In Pursuit of Optimal Workflow

Within The

Apache Software Foundation

By

Ryan Panos

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved April 2017 by the
Graduate Supervisory Committee:

James Collofello, Co-Chair
John Fowler, Co-Chair
Teresa Wu
Rong Pan

ARIZONA STATE UNIVERSITY

May 2017

ABSTRACT

The following is a case study composed of three workflow investigations at the open source software development (OSSD) based Apache Software Foundation (Apache). I start with an examination of the workload inequality within the Apache, particularly with regard to requirements writing. I established that the stronger a participant's experience indicators are, the more likely they are to propose a requirement that is not a defect and the more likely the requirement is eventually implemented. Requirements at Apache are divided into work tickets (tickets). In our second investigation, I reported many insights into the distribution patterns of these tickets. The participants that create the tickets often had the best track records for determining who should participate in that ticket. Tickets that were at one point volunteered for (self-assigned) had a lower incident of neglect but in some cases were also associated with severe delay. When a participant claims a ticket but postpones the work involved, these tickets exist without a solution for five to ten times as long, depending on the circumstances. I make recommendations that may reduce the incidence of tickets that are claimed but not implemented in a timely manner. After giving an in-depth explanation of how I obtained this data set through web crawlers, I describe the pattern mining platform I developed to make my data mining efforts highly scalable and repeatable. Lastly, I used process mining techniques to show that workflow patterns vary greatly within teams at Apache. I investigated a variety of process choices and how they might be influencing the outcomes of OSSD projects. I report a moderately negative association between how often a team updates the specifics of a requirement and how often requirements are completed. I also verified that the prevalence of volunteerism indicators is positively associated with work completion but what was surprising is that this correlation is stronger if I exclude the very large projects. I suggest the largest projects at Apache may benefit from some level of traditional delegation in addition to the phenomenon of volunteerism that OSSD is normally associated with.

DEDICATION

This dissertation is dedicated to the most supportive wife I could possibly ask for. House hunting in 117 degrees, living with a 39-year-old pulling all-nighters and all the material sacrifices we have made did not dissuade Becky Deeter Panos from supporting this ludicrous goal.

This is also dedicated to the master of silly, Miss Samantha Maxine Panos. We will be visiting the animals[1] at the zoo more often now.

Of the countless educators that helped me get this far, I am exceptionally grateful to:

Rich Acton for encouraging my self-study of the Apple IIe/c over 25 years ago

Dr. Lee Schruben for nearly two decades of advice and support, particularly in the years that I undersold my own potential

Dr. Tali Freed for giving me exceptional exposure to applied research and for being the first to propose this preposterous notion that I obtain a terminal degree

Dr. Robert Crockett for providing so much advice and skillfully selling the PhD degree during our graduate seminar a dozen years ago

[1] Many named Phil

ACKNOWLEDGMENTS

The greatest mental and emotional challenge of my life was far from a solo effort.

My four committee members, Dr. James Collofello, Dr. John Fowler, Dr. Rong Pan and Dr. Teresa Wu, are some of the most productive faculty members at Arizona State University yet they have been generous, supportive, and patient. They frequently gave me guidance so that I could find the most direct path to the next goal. All four are priceless members of the academic community and I am grateful that they continue to contribute to Arizona State.

I am also incredibly grateful to Dr. Kevin Thompson of CPrime for generously sharing so many thoughts on software engineering workflow, showing great patience with my inexperience with research and for displaying an inspiring ability to think outside the box. Roy Chartier effectively acted as my product manager before I knew what the product was going to be and may have prevented a product failure three years before the finish line. Dr. Philippe Fournier-Viger of the Harbin of Institute of Technology Shenzhen, the creator of the pattern-mining tool SPMF that my platform is based on, has been remarkably generous in his knowledge and expertise through correspondence. Dr. Lei “Erik” Liu not only lent me quite a bit of his exceptional skills in statistics and data mining but he was the first to inspire my passion for data science. That I should utilize sequential pattern mining was completely his idea.

Some coworkers had a greater impact than they might have imagined: from Protagonist, I am very grateful to Dr. Ali Shahed hagh ghadam and Dr. David Sun for some excellent ideas but especially for giving me a healthy fear of causal arguments; from WalmartLabs: Dr. Liliya (Besaleva) McLean, Dr. Kannan Achan and especially almost-Dr. Jason Cho for a lot of brilliant ideas and encouragement; Of the late-great OnLive: Ruth Helfinstein, Noah Gibbs, and Josh Olson for the greatest unofficial support group in Silicon Valley and pushing my tech blogging; Robert Jacobs for advice on relationships in and out of the workplace that I will never forget but also for being the first to suggest my topic of Systems Engineering.

Some friends and family had a direct impact in my PhD program but probably did not know it. Counsel from almost-Dr. Harleigh Marsh made so many stressors more manageable because he could relate in ways no one else could. I know that in the next few decades

countless students and researchers will benefit from his remarkable empathy and astonishing ability rise above ego battles. Zach Brown, Nick “Holmes” Travaglini, Jeff Engler and countless other younger Gammas continually reminded me I began this journey because I wanted to have an impact. Ross Panos passed on the idea that a dissertation can also be the birth of a product. I am also thankful to Dr. Amit Shinde for discussing research while partaking in free golf in 105 degree heat but especially for not letting me neglect the terror of our qualifying exam. I thank Dr. Shilpa Shinde for the same and for permitting said golf.

Other supportive and inspirational family and friends helped me get through some of the more discouraging times. I am grateful to James Lloyd Johnson for off-roading, schnitz and rarely doubting my unconventional choices; unless they were really really stupid then I have trusted him to make me hear that. I am really fortunate Leo Cheng is able to do the same. Almost-Dr. Mandy Panos is an inspiration for never letting the world slow her down. Dr. Sandy Pavelka and Dr. Hank Childs proved that responsibility does not prevent one from completing a dissertation. I might have given up if it weren’t for the generosity of Caesar Garcia. Dr. Brent Nelson and Dr. Mark Yeary gave me their candid insights into this bizarre academic universe. I am also inspired by the late Dr. Michael Summers of Pepperdine University for the impact he had on countless young lives and I am grateful to him for painting a positive view of the academic life.

I also thank Dr. Reza Pouraghabagher for being the most enthusiastic Engineering Professor I’ve ever known and Lura Dolas for all that she helped me learn about myself. ASU will not permit me the space to elaborate but I also credit the following supportive “peoples”: my Arizona big sister Heather Cate, Louise Geist, Rick Sanders, James Niehaus, 4 Peironnets, Simon G. Schreiber, Dr. William Josephson, Drs. Buckley, Dr. Cervantes, Dr. Brian Noble, Dr. David Porter & Dr. David Kim of Oregon State, Bryan Cardoza, Susan Pico, Both Trocks, Pab, Habs, ROG, John C Cardoza, Karen Lund-Dennison, Dr. George Runger, Clifton Volstorff, and the selfless & infinitely accepting Brother Justin.

Lastly, I thank my mother and father for, like both of their parents before them, emphasizing the importance of education from the beginning.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

1 Dissertation Introduction

Software development projects are infamous for missed deadlines and budget overruns. These are often caused by unexpected issues and obstacles we prefer to think of as anti-patterns. The software development process is expected to follow a short list of accepted processes patterns but often does not. Any attempt to reduce these anti-patterns will improve workflow and reduce delays or cost over runs. This is why many research topics in Software Engineering can be boiled down to this attempt at understanding hindrances to optimal workflow. We set out to contribute to more than one of these areas.

To do so, there were many established approaches we could have taken such as surveys, simulation modeling and scientific experiments. We opted to avoid surveys as perceived processes are often not the real processes that are taking place(W. Van Der Aalst, Weijters, & Maruster, 2004). Scientific experiments might provide stronger data but they are usually very expensive to carry out (Zelkowitz & Wallace, 1997) (Ur, Yom-tov, & Wernick, 2007) in the software development environment primarily due to the value placed on an engineer's time. Simulation models are often the popular alternative but many are not sufficiently based on hard data.

This leaves case studies. However, despite their popularity in Software Engineering research (Bai, Zhang, & Huang, 2011), case study findings have been criticized as not readily generalizable(Tsang, 2014) at least in part because the "sample size in any case study research project is never going to be large enough to qualify for the use of statistical inference (Easton, 2010)." However, advocates argue that the most powerful case studies utilize larger datasets and emphasizing those studies "increases the generalizability and validity of the findings (J. L. Jensen & Rodgers, 2001)." (Lee, 1989) proposes that generalizability can be increased by making confirming observations "when the theory is tested on other empirical circumstances." When a topic has very little literature surrounding it, "case research strategy is well-suited to capturing the knowledge of practitioners and developing theories from it" according to (Benbasat, Goldstein, & Mead, 1987) So quite simply, if we are not in a position to conduct scientific

experiments, conduction case studies using a very significant number of entities is a well-supported methodology in software engineering research, despite some criticism in other fields. In this dissertation, we chose this route.

