimmermann, co-author of that paper.

5) Conclusion: Researchers can use such survey results to collaborate with industry and influence their software development processes, practices and tools. Professionals can get idea about which factors to analyze. Educators can get guidance on what analytical techniques to teach.

B. X. Ye, R Bunescu, Learning to Rank Relevant Files for Bug Reports using Domain Knowledge, 2014

The paper is about two surveys related to data science applied to software engineering. The first survey solicited questions that software engineers would like to ask data scientists to investigate about software, software processes and practices. This yielded 145 questions grouped in 12 categories. The second survey asked a different pool of software engineers to rate the 145 questions. Respondents favored questions that focus on how customers typically use their applications. The idea behind categorizing and cataloging 145 questions is to help researchers, practitioners, and educators to more easily focus on their efforts on topics that are important to the software industry.

1) Keywords:

- Software Analytics: It is a subfield of analytics with the focus on software data.
- Affinity Chart: The affinity diagram is a business tool used to organize ideas and data. The affinity diagram organizes ideas with following steps:
  - Record each idea on cards or notes.
  - Look for ideas that seem to be related.
  - Sort cards into groups until all cards have been used.
- Churn: Your churn rate is the amount of customers or subscribers who cut ties with your service or company during a given time period. These customers have churned. Business people say an acceptable churn rate is in the 5 7% range annually, depending upon whether it measures customers or revenue.

- Telemetry: Software Project Telemetry is a project management technique that uses software sensors to collect metrics automatically and unobtrusively. It then employs a domain-specific language to represent telemetry trends in software product and process metrics. Project management and process improvement decisions are made by detecting changes in telemetry trends and comparing trends between different periods of the same project.

2) Features:

- Motivational Statement: The data science stream is growing. All businesses want to use analytics to better reach their customers. As a result, there are keynote speeches and panel talks about how to make the software engineering community more data-driven. With this survey the authors have given the software developers a chance to express their doubts regarding to data-science.
- Related Work: With the big data boom, several research groups pushed for more use of data for decision making, and shared their experiences collaborating with industry on analytics projects. Zhang et al. emphasized the trinity oh software analytics in the form of three research topics (development process, system, users) as well as three technology pillars (information visualizations, analysis algorithms, large-scale computing). Buse and Zimmermann argued for a dedicated analyst role in software projects and presented an empirical survey on guidelines for analytics in software development.
- Patterns: The authors received 728 items in 203 responses in the first survey. In order to categorize they used the open card sort technique. Card sorting is a technique that is widely used to create mental models and derive taxonomies from data. In this case, card sorting also helps us to deduce a higher level of abstraction and identify common themes.
  Card sorting has three phases: in the preparation phase, cards for each question written by the respondents are created; in the execution phase cards are sorted into meaningful groups with a descriptive title; finally, in the analysis phase, abstract hierarchies are formed in order to deduce general categories and themes.

3) Improvements:

- The survey could have been both internal and external: The survey was internal to Microsoft. The authors could have reached out to a bigger audience and weighted the opinions of other software engineers around the world.
- Data scientists view on the survey results: Since the idea to help researchers, practitioners and educators improve their focus and efforts on topics that are important, the authors could have added a step where the data scientists explain how each of the top 24 questions could be analyzed.
- Taking input from data scientists: The motivation is to make software processes more data driven. Then why not include the opinion of data scientists (since they are also part of the organization and possess knowledge about the processes). The authors could have collaborated with data scientists to rate the questions. Moreover, they could also have created a reverse survey where data scientists ask the questions regarding current software practices and then told the data scientists to analyze how these practices could be transformed into data driven practices.

4) Connection to Paper 1: The authors in this paper tried to understand the questions that software engineers would like to ask to data scientists about software. To me, this paper seems to be inspired from the paper, where the authors have surveyed 110 software developers to present a spectrum of information needs and the corresponding insights gained from them. Moreover, the authors categorized those information needs into 9 categories. Interestingly, Thomas Zimmermann, co-author of that paper.

5) Conclusion: Researchers can use such survey results to collaborate with industry and influence their software development processes, practices and tools. Professionals can get idea about which factors to analyze. Educators can get guidance on what analytical techniques to teach.

V. Threats to Validity

VI. Our Commentary

VII. Future Scope

VIII. Conclusion

IX. References

X. PROCEDURE FOR PAPER SUBMISSION

A. Selecting a Template (Heading 2)

First, confirm that you have the correct template for your paper size. This template has been tailored for output on the US-letter paper size. Please do not use it for A4 paper since the margin requirements for A4 papers may be different from Letter paper size.

B. Maintaining the Integrity of the Specifications

The template is used to format your paper and style the text. All margins, column widths, line spaces, and text fonts are prescribed; please do not alter them. You may note peculiarities. For example, the head margin in this template measures proportionately more than is customary. This measurement and others are deliberate, using specifications that anticipate your paper as one part of the entire proceedings, and not as an independent document. Please do not revise any of the current designations

XI. MATH

Before you begin to format your paper, first write and save the content as a separate text file. Keep your text and graphic files separate until after the text has been formatted and styled. Do not use hard tabs, and limit use of hard returns to only one return at the end of a paragraph. Do not add any kind of pagination anywhere in the paper. Do not number text heads-the template will do that for you.