So we set out to acquire a comprehensive dataset that would provide the potential to offer insights into numerous sources of software engineering delays and hurdles. As our attempts to acquire empirical event log data from corporate settings were politely denied and a few times mocked, we sought open source data. We discovered that the requirements repository of the one of the most significant open source software development (OSSD) organizations, the Apache Software Foundation (Apache), is available to the public in read-only form. We argue that their ~500,000 tickets and nearly 450 teams[2] is an usually large sample size for software engineering research and therefore the ideal dataset to acquire in a single effort. This acquisition effort is detailed in chapter 3.

We acknowledge that much of our findings will carry much greater weight with the less structured, volunteer dependent and sometimes unconventional open source community instead of the software engineering community as a whole. However, some findings might also apply more broadly. It was established at Microsoft that tickets written by the more experienced team members were met with more confidence (Guo, Zimmermann, Nagappan, & Murphy, 2010). In other words, experience level influences success in requirements writing. In chapter 1 we were able to verify this principle also applies in the many Apache teams as well. In contrast, our findings with regard to volunteerism are likely to be heavily influenced by the unorthodox nature of OSSD. However, much literature supports the idea that highly motivated developers make more productive employees. Could the self-driven motivators of OSSD work possibly parallel some of the studies on how best to drive software team members to excel?

We also argue that if our work primarily applies to OSSD, this is still a significant contribution as OSSD is vital to the industry as a whole for many reasons. The web may have been cast in the iron shackles designed by MSFT from the start had Apache's original project (the Apache Web

[2] As of our 2015 data acquisition

Server) not been made free to the public in 1995. In recent years, major software organizations deliberately turn over some of their greatest infrastructure projects to Apache so that, not only can the rest of the world utilize these invaluable tools but so that volunteers might help these projects evolve. Facebook donated Hive and Cassandra, Yahoo donated Hadoop and Adobe donated Flex, just to name a few. The world of big data is quite dependent on Apache's clustering framework projects Hadoop and Spark or Apache's NoSQL stores such as Cassandra or Couchbase. So any improvement in the processes that these OSSD teams choose will only improve the entire industry's path forward in countless ways.

This dissertation is laid out as such. Chapter one focuses on the beginning of a requirement's existence as we survey the many contribution levels of Apache participants. After describing data we have on the contributions from the ~80,000 participants at Apache, we address some research questions regarding how a participant's experience level influences the types of requirements they tend to submit in addition to the finding on the effectiveness of these requirements mentioned previously.

Chapter two focuses on how tickets are distributed throughout the team. We found that the participants that create the tickets tend to have a better track record for finding an owner that is going to complete the work than other participants. Generally, the tickets that were volunteered for (A.K.A. self-assigned) have been completed at a higher rate than those that were assigned to other participants. We also discovered a major pattern of delay. Tickets that are volunteered for but not completed almost directly after being volunteered for take five to ten times as long, depending on the ticket type and other factors, to complete than tickets whose voluntary owner completes the task in that same sitting. We provide extensive analysis and recommendations regarding this delay causing scenario.

In chapter three we not only detail how we acquired this data set but also describe a pattern mining platform we built to facilitate repeatable and scalable pattern mining experiments utilizing the pattern mining algorithms provided by the open source tool SPMF. With this platform, we were able to study each team and their processes individually, provided they had enough tickets to justify the association rules we sought. Our objective was to establish various measures of

effectiveness (especially with regard to completion) and various indicators for process choices for each team. We were especially interested in how processes of the teams vary in requirements updates, assignment changes (especially abandonment) and exposure to volunteerism. When comparing these teams by their completion measures found a negative association with requirements changes and an even stronger negative association with the rate of participants removing themselves as ticket owners (abandonment). Stronger volunteerism indicators are associated with greater completion rates but not nearly as much with the very largest projects to the point that we suggest the effectiveness of occasional delegation may be a factor in the biggest projects succeeding. We then wrapped up this last investigation of our case study by highlighting some process indicators measures that are common in the teams that score highest on our completion measures.

2 WORKLOAD INEQUALITY AND REQUIREMENTS SUBMISSION IN THE OPEN SOURCE COMMUNITY

*Ryan Panos * James Collofello * Rong Pan * Lei Liu*

Participants that form the Open Source Software Development (OSSD) community contribute at tremendously different rates with a long tail of members that make only one or two changes to a project. We leveraged a large requirements repository dataset from the Apache Software Foundation (Apache) to build off previous research that not only describes this workload inequality in some detail but also gives some insight into how differently these groups contribute to the success of projects. We first perform exploratory analysis and developed our own set of Software Development Analytics to gain a better view of how the tasks of OSS projects are divided. We then investigated a specific contribution area: requirements submission. We found that a very large portion of the Apache community participates by submitting only a few sets (and often just one set) of suggestions that are collected in a work ticket (ticket). Of the single ticket contributors, about 75% of the time these requirements pertain to a defect rather than improvements. Since roughly half of the tickets from all contributors pertain to product improvements rather than defects, we were able to show that a very small but extremely active subset of the population focuses on writing these improvement tickets. We also found the longer a person is involved or the more requirements they submit, the more likely someone will implement their suggestions.

Key Words: Software Development Analytics, Mining Software Repositories, Requirements Repositories, Open Source Software Contributors

Ryan Panos
School of Computing, Informatics, and Decision Systems Engineering
Arizona State University - Tempe, AZ
Email: rpaons@asu.edu

James Collofello
School of Computing, Informatics, and Decision Systems Engineering
Arizona State University Tempe, AZ
Email: james.collofello@asu.edu

Rong Pan
School of Computing, Informatics, and Decision Systems Engineering

Arizona State University - Tempe, AZ
Email: Rong.Pan@asu.edu

Lei Liu
Apple Inc - Cupertino, CA
Email: erikhpc2014@gmail.com

2.1 Introduction

Unlike commercial ventures where every employee is expected, at least officially, to contribute in equal measure, Open Source Software Development (OSSD) projects are primarily driven by two groups: a relatively small group of core developers and endless sea of casual contributors[3] (Pinto, Steinmacher, & Gerosa, 2016) (Herraiz, Gonzalez-Barahona, Robles, & German, 2007)(Baysal, Kononenko, Holmes, & Godfrey, 2016). This workload inequality has been the topic of numerous research efforts including some that investigated the factors that might lead an initial contributor to becoming a Long Time Contributor (LTC) (Minghui Zhou & Mockus, 2015)(Steinmacher, Wiese, Chaves, & Gerosa, 2013)(Steinmacher, Conte, Gerosa, & Redmiles, 2015). In (Vasilescu et al., 2015) they showed that tenure diversity improve a team's productivity and turnover rate. Meanwhile, (Pinto et al., 2016) and (Goeminne & Mens, 2011) both investigated how and why both groups are essential. In particular (Pinto et al., 2016) used data from the popular code repository tool GitHub to establish that casual contributors made up 49% of the population but that they contributed 1.73% of the changes to code (commits) in this study's samples set. They also established that a casual contributor is approximately 60% more likely to resolve a defect than to create a new feature. Meanwhile (Goeminne & Mens, 2011) used Pareto Analysis to accentuate how much of the total work is accomplished by the top 10-20% of contributor population. They tracked email and commit activities across three projects like other related studies, but they also tracked the requirements being written in by tabulating the number changes made to work tickets (tickets) by each participant. We observed that this measure of project progress is less common in literature, especially with regard to this topic of the diversity in contribution workloads for open source projects.

[3] Sometimes known as "drive by committers" (Steinmacher et al., 2015)

Methods from these studies varied in other ways. Some have used software repository data to support their conclusions (Minghui Zhou & Mockus, 2015)(Steinmacher et al., 2013)(M Zhou, Mockus, Ma, Zhang, & Mei, 2016)(Minghui Zhou & Mockus, 2012)(Vasilescu et al., 2015), but many relied heavily on surveys (Georgios & Bacchelli, 2014)(Georgios & Bacchelli, 2014)(Roberts, Hann, & Slaughter, 2006). Data set sizes also varied. In (Goeminne & Mens, 2011), they manually analyzed source code and defect data but, likely due to the effort involved, they only pulled from three small OSSD projects. Approximately 265, 255, and 230 participants were involved in each project. Meanwhile, in our data set high profile OSSD projects such as Hadoop, Spark and Cassadra had unique user counts of ~15k, ~15k and almost ~18k respectively as of early 2015.

We argue that these studies have allowed the open source community to gain a better understanding of how these two seemingly disparate groups contribute in different ways. However, we also argue that seeking greater insight into the complex relationship between the dedicated core developers and the casual contributors would benefit the community in numerous ways, not least of which is a greater appreciation for the essential nature of both population segments but also a better understanding of what roles they both tend to fill with greater skill. Therefore, we set out to build on some of these previous studies but in our effort we used a data type underutilized in previous studies and a data set that was much larger than most.

As many previous studies had focused on the commit patterns found in code repositories (A.K.A. code change logs), we chose to utilize data from a requirements repository system used by one of the most prominent open source software organizations: The Apache Software Foundation (Apache). When we started our efforts, Apache had nearly a half a million tickets, 79,000 users and 448 projects in their system, which made it larger than nearly all of the related studies with the notable exception of (Choetkiertikul, Dam, Tran, & Ghose, 2015). When compared to code repository data, we argue requirements repository data has the advantage of easily distinguishing the different roles a contributor can play. Meanwhile, measures beyond the “who” and the “when” that a researcher might gain from code repository data have more to do with how much code was changed or what module the changes pertained to.