Finally, complete content and organizational editing before formatting. Please take note of the following items when proofreading spelling and grammar:

A. Abbreviations and Acronyms

Define abbreviations and acronyms the first time they are used in the text, even after they have been defined in the abstract. Abbreviations such as IEEE, SI, MKS, CGS, sc, dc, and rms do not have to be defined. Do not use abbreviations in the title or heads unless they are unavoidable.

B. Units

- Use either SI (MKS) or CGS as primary units. (SI units are encouraged.) English units may be used as secondary units (in parentheses). An exception would be the use of English units as identifiers in trade, such as 3.5-inch disk drive.
- Avoid combining SI and CGS units, such as current in amperes and magnetic field in oersteds. This often leads to confusion because equations do not balance dimensionally. If you must use mixed units, clearly state the units for each quantity that you use in an equation.
- Do not mix complete spellings and abbreviations of units: Wb/m2 or webers per square meter, not webers/m2. Spell out units when they appear in text: . . . a few henries, not . . . a few H.
- Use a zero before decimal points: 0.25, not .25. Use cm3, not cc. (bullet list)

C. Equations

The equations are an exception to the prescribed specifications of this template. You will need to determine whether or not your equation should be typed using either the Times New Roman or the Symbol font (please no other font). To create multileveled equations, it may be necessary to treat the equation as a graphic and insert it into the text after your paper is styled. Number equations consecutively. Equation numbers, within parentheses, are to position flush right, as in (1), using a right tab stop. To make your equations more compact, you may use the solidus (/), the exp function, or appropriate exponents. Italicize Roman symbols for quantities and variables, but not Greek symbols. Use a long dash rather than a hyphen for a minus sign. Punctuate equations with commas or periods when they are part of a sentence, as in

$$\alpha + \beta = \chi \tag{1}$$

Note that the equation is centered using a center tab stop. Be sure that the symbols in your equation have been defined before or immediately following the equation. Use (1), not Eq. (1) or equation (1), except at the beginning of a sentence: Equation (1) is . . .

D. Some Common Mistakes

- The word data is plural, not singular.
- The subscript for the permeability of vacuum ?0, and other common scientific constants, is zero with subscript formatting, not a lowercase letter o.
- In American English, commas, semi-/colons, periods, question and exclamation marks are located within quotation marks only when a complete thought or name is cited, such as a title or full quotation. When quotation marks are used, instead of a bold or italic typeface, to highlight a word or phrase, punctuation should appear outside of the quotation marks. A parenthetical phrase or statement at the end of a sentence is punctuated outside of the closing parenthesis (like this). (A parenthetical sentence is punctuated within the parentheses.)
- A graph within a graph is an inset, not an insert. The word alternatively is preferred to the word alternately (unless you really mean something that alternates).
- Do not use the word essentially to mean approximately or effectively.
- In your paper title, if the words that uses can accurately replace the word using, capitalize the u; if not, keep using lower-cased.
- Be aware of the different meanings of the homophones affect and effect, complement and compliment, discreet and discrete, principal and principle.
- Do not confuse imply and infer.
- The prefix non is not a word; it should be joined to the word it modifies, usually without a hyphen.
- There is no period after the et in the Latin abbreviation et al..
- The abbreviation i.e. means that is, and the abbreviation e.g. means for example.

XII. USING THE TEMPLATE

Use this sample document as your LaTeX source file to create your document. Save this file as **root.tex**. You have to make sure to use the cls file that came with this distribution. If you use a different style file, you cannot expect to get required margins. Note also that when you are creating your out PDF file, the source file is only part of the equation. *Your TEX → PDF filter determines the output file size. Even if you make all the specifications to output a letter file in the source - if you filter is set to produce A4, you will only get A4 output.*

It is impossible to account for all possible situation, one would encounter using TEX. If you are using multiple TEX files you must make sure that the "MAIN" source file is called root.tex - this is particularly important if your conference is using PaperPlaza's built in TEX to PDF conversion tool.

A. Headings, etc

Text heads organize the topics on a relational, hierarchical basis. For example, the paper title is the primary text head because all subsequent material relates and elaborates on this one topic. If there are two or more sub-topics, the next

level head (uppercase Roman numerals) should be used and, conversely, if there are not at least two sub-topics, then no subheads should be introduced. Styles named Heading 1, Heading 2, Heading 3, and Heading 4 are prescribed.

B. Figures and Tables

Positioning Figures and Tables: Place figures and tables at the top and bottom of columns. Avoid placing them in the middle of columns. Large figures and tables may span across both columns. Figure captions should be below the figures; table heads should appear above the tables. Insert figures and tables after they are cited in the text. Use the abbreviation Fig. 1, even at the beginning of a sentence.