We captured Apache's Jira data by developing a Python based web application that interrogates Jira's REST interface and persists this data in a Postgres database. As some of the key information, namely the change logs, were not available through the REST interface, we also wrote web scrappers to acquire this additional data.

Although we acknowledge that we have no means of asserting that this data from the Apache organization is a truly random sampling of the entire OSSD community, we chose it due to its tremendous diversity in project size and unmatched historical reach. Our sample set includes seven projects with over 10,000 tickets and 245 with fewer than 200 tickets. The date ranges of activity in this data set exist from October of 2000 through (in our second capture) August of 2015. So if a research effort is to only take from one system, we speculated that Apache's Jira instance would be the most representative set to the OSSD community as a whole.

Lastly, another dimension of our study that differentiates us from most of those that came before is that not only did we demonstrate a relationship between how a contributor's contribution level influences the roles they are likely to play but we also addressed how effective they are likely to fulfill these roles successfully based on various indicators of their experience level. As an area of focus, we took inspiration from (Goeminne & Mens, 2011) and investigated the area of requirements writing in the second portion of our analysis.

After reviewing some previous work and clarifying the source of our data in Section 2.2, we briefly describe the data we acquired from Apache in Section 2.3. Then in Section 2.4 (Phase 1) we conduct exploratory analysis on the contributors of Apache to show how common the casual contributor is as well as show how the most active users vary in their contribution both in terms of change counts and how long they are involved (A.K.A career spans). In Section 2.5 (Phase 2) we look into a few hypotheses about the different roles contributors fill and how we propose to study the effectiveness of such contributors. Specifically, we address the writing of requirements as (Goeminne & Mens, 2011) was one of the few studies pertaining to OSSD workload inequality that took into account this measure. Finally we conclude and elaborate on where these findings and our data may take us in the future.

2.2 Background

For decades, researchers have been harvesting data from any tool involved in the development and management of software products and referring to it as Mining Software Repositories (MSR). Using data from the source code repository SourceForge, (Bantelay, Zanjani, & Kagdi, 2013) and (Comino, Manenti, & Parisi, 2007) attempted to evaluate and track the success of OSS projects, which is a pursuit we find to be multidimensional. (Comino et al., 2007) in particular found that, "the size of the 'community of developers' increases the chances of progress but this effect decreases as the community gets larger, a signal of possible coordination problems." (Giuri, Ploner, Rullani, & Torrisi, 2010) also used SourceForge data to infer the abilities of OSS contributors from their activities and found that, "skill heterogeneity is associated with project survival and performance." Apache was also the subject of (Roberts et al., 2006) where they used change logs to measure a participants contribution level and then a survey to find relationships between their standing and proclaimed motivation factors. They found that, "developers' contribution levels positively impact their performance rankings." We find that all of these research efforts brought great insight into Open Source Software (OSS) community but they were still very focused on the developers and the updating of code.

In (Gousios, 2008) they proposed, "a new model for evaluating developer contribution, which takes advantage of software development repositories to extract contribution indicators" that utilized code repository data, requirements repository data and email list information. They also emphasize "other activities that do not directly affect the product itself but are essential to the development process." We argue this supports our choice to leverage requirements repository data instead of code repository data.

Many papers utilized similar data to pursue better methods of using a requirements repository tool such as (Schugerl, Rilling, & Charland, 2008), (Lamkanfi, Demeyer, Giger, & Goethals, 2010) and (Weiß, Zimmermann, & Zeller, 2007). Others used requirements repository data to gain better insight into how different teams distribute task assignments (Jeong, Kim, & Zimmermann, 2009) or how management oversees the variety of requirements differently

(Knauss, Damian, Poo-caamaño, Cleland-huang, & Victoria, 2012). (Steinmacher et al., 2013) tracked the activities of newcomers to an OSS project and attempted to link the nature of their interaction with others to find a relationship with the quality of those communications and the likelihood they would become long term contributors. In our research, we draw inspiration from all of these efforts but in particular those that pertain to the division of labor at the macro-level and specifically within the OSS community.

When utilizing empirical data over a group or over a significant period, data doesn't always just form metrics but it can also provide insights. This leads to the area of software engineering research known as "Software Analytics." This area focuses on improving the use or development of software by turning vast amounts of raw data into "actionable insight to inform better decisions related to software (Buse & Zimmermann, 2012)." As we are setting out to compare different segments of the OSS community by many measures, we set out to create our own "Software Development Analytics" to gain a better understanding of the OSS contributor population as a whole and we had many previous works to draw from.

Other efforts used requirements repository data to form software analytics over the OSS community that allowed them to gain insights into the dynamics of the population and their impacts. In (Hayashi, Ihara, Monden, & Matsumoto, 2013) they were able to establish that ~50% of committers[4] in an OSS project are familiar with just one module and 75% of committers know less than five modules. We speculate that this tendency toward specialization may be one of the explanations for the workload inequality that we examine in this paper. In (Hayashi et al., 2013) they also examined the relationship between the instances of a proposed solution to a defect that had been rejected in some form (A.K.A. a "re-opened bug") with the level of collaboration surrounding that piece of work. They concluded that the increased levels of collaboration do increase the possibility of re-opening bugs.

Developing analytics on the progress of a project allowed (M Zhou et al., 2016) to measure the impact of the departure or retention of "key contributors" have on the long term success of an

[4] Committers are the contributors that propose code changes.

OSSD project. In (Herzig, Just, & Zeller, 2013) they found 33.8% of all bug reports to be misclassified. If a ticket marked as a bug turns out to be a new feature, prediction models and planning are impacted.

In summary, MSR studies have brought great insight into the Open Source community and Software Engineering in general. Some have addressed the topic of workload inequality but our study differs from others in terms of the type of data we are leveraging, that we are not just focusing on requirements writing but that there are differences in the type of requirements written by different groups and that we address the effectiveness of this vital role of requirements writing. We also argue our study is a special contribution in that we use a dataset of unusual size. In addition, our research is in the minority of the vast body of empirical work software engineering research because as (C. Jensen & Scacchi, 2007) claim, "Studies of OSSD processes are increasing in number, while OSSD organizational structures, technical roles, and career opportunities are much less studied."

2.3 The Data within Requirements Tracking Repositories

Not long ago, the details describing new features of an application or software system would usually be articulated in some form of a requirements document; meanwhile the defects discovered in the delivered product would usually be tracked in a "Bug Tracking System" (BTS). More recently it has become common place to attempt to break a new feature into pieces such that an individual developer can commit to delivering the piece in a timely manner. These deliverables are best managed using an application that communicates their details and priority, as well as tracks the person that is assigned to them, and reports their progress just as defects were in a BTS. Therefore, requirements for new features are now often managed in the same applications as the details of the defects discovered in these deliverables. So for our purposes, we are interested in project management data that pertains to new features and defects but we will refer to it all as coming from a requirements repository.

Although utilizing requirements repository data in research data has many challenges, (Canfora & Cerulo, 2005) argue that requirements tracking data from OSS projects will be the most

comprehensive because the contributors in an OSS project are completely dependent on the tracking tool to organize and negotiate requirements. Unlike industry projects, OSS contributors are almost never in the same room and they rarely know one another so an OSS instance of Jira, or another requirements repository, will have a data set that is much more complete than in a commercial setting where verbal communication can make brevity a temptation (Mockus, Fielding, & Herbsleb, 2002a)

2.3.1 The Tables

In a requirements repository**,** *Tickets* (sometimes: Work Tickets) are effectively units of work that a software engineer will attempt to deliver to the project. They include a description of what is usually a new feature, an enhancement or defect (bug). Requirements also live in the dialogue represented by in comments, the summary (title) and sometimes attachments.

Tickets belong to *projects* and are often tagged to involve at least one part of the product called a *component*. *Users* write, evaluate, deliver, verify and generally manage these tickets. In most systems, every change to the ticket is tabulated in an event that we generically refer to as a *history record***.** Each of these records has a time stamp and the associated user making the change. In the case of a change of assignment, there is also the new user owning the ticket that is associated with this important change.

The workflow for each ticket varies greatly based on the state of the team, the demand for the "change" being discussed, the policies of that project and many other factors. We are especially interested in utilizing these event driven data, as they are our greatest insight into the activities of the contributors.

2.3.2 Table Details

In a previous section we explained how most of the data tables related to each other and what their purpose was. We will now explain the most critical fields within these tables so that the significance of this data can be fully appreciated.

2.3.3 Tickets

The focus of Jira are the tickets, each of which has a list of "required" fields and quite a few optional ones that bring us great insight, however infrequently. The ID and the key uniquely identify the ticket. Three time stamps are available: "Open Date," "Update Date," and ideally "Resolved Data" but not every ticket gets resolved. Similarly, the user that created the ticket is recorded as well as if the ticket is assigned to someone, which occurs almost exactly 2/3rds of the time at the Apache Foundation.

The fundamental text of a Jira ticket is not our focus, but we did capture how many characters are in the tickets title (A.K.A. summary) and its sometimes lengthy description just as with the comments. Every ticket also has to be assigned to a project as well as have an IssueType, priority and status.

The choice for IssueType has many options, but this field is usually used to distinguish "Defects" (A.K.A. Bugs) from "Enhancements" or "New Features." There are also rarely used custom IssueTypes that are of interest, particularly those that might indicate the use of a certain process.

Priority represents the relative importance of a ticket to other tickets. Officially, the choices are "Highest", "High," "Medium," etc. But these labels are customizable and the choices most teams make on the labels isn't nearly as interesting as how often they choose the six given levels of priority and who on different teams determines this choice.

Status is the most fundamental way to indicate where a ticket is in its workflow.[5] Officially, a ticket starts as "Open" and once completed us usually moved to "Resolved." When this proposed

[5] This and many other specifics of various field uses are best found at https://confluence.atlassian.com/display/JIRA/What+is+an+Issue

change is confirmed, it is supposed to be “Closed.” There is also the option to customize the status types.

2.3.4 Users

Although a key focus of this first study with our system, the actual data contained in the user table is minimal. It is effectively identification information such as username, email and the all-important: time zone. Our objective was to accumulate analytics based on a user’s interaction with various tickets, especially with regards to the changes they make recorded in the events table.

2.3.5 Events

The record in the Events table always involves a timestamp, a change to a ticket and nearly always a user. Much of our analysis will pertain to this data that is often excluded in MSR papers. So far there are 108 different types of changes to a ticket in the Apache Foundation, but the most interesting are changes in Status, Resolution, User Assignment, and Versions as they are key changes in a ticket’s workflow. In this paper we focus on the resolution and especially creation events.

2.3.6 Projects

The Project object has a key and name but its effective purpose is to aggregate a list of tickets and the users that contribute to those tickets. Each project has a different history and that is occasionally relevant. For instance, Apache Flex is *by far* the biggest project but it is also special because it was a funded project at Adobe Systems for 7 years before being donated to Apache in 2011. For confidentiality reasons, all the users in the project during the Adobe years are tagged together as “adobeJira”. Therefore many of our metrics will not be valid in this large project because dozens of developers and countless more contributors are represented by one user tag.

2.4 Phase 1: Exploratory Analysis

Now that we have given an overview of the most critical fields in four major tables, we will show how this data can be used to demonstrate the great variety in user activity and to present a few

software development analytics, some of which we will use in phase 2. We refer to these metrics as “User Activity Analytics.”

2.4.1 The Casual Users

An important attribute of this data that needs to be taken into account is that the vast majority of users only make one change to the system, just as was established in 2002 by (Mockus et al., 2002a). This would appear to be because they just had one suggestion and therefore wrote one ticket. In fact, of the 71,900 users that have ever contributed to the Apache Software Foundation as of January of 2015, 34,762 made less than two changes to the system.

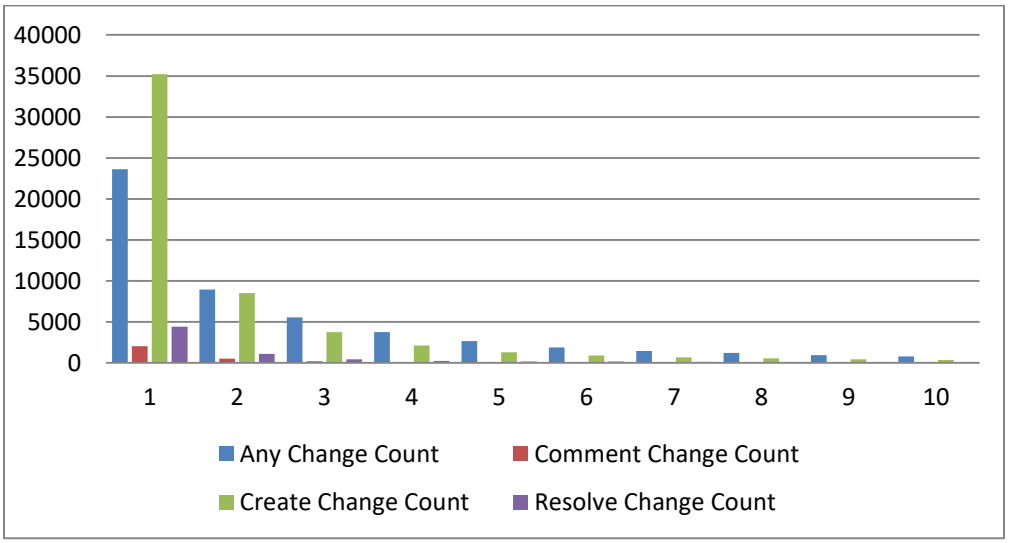


Fig. 1 A histogram of the activity counts for 10 Count or less shows how many users only make one contribution, usually creating one ticket

By utilizing the Events table, we were able to track the total contribution of each user in terms of “any” contribution, creating tickets, resolving them and making comments. We place all four categories in a histogram with a logarithmic scale in Fig. 1 to illustrate the severity of this special group of what we refer to as “casual contributors,” but it’s important to understand exactly what this is indicating. For instance, of all the users, 35,235 have created only one ticket. Meanwhile, 913 have created exactly six tickets. In contrast, 4,435 have resolved one and only one ticket but 150 have resolved exactly six tickets.

What we found intriguing is that the count of users that had only one change of “any kind” is lower than the count that has created only one ticket. We speculated this is because those that created

a ticket are likely to make at least one non-create change, thus removing them from the only-one-change group. Of course, one count toward “any Change” could be as simple as updating the priority setting or tagging the ticket with a component after creating the ticket. As it turns out, only 36.2% of users who only create one ticket end up making an additional non-create change, but this enough explain the finding.

Some additional analytics that give insight into the casual contributors are what we called the “conditional count.” By looking at the counts of only those that made 1 change anywhere, we established that 95% of them were creating a ticket and none of them were resolving a ticket. The additional columns in Table 1 count the number of incidents that a user had “at least one” of the following activities, given that they had 1, 2, etc. changes total. So for instance, even once the activity count got to an unimpressive six changes, only ~19% of users had resolved at least one ticket. We argue this gives serious insight into the focus of the casual user and the focus clearly seems to be writing a ticket of new work.

Total Activity	1	2	3	4	5	6
Any Change Count	23635	8961	5543	3738	2661	1890
Comment Change Count	129	110	89	121	106	110
Create Change Count	22469	8452	5278	3584	2554	1814
Resolve Change Count	0	57	690	580	442	354

Table 1 A count of number of users that participated in a specific activity, given that they had a total activity count of 1, 2 through 6

One caveat worth noting is that some casual users may have registered for more than one username, thus exaggerating the relative size of the casual user group. A possible future study might be to use text mining to search the user records for similarities and ideally establishing the likelihood of duplicate user records. However, we suspect more significant contributors would be less likely to have more than one username simply because of the clout and recognition associated within the work they contribute.

So it is clear that there are a lot of contributors that make only a few changes but these are largely folks that are making a suggestion for a new feature, an enhancement or in all likelihood reporting a defect. There is still a significant amount of data pertaining to the active users and, as we have shown, they are doing most of the work.

2.4.2 The Active Users

As we wanted to differentiate the casual users from more active users, we set about proposing criteria. The challenge is: how do we identify them? Do we draw a line on total changes ever? This is convenient in that it is general, but this presents weaknesses by equating making a comment to resolving a ticket. And how do you compare users that have made a few dozen contributions over the course of a decade to a user who resolved a dozen critical tickets during a crucial time in one projects history but was never heard from again? We concluded that it is challenging to define an active user and it requires multidimensional analytics to do so as supported by (Shibuya & Tamai, 2009) and (Gousios, 2008). In order to gain insight into the type of contributions a user has made to the Apache project, the analytics we propose include the aggregate contribution a user has made and the time period this covered.

2.4.3 Active User Activity counts

To investigate active user contributions, first we sorted the users based on of the four different criteria counts mentioned before in “any contribution,” resolving tickets, creating them, and making a comment. By grouping the users into half percentiles and viewing the population in these 4 different rankings we showed that nearly all the work is done by the “top 5%.” As the first histogram was so skewed, this should be no surprise. In fact, out of every user, the only ones that have even proposed a solution to a ticket (A.K.A. *resolved* even one ticket) are in the top 88 percentile. The extreme values of this top 5% are depicted in

Table 2.

Commenting was originally tracked as we suspected it would be a key indicator of roles such as requirements negotiation and planning. Other studies have focused on email lists to study OSS

communication (O'Mahony, 2003) (Gala-Pérez, 2013) and these low comment counts indicate that most communication is not done in comments contrary to what we suspected.

Interestingly, this data shows that for a small segment of the population (the upper 99 percentile), resolving tickets is more common than creating them even though the vast majority of users in the system do not resolve tickets at all.

Percentile	Comment Change Count	Create Change Count	Resolve Change Count	Any Change Count
95.0%	0	14	2	50
95.5%	0	16	3	60
96.0%	1	18	4	75
96.5%	1	22	6	97
97.0%	1	28	10	131
97.5%	1	35	17	183
98.0%	1	46	30	277
98.5%	2	66	55	446.03
99.0%	2	99	110	811.01
99.5%	4	183	257.505	1731.505
100.0%	201	33057	29862	183195

Table 2 Activity counts based the four activity areas of interest, sorted in different percentiles to show the disproportional amount of work accomplished by the top 5%, 2%, and especially 1%

Based on this table, we conclude that to refer to a user as being in the top 5%, top 2% or top 1% of either total changes, total resolved tickets or even total created tickets is to utilize a strong analytic that indicates a user has participated at a certain level in essential ways. In other words, if you have resolved 30 or more tickets then you are in the top 2% and you have made a very serious commitment.