TABLE I
AN EXAMPLE OF A TABLE

One	Two
Three	Four

We suggest that you use a text box to insert a graphic (which is ideally a 300 dpi TIFF or EPS file, with all fonts embedded) because, in an document, this method is somewhat more stable than directly inserting a picture.

Fig. 4. Inductance of oscillation winding on amorphous magnetic core versus DC bias magnetic field

Figure Labels: Use 8 point Times New Roman for Figure labels. Use words rather than symbols or abbreviations when writing Figure axis labels to avoid confusing the reader. As an example, write the quantity Magnetization, or Magnetization, M, not just M. If including units in the label, present them within parentheses. Do not label axes only with units. In the example, write Magnetization (A/m) or Magnetization A[m(1)], not just A/m. Do not label axes with a ratio of quantities and units. For example, write Temperature (K), not Temperature/K.

XIII. CONCLUSIONS

A conclusion section is not required. Although a conclusion may review the main points of the paper, do not replicate the abstract as the conclusion. A conclusion might elaborate on the importance of the work or suggest applications and extensions.

APPENDIX

Appendixes should appear before the acknowledgment.

ACKNOWLEDGMENT

The preferred spelling of the word acknowledgment in America is without an e after the g. Avoid the stilted expression, One of us (R. B. G.) thanks . . . Instead, try R. B. G. thanks. Put sponsor acknowledgments in the unnumbered footnote on the first page.

References are important to the reader; therefore, each citation must be complete and correct. If at all possible, references should be commonly available publications.

REFERENCES

[1] G. O. Young, Synthetic structure of industrial plastics (Book style with paper title and editor), in Plastics, 2nd ed. vol. 3, J. Peters, Ed. New York: McGraw-Hill, 1964, pp. 1564.
[2] W.-K. Chen, Linear Networks and Systems (Book style). Belmont, CA: Wadsworth, 1993, pp. 123135.
[3] H. Poor, An Introduction to Signal Detection and Estimation. New York: Springer-Verlag, 1985, ch. 4.
[4] B. Smith, An approach to graphs of linear forms (Unpublished work style), unpublished.
[5] E. H. Miller, A note on reflector arrays (Periodical styleAccepted for publication), IEEE Trans. Antennas Propagat., to be publised.
[6] J. Wang, Fundamentals of erbium-doped fiber amplifiers arrays (Periodical styleSubmitted for publication), IEEE J. Quantum Electron., submitted for publication.
[7] C. J. Kaufman, Rocky Mountain Research Lab., Boulder, CO, private communication, May 1995.
[8] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, Electron spectroscopy studies on magneto-optical media and plastic substrate interfaces(Translation Journals style), IEEE Transl. J. Magn.Jpn., vol. 2, Aug. 1987, pp. 740741 [Dig. 9th Annu. Conf. Magnetics Japan, 1982, p. 301].
[9] M. Young, The Techincal Writers Handbook. Mill Valley, CA: University Science, 1989.
[10] J. U. Duncombe, Infrared navigationPart I: An assessment of feasibility (Periodical style), IEEE Trans. Electron Devices, vol. ED-11, pp. 3439, Jan. 1959.
[11] S. Chen, B. Mulgrew, and P. M. Grant, A clustering technique for digital communications channel equalization using radial basis function networks, IEEE Trans. Neural Networks, vol. 4, pp. 570578, July 1993.
[12] R. W. Lucky, Automatic equalization for digital communication, Bell Syst. Tech. J., vol. 44, no. 4, pp. 547588, Apr. 1965.
[13] S. P. Bingulac, On the compatibility of adaptive controllers (Published Conference Proceedings style), in Proc. 4th Annu. Allerton Conf. Circuits and Systems Theory, New York, 1994, pp. 816.
[14] G. R. Faulhaber, Design of service systems with priority reservation, in Conf. Rec. 1995 IEEE Int. Conf. Communications, pp. 38.
[15] W. D. Doyle, Magnetization reversal in films with biaxial anisotropy, in 1987 Proc. INTERMAG Conf., pp. 2.2-12.2-6.
[16] G. W. Juette and L. E. Zeffanella, Radio noise currents n short sections on bundle conductors (Presented Conference Paper style), presented at the IEEE Summer power Meeting, Dallas, TX, June 2227, 1990, Paper 90 SM 690-0 PWRS.
[17] J. G. Kreifeldt, An analysis of surface-detected EMG as an amplitude-modulated noise, presented at the 1989 Int. Conf. Medicine and Biological Engineering, Chicago, IL.
[18] J. Williams, Narrow-band analyzer (Thesis or Dissertation style), Ph.D. dissertation, Dept. Elect. Eng., Harvard Univ., Cambridge, MA, 1993.
[19] N. Kawasaki, Parametric study of thermal and chemical nonequilibrium nozzle flow, M.S. thesis, Dept. Electron. Eng., Osaka Univ., Osaka, Japan, 1993.
[20] J. P. Wilkinson, Nonlinear resonant circuit devices (Patent style), U.S. Patent 3 624 12, July 16, 1990.