Referring to these small percentages may seem like a disappointingly small data set, but we must keep in mind that each bucket of one percent still represents 719 users, so the population that is making the serious contributions is not small in total, but merely small relative to the majority of OSS free loaders who primarily observe but do not contribute.

2.4.4 Active User Career Span

We are also interested in tracking the period of time that a user makes contributions as a reflection of how long their contributions impacted the projects they were involved in. So for

every user, we tabulated the time delta between their first change event and their last, along with the deltas for resolve events and ticket create events. We refer to these three measures as "User Career Spans" and argue these measures reflect the amount of time a user was involved at Apache.

Not surprisingly, as so many users only make a few changes or less to any project within Apache, the vast majority of them participate for less than a day. Even if we only look at those 37,138 who made at least two changes, 40% of them did so within a week and were never heard from again. Of the ~9,000 that made exactly two changes, 72% of them did so within a week.

In contrast, of the 10,749 users that made at least ten changes, almost 95% did so in a period that spanned more than a week. Also, of the ~19,000 users that made at least 5 total changes, only 17% did so within a week. However, when you look at the time period that these same users *resolved* tickets, nearly 80% of them did so within a week. We conclude much of these contrasting results are because those that resolve more than a few tickets are in such a small minority within that 5+ change group. Fig. 2 shows a histogram of time in the system for those making at least 5 changes utilizing a logarithmic scale to compensate for the large counts at less than one day. As Figure 2 suggests, the most common time for this group of active users to be in the system is less than a day, even though 83% of these users participate longer than that.

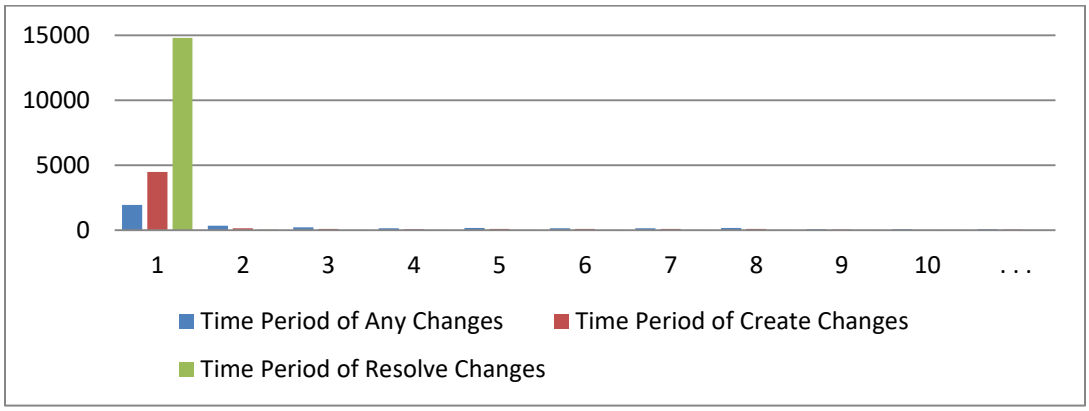


Fig. 2 Histogram of Career Spans in days for users making over five changes shows how incredibly common it is to only participate for one day, even among the contributors making at least five changes

Focusing on the top 5% of most active users we find participation levels with more inspiring career spans. In terms of users that made total changes of at least 50, of these 3600 users, 21.5

% did all of their resolving within a month but well under 1% of the same group did *all* their changes within a month. This reinforces the notion that many folks in the top 5% are helping in ways other than resolving bugs. As for the top 5% of users in terms of tickets resolved, which includes those having a resolve count of as low as two, we found that only 20% of this group of 4511 resolved all of their issues within a month. When we look at the approximate top 5% in the area of creating tickets (at least 14 tickets), only 0.85% do all their creating within a month. So the top 5% most active contributors when measured by total changes, resolved changes and especially created tickets, have career spans of more than a month for the vast majority of the instances.

2.4.5 Insights Pertaining to the Core Contributors

We argue that there is no universal definition for a core contributor but we find the most active contributors fascinating in numerous ways. Users that have made at least 500 changes are less than 1.5% of the Apache population but they are the ones that have some serious longevity. By a similar measure, slightly more than 1.5% has resolved at least 50 tickets. Unlike previous user behavior analytics discussed, this group's career span distributions are not nearly as skewed. These critical contributors stick around for an average of about five years and resolve tickets for about four of those years as shown in Table 3 The career spans in months of core contributors based on two criteria: those that made over 500 changes and those that resolved more than 50 tickets. Both are approximately in the top 1.5% for those criteria

.

Total months participating	Any Change > 500	Resolved > 50
mean	59.97	59.57
median	53.53	53.58
Total months resolving		
mean	48.60	48.41
median	40.22	39.88
Total months creating		
mean	56.92	56.03
median	50.58	50.16

Table 3 The career spans in months of core contributors based on two criteria: those that made over 500 changes and those that resolved more than 50 tickets. Both are approximately in the top 1.5% for those criteria

A histogram in Fig. 3 shows this distribution is heavily weighted around the two year mark with a fair amount of core contributors represented for careers lasting almost a decade. Visually, we can see a very small population that manages to contribute at this level but do so in one or two months. We also see that only a few have managed to be this active for more than one month but less than a half year. The more common career span for these 1013 participants starts at around 18 months.

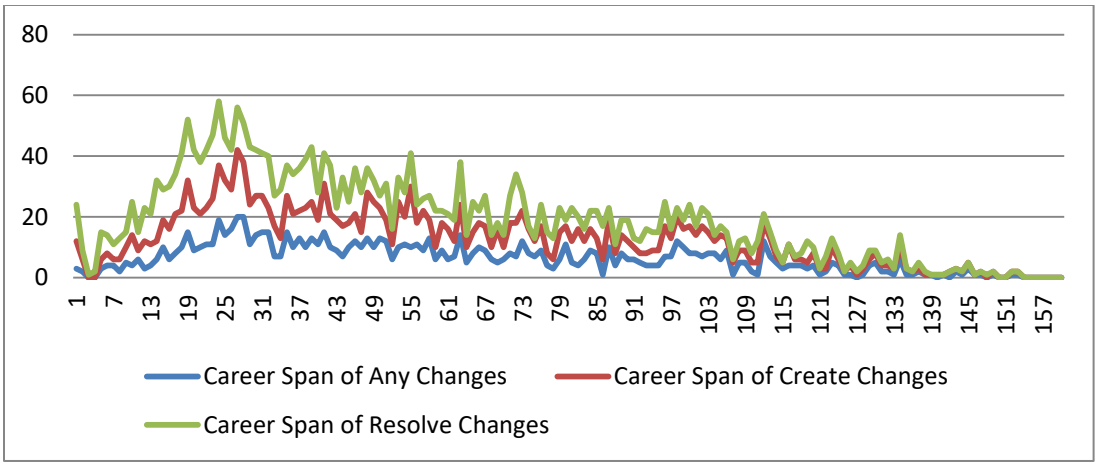


Fig. 3 A histogram of career spans in months of the core contributors (over 500 changes) shows how common a career of approximately two years is relative to the career spans of other core contributors of Apache

So based on total time participating in various activities, this top 1.5% group stands out as a strong example of the core contributors (also sometimes: long term contributors) mentioned in previous works.

2.5 Phase 2: The Effect of Experience on Requirements Creation Effectiveness and Type

The two primary purposes of writing a Jira ticket are to report a defect or to suggest an improvement. To investigate the OSS community's collective effort in this area, we confirm some of our assumptions and test the hypotheses regarding the requirements creating contributions of different groups in the open source development community. Specifically, we describe some of the relationships between the experience a contributor has and the level of effectiveness we might expect from them in terms of writing the different kinds of tickets.

As time had passed from our first phase, we were able to task our system with acquiring new data through to August of 2015. In this phase, we had 530,982 tickets and 85,131 registered users to draw from.

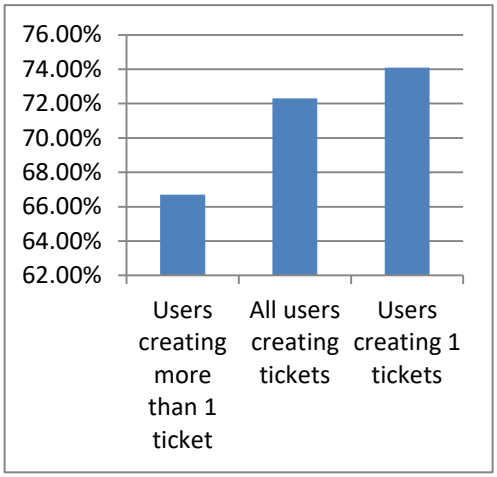


Fig. 4 Contrasting the User Defect Ratios of the large pool of single ticket creators, those that create more and all users that create

2.5.1 The One Time Contributor and the Essential Quality Role

In the previous section, we explained that 48.3% of the users who have used Jira to coordinate their contribution to the Apache Software Foundation through January of 2015 made one or zero changes. As of August 2015, there were 32.0% who made the singular contribution and 16.6% made zero contributions. This provoked our first research question:

"What is the primary activity of these users that go to all the trouble of registering on Apache's Jira to make only one change?"

We speculated that they were using an Apache product and had a suggestion for the collective development team. We also speculated that a large portion of these suggestions were defects they had found while using the product. Simple queries on this August 2015 data confirmed the previous finding in that 94.6% of the 27,238 single change contributors were writing a ticket. This second assumption regarding the type of ticket by this group was also confirmed because 74.1%

of the 40,389 tickets from contributors that only created one ticket (but they could have made other changes) were tickets identifying a *defect* in the product rather than an enhancement or new feature as shown in Fig. 4.

Of course, the figure of 74.1% of single ticket writers is especially interesting when we contrast them to all users that created *more* than one ticket. In this case, we refer to the "user defect ratio" as the ratio of tickets created per contributor that were defects. Of course, as a group this is effectively accomplished in the previous statistic by averaging zeros and ones.

$$user\ defect\ ratio = \#\ of\ tickets\ this\ user\ created\ as\ defects\ /\ \#\ of\ total\ tickets\ this\ user\ created$$

Of the 27,258 users writing *more* than one ticket, the average ratio of defects was 66.7%. Although this difference is not tremendous from the single ticket group, a one-sided hypothesis test of these two samples has a p-value of well under 0.001% so the probability of the measure of defect ratios not being different between these two groups is miniscule. The average defect ratio for the entire population of creating users is 72.3%.

What is striking is that the portion of tickets that are defects in the entire system at the point of this second data pull is 57.4%. This led to the suspicion that although the average ratio of defects per person might 72.3%, there had to be a small but tremendously active group that was heavily involved in writing non-defect tickets so we attempted to identify this group.

From section 2.4.1 we know that the ticket count totals for each user in creating as well as resolving tickets are tremendously skewed toward the most active users. This showed that a tremendously large portion of the total contributions are given by the most active (especially the top 5%) of the user population. To highlight the difference in user defect ratios between these most active groups and the more typical user we sorted the users by how many tickets they wrote. Then we chose to bucket the users into 50 quintiles (or groups of 2 percentiles) and tracked the average defect ratio for that slice of the population. This served as a smoothing function but also allowed us to highlight the user defect ratio of these most active users instead

grouping them in a simpler manner like we had with the “more than one ticket created” group. This more refined grouping confirmed our suspicion about the proportion of non-defect ticket creation the most active users contribute. To further highlight how special the top 5% is, we also plot the average number of written tickets in each 2% bucket. The difference in ticket writing in top percentiles is remarkable.

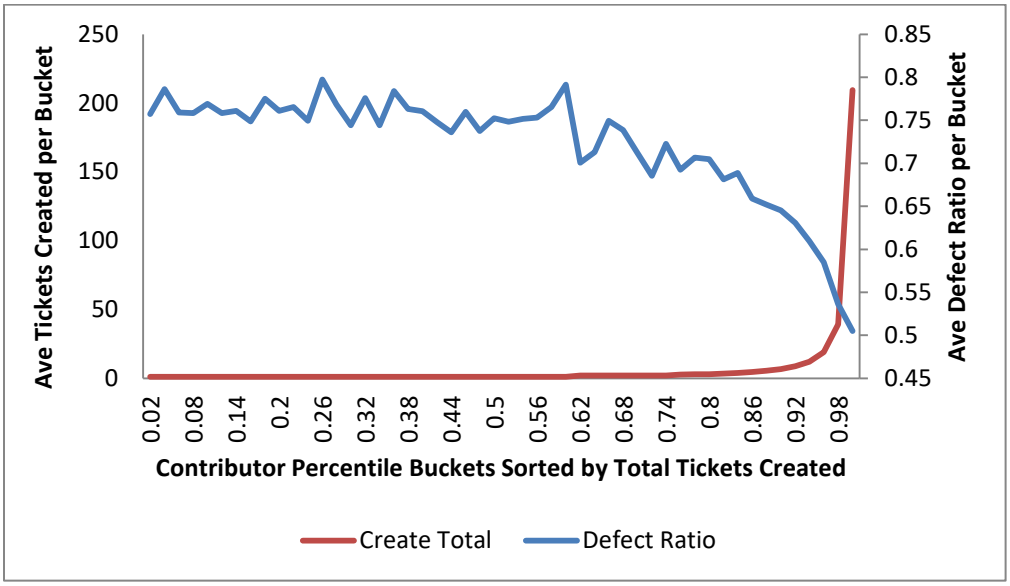


Fig. 5 Depicting the defect ratio of all ticket creating users bucketed in 2% groups sorted by total tickets created, along with average ticket created total within that bucket

Of course, the first 58% of the population only wrote one ticket and therefore their averages per bucket are rather consistent just as the next grouping to 75% is for those writing two. The drastic change really only occurs in the top 20% of the population and we plot both the average defect ratio and the number (average of quartile) of created tickets in the given quantile, this time as measured per one half of a percent in Fig. 6. This shows that, on average, the top 1% actually writes a bit more non-defects rather than defects.

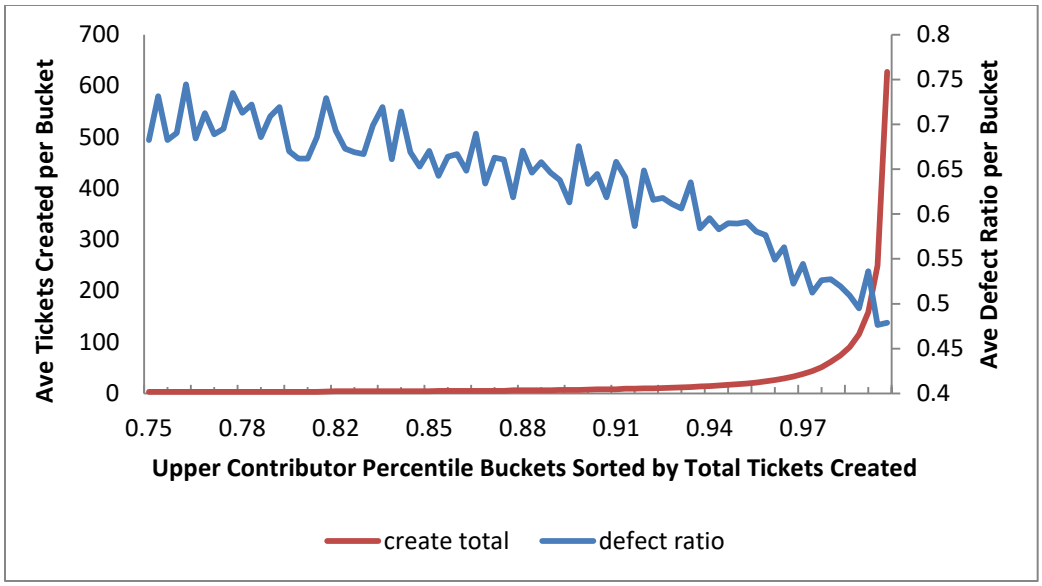


Fig. 6 The defect ratio of the top ticket writing users bucketed in 0.5% groups and sorted by total tickets created, along with average create total in that bucket which shows only the very top 1% actually creates more non-defects as a group.

So as a user approaches the level of experience that is indicated by writing more than a few tickets, it becomes less and less likely that they will write a ticket as a defect but it is still the dominant case until the very top percentile where the average user is literally writing hundereds of tickets.

2.5.2 Dedication Level and its Impact on Effectiveness

Within this Jira data, we claim that one of the indicators of a contributor's effectiveness as a requirements creator is whether or not an actual solution was proposed for the ticket they created. This is usually indicated with a mark of "fixed." Unfortunately, many other scenarios exist. A decision maker can decide that the new requirements or the description of the defect are not valid or a duplicate. Specific to defects, a decision maker can also simply mark the ticket "cannot reproduce." The ticket can also be marked with the blunt and feckless mark of "won't fix." And some tickets are simply "in-progress" because the ticket is claimed and the solution is being sought but said solution is not yet complete or, especially in certain projects, the ticket can also simply be ignored. Aside from being ignored, all of this information pertaining to how the ticket

was eventually received by the decision maker (who is often person who presents the first solution) is obtained by the field of “Resolution Type.”

Fortunately most installations of Jira require a resolution type to be chosen when a ticket is closed or resolved. Therefore all the tickets that are missing a resolution type officially indicate that a solution is underway or no one has considered it worthy of their time just yet. As only 0.36% of tickets officially have the status of “In progress” yet 22.64% are missing a resolution type, we consider a missing resolution a strong reflection of a ticket being considered an inferior or invalid suggestion. Admittedly, this is not always the case as a ticket simply might not have been seen yet by the proper person qualified to propose a solution. However, we argue the percentage of tickets that a user creates that are eventually marked fixed is a proxy for the contributor’s ability to propose requirements. This is particularly true if their tickets were proposed significantly in the past.

Using this measure, our hypothesis is simply that the users that have created a lower than average number of tickets will, as a group, have a statistically significantly lower “fixed ticket ratio” when we study the tickets they created.

$$fixed\ ticket\ ratio\ = \#\ of\ \text{“}fixed\text{”}\ tickets\ this\ user\ created\ /\ \#\ of\ tickets\ this\ user\ created$$

When we exclude, as we have done in most of these experiments, the monster user adobeJira, the mean number of tickets a user creates is 5.8. Of the 58,955 users that created this many tickets or less, we found that 39.3% of their tickets were marked fixed. Meanwhile the 8,691 users that had created more than the average, 59.3% were marked fixed. The one-sided hypothesis test of these two samples has a p-value of well below 0.001% so the probability of this measure not being different between these two groups is miniscule.

Of course, if this is largely a reflection of the ticket type (defect or improvement) then this is a less valuable metric. As we have shown, a user is more likely to submit a non-defect with the more tickets they submit. However, we found that for the entire Apache population, the tickets that are submitted as defects are just slightly less likely to be marked fixed at a rate of 62.3% versus the non-defect tickets at a rate of 65.2%. What is also intriguing is that, when only considering the users that have created tickets, there is a weak but positive correlation between a user's "fixed ticket ratio" and their "user defect ratio" with a Pearson's coefficient of 0.333.

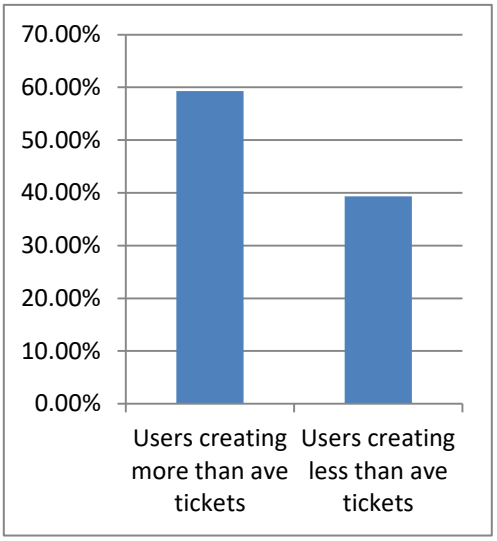


Fig. 7 The Fixed Ticket Ratio of the two groups of those creating more than the mean number of tickets (5.8) and those creating less shows a significant contrast

So the effect of ticket type may be a factor but we argue it is not the dominant source of a more experienced user's tendency to write tickets that actually solicit a solution. We suspect this wide gap between the two groups is a reflection of many attributes of a contributor's OSS career including their standing in the group. This is supported by (Marlow, Dabbish, & Herbsleb, 2013) where the concluded that a contributor's "history of activity across projects . . and successful collaborations" influence how receptive others are to their work contributions. However, we speculate it is also most likely related to their familiarity with the product they are using and writing tickets for. Quite simply, a person that has been involved with a product long enough to write over five tickets, even if many of the initial tickets pertained to defects, has a much better view on what is changes are needed for the future and what is not working currently. To a lesser degree, it might also reflect a user's ability to write good requirements.

Of course, the influences might often be reversed. The fact that a contributor's suggestion was resolved by a developer may impact their willingness to submit more tickets and be motivated to put in the effort to do so in a quality manner. This suspicion corroborates with (Minghui Zhou &

Mockus, 2012) where they observed that if one of new contributor's reported defects was fixed, that contributor was twice as likely to become a long term contributor. Further study would clarify the specifics of why an increase in the number of tickets written tends to improve the likely hood that a user's suggestion is fixed. For now, we have established a statistical relationship between the two factors.

2.5.3 The Career Span Impact

In sections 2.4.2 and 2.4.3 an investigation of career spans showed that the vast majority of users make their contributions within a week, yet in the active users in the top 1.5% participated for an average of approximately five years. Therefore, we speculated that those that participate for an *above* average amount of time yet made a *below* average amount of changes would be rare. As the average total creating career span is 183.6 days and the average number of tickets[6] to write is 5.8 then this long participating but below average contributor group is composed of only 7,343 out of the 85,131 users in the August 2015 data, or 8.6% of the population. Interestingly, the number of users that participated longer than this mean number of days but contributed any number of tickets is only about twice as large at 14,115. So although we suspected the user that had a long OSS career but wrote few tickets was rare, that fact is just having a long career is what is truly rare in this community. This is explained with the highly skewed distribution of career spans mentioned in section 2.4.4.

Once we identified this special group, we arrived at the research question:

> ***"Ignoring the number of tickets a user creates, how does their time in the system relate to their effectiveness as a requirements creator and the types of tickets they create?"***

We hypothesized that career span would increase our two major measures: effectiveness *and* likelihood of submitting an improvement (non-defect) ticket. For the 8,233 that had an above average career span but created at least two tickets (so that we can measure the days between the first and last time they created a ticket), their average fixed ratio was 56.1% compared to the

[6] If we ignore the anomaly of adobeJira

19,015 in the below average career span group that had average fixed ratio of 46.3%. The p-value for this one-sided two proportion test was well below 0.001% so we conclude it is a statistically significant finding. The difference between the two groups also showed a defect submitting ratio of 63.5% vs 68% with a p-value of well below 0.001%, so we also find this statistically significant. We, therefore, conclude that the number of months being involved in the OSS community and not just the number of tickets submitted is an indicator that a user is more likely to write a ticket that will solicit a solution and the ticket is a bit more likely to be a non-defect.

2.6 Threats to Validity

There are two major weaknesses we disclose about our data: duplicate users and missing tickets. Could a user in 2011 have been an early adopter of Hadoop, registered with their work email and then reported a defect? Then in 2015 could they have been a Spark user and used an email address from a new employer and reported another defect? Absolutely. Is there a practical way to estimate how often this happens? Not without choosing one of many convoluted methods to merge user accounts, none of which are perfect or non-trivial as explained in (Goeminne, 2014). We conducted our own cursory analysis of possible duplicate accounts simply based on "display name" in an attempt to measure the practicality of removing all duplicate accounts representing the same human being.

It turns out that our data set has 33 accounts with the display name of "Alex", 26 with "David", 16 with "Sergey" and many other display names of seemingly common first names. If only one of the Alex's and one of the Sergey's were valid, then 8,181 of the 85,133 (or 9.6%) in our second pull of data could possibly be user duplicates but we deemed it impractical to determine which of these were certainly duplicates. So we then only looked at display names of at least two words. Using this two word criteria, 9,048 accounts have the same name as one other, 1791 have the same name as two others, 440 have the same name as three others and so on. Only one name is used eight times: John Smith. Are they all the same person? Probably not. But if we assumed that all accounts with exactly the same display name (but two words) were the same person, which is highly unlikely especially with a name like John Smith, then 6,201 (or 7.3%) of our

accounts are duplicate accounts. In short, we assume that well under 10% but possibly over 5% of the accounts in our data set were representing a person that had already been represented by another username. However, with no means to be sure which we duplicates and which were not, we decided not to merge user accounts.

We acknowledge that this issue does have an impact on our findings, although we argue it's minimal. The career spans of all but the most dedicated contributors are much shorter than the life of an average email address, even if we do assume many would use their work address. Also, there is much clout in the open source world for making contributions. Code contributions in particular are made visible to the public as a career incentive to the more active users. Therefore, we speculate that only the most uninterested parties would not bother to login using the same user account. So although this duplicate account factor may slightly accentuate the number of casual contributors because some portion of the population has registered at least two user accounts over these many years, the analytics on developers with any dedication is much less likely to be impacted due to the culture of open source.

The second notable threat to validity is that there are some tickets, for reasons that we can only speculate, missing in the sequence of "Jira keys." A ticket is auto assigned a key based on the short hand label (project key) of the project name plus an auto-increment value based on the previous ticket written. So by querying every sequence possible using "(project key)-(natural number)" we were able to count the missing data and the percentage of missing tickets between projects varied greatly. Hadoop somehow has 16.4% missing sequences but Spark only lacked 0.1% of the expected sequences. Fortunately, relative to the large projects Hadoop was clearly an anomaly because only 1.9% of the whole Apache ticket population were missing at the time we captured the data.

In short, we believe these activities where tickets were started but then deleted are rare events and have minimal impact on our conclusions.

2.7 Conclusion

In this paper we established that a very large portion of the Apache community contributes by making just a few, and often just one change to a project and that change is often a requirements suggestion(s). These casual contributors usually have a brief tenure in the community; quite often less than a week. This is in stark contrast to what previous researchers have often called the core contributors. This very important group contributes to a much greater extent and we provide some statistics on a thousand or so contributors who wrote dozens if not hundreds of requirement tickets and also provide nearly all of the solutions to these requirements requests.

In the last phase of this study we focus on the pivotal area of creating requirements as a measure of a user's contribution. We confirm the assumption that the vast majority (~75%) of the tickets being submitted by the users who only wrote one ticket (which is 58% of the population) pertained to defects. The ratio of tickets that a user submits that are defects tends to decrease as a contributor makes more suggestions. In fact, at the very top percentile of the most active requirements creators, there are some people who submit more improvements than defects.

We also investigated a contributor's track record of creating requirements suggestions in which a solution was proposed and considered this a measure of a user's effectiveness at creating requirements. It is discovered that if a user created more than the average amount of tickets (5.8) then they were about 50% more likely to have a solution presented to their tickets as a group. Investigating contributors only based on their time in the system (that we call their career span) as an indicator of experience, the users that had an above average career span wrote tickets that solicited a solution ~21% more often. This more experienced group was also slightly more likely to submit an improvement suggestion rather than a defect.

So not only do both the number of tickets a user submits and the time they have spent in the system have a statistical association with a user's improved ability to write a worthwhile requirement, but these measures also increase the likelihood of a user submitting an improvement.

We suspect these measures are often a reflection of the level of product knowledge the user has gained through experience with said product. While some one with minimal experience might see

a flaw in the implementation and present a ticket that represents an almost obvious issue whose value is easily agreed upon, the value of an improvement is harder to determine. So while experience might help a user know which are worthwhile defects to resolve, this may be more so with non-defect tickets. Establishing additional indicators of experience and product knowledge in future research efforts might support this suspicion.

We also suspect that the ability to write an improvement not only requires extensive experience with the product but it might require a reputation to be heard. So these findings may also support the hypothesis that in an Open Source project, the most dedicated contributors are likely to also be the most dedicated stakeholders. It is a very rare scenario in the open source community to get paid for your efforts so in many cases those that care the most about the evolution of a project like Hadoop are also the ones most involved in the conversation.

2.8 Future Work

Collecting and analyzing more data can further validate our research conclusions. The challenge is that Jira allows for heavy customization on each of its instances so if we want this additional data, development efforts will be involved with each new OSS organization or, maybe someday, commercial organization that uses Jira. Another source of data might be all the OSS groups that use the application Bugzilla. It provides simple export functions so the development efforts should be minimal.

Our future work can also leverage data mining and we may use machine learning to predict many useful metrics. Specifically, the temporal nature of this data will make various applications of sequential pattern mining very advantageous. We were able to connect experience to requirements creation ability in a broad sense, but it may go the other way around. Inspired by previous studies that used very different methods (Minghui Zhou & Mockus, 2015)(Steinmacher et al., 2013)(Steinmacher et al., 2015) we could address what events lead to a contributor wanting to contribute more. Could successes lead to that same person proposing more requirements? Of course there could be many other ways to define success, but the initial attempt might be based on this work where we define success as having written a ticket that was

"fixed." Perhaps the more active users have some common events that happen to them that are much less common with those who do not decide to continue contributing.

Previous work (Weiß et al., 2007) made profound progress in predicting how many person hours a defect might take. Taking into account the state of the entire project and team, we are interested in predicting which tickets are postponed. Adding to other works, including (Radtke, Janssen, & Collofello, 2009), that may give insight into the likelihood that a OSS project will continue to progress as opposed to lose traction and cease improving.

But the real opportunity lies in leveraging the many fields we didn't even mention to bring greater insight into the processes and workflow patterns used by different teams. Especially with a greater sample set, we might be able to find indicators of cutting edge processes and measure them against traditional workflows.

3 WORKFLOW MINING THE TASK DISTRIBUTION PROCESSES IN OPEN SOURCE SOFTWARE DEVELOPMENT

3.1 Abstract

Open Source Software Development (OSSD) projects are typically coordinated through requirements repository tools such as Jira. Tasks are divided into work tickets (tickets) and often assigned to participants in the project. We are interested in what impact this action has on moving projects forward and what scenarios are most often associated with both positive and negative outcomes.

We acquired a data set of process log information from the Jira tool as it was utilized by one of the largest OSSD organizations, Apache Software Foundation (Apache), for well over a decade. Many insights are presented including the tendency for the person that created the ticket to have the best track record for determining the ideal assignee for that unit of work. In numerous categories, volunteerism tends to be associated with more positive outcomes than traditional delegation but we acknowledge many other factors could be in play. In the process of investigating the many scenarios where work is distributed, we also discovered a major source of delay in open source projects. On average, a ticket that is never assigned to any one takes 65% longer to complete than when a volunteer completes the give task in the same sitting that they agree to take on the work. However, if the volunteer agrees to complete the work but postpones said task, there is an extremely long tail of procrastination that leads to average cycle times of five to ten times as long, depending on the circumstances, as the volunteer that almost immediately completes the task. We conclude by asking the open source community to re-think how the phenomenon of volunteerism in the community is encouraged and facilitated in hopes to reduce this tendency of participants to prevent others from volunteering while avoiding the completion of the claimed tasks themselves.

3.2 Chapter Introduction

In a commercial setting, a development team might distribute tasks by allowing those to volunteer for a task(s) of their choice or the boss might hand the tasks out. This choice is likely based on the culture of the team but those that distribute the salaries probably have a large say in this matter. But what if no one is being paid? What if there really isn't a boss aside from a few dedicated volunteer project leaders? What if said project leaders have no monetary means to maintain traditional control over their team members?

Open Source Software Development (OSSD) projects are driven by small groups of very dedicated participants (core developers) and large (seas) of "casual contributors" or "drive by committers and all of their contributions are critical to moving the project forward. The aspect of the workflow that we are especially interested in is: After a defect is identified or the details of a new feature are articulated, how does the team decide who should undertake this task? Or should it be taken on at all?

A complete study on workflow in an OSSD project is beyond the scope of any one article but we set out to make significant contributions regarding how work is assigned or volunteered for, what we broadly call "task distribution methods," by utilizing the method of workflow mining. Workflow mining can be utilized to efficiently consider an otherwise insurmountable combination of process steps and determine statistical associations with objective measurements. So although we were not looking to find cause and effect relationships, we did hope to find some process patterns in historical data that increased or decreased the probability of the certain positive outcomes.

After some background and a proper explanation of our data set are given, we describe a few of what we call: success measures. We argue these measures signify positive outcomes in OSSD project workflow. As a major challenge of process mining is converting change logs into meaningful and appropriate abstractions, we discuss our approach before running the algorithms. Then we search our results in order to begin answering a long series of research questions that ultimately led us to the discovery of some process choices that are associated with more positive outcomes. Finally, after finding what appears to be a strong lead for a positive process pattern,

we follow suit with many process studies before us and measure the process in question in terms of a continuous measure: cycle time.

3.3 Background

Data mining is the analytical pursuit of insights from vast quantities of data using methods from statistics, computer science and probability theory. Both the quantity (Krzysztof J. Cios, Roman W. Swiniarski, Witold Pedrycz, n.d.) and format of the data (Han, Kamber, & Pei, 2012c; Tan, Steinbach, & Kumar, 2005; Ye, Baddeley, & Kopelman, 2015)) are a part of the challenge which is why the data mining process includes the collection and cleaning of data before the analysis can begin (Baker, 2010).

When data mining techniques are applied to process logs, it is often referred to as process mining (Rebuge & Ferreira, 2012)(W. M. P. van der Aalst, 2016)(W. Van Der Aalst et al., 2012)(Huang & Kumar, 2007) or sometimes workflow mining (Berlingerio, Pinelli, Nanni, & Giannotti, 2009; Schur, Roth, & Zeller, 2015; Silva, Zhang, & Shanahan, 2005; W. Van Der Aalst et al., 2004)(Li, Reichert, & Wombacher, 2011). In (Sunindyo, Moser, Winkler, & Dhungana, n.d.) they specify that process mining assumes that the data in the event logs contain:

1. A specific activity
2. A process instance (sometimes called a case)
3. A person performing the task
4. A timestamp

Our data follows these requirements as each ticket, which is out case or process instance, has a series of events (activities) tied to it with a user and timestamp associated with each action. Of course, the order of events within a case (ticket) is important but the order of the tickets within the projects are not just as in (W. Van Der Aalst et al., 2004).

The primary purpose of process mining is to discover, monitor and improve processes (W. Van Der Aalst et al., 2012)(Lou et al., 2013). Discovery can come in many forms but, for instance, we might use process mining to discover the different roles participants play in a team (Checking, 2016). Monitoring a process is important for many reasons, not least of which is a reality check as there is often a surprising gap between what team members and management believe is the

process being followed and the processes that are actually utilized (W. Van Der Aalst et al., 2004). It also has been argued by (W. Van Der Aalst et al., 2004) that the goal of workflow mining is to "find a workflow model." We argue this can be a form of discovery and monitoring all in the hope of improving a process. Our efforts are best described in this way.

3.3.1 Our Data

Not long ago, the details describing new features of an application or software system would usually be articulated in some form of a requirements document; meanwhile the defects discovered in the delivered product would usually be tracked in a "Bug Tracking System" (BTS). More recently it has become common place to attempt to break a new feature into pieces such that an individual developer can commit to delivering the piece in a timely manner. These deliverables are best managed using an application that communicates their details and priority, as well as tracks the person that is assigned to them, and reports their progress just as defects were in a BTS. Therefore, requirements for new features are now often managed in the same applications as the details of the defects discovered in these deliverables. So for our purposes, we are interested in project management data that pertains to new features and defects but we will refer to it all as coming from a requirements repository. Therefore, we sought to acquire data from such a system that would allow us to study the workflow of both defects and new features in an open source software project.

Upon surveying the open source landscape, we set our sights on one of the most prominent open source software organizations: The Apache Software Foundation (Apache). When we started our efforts, Apache had nearly a half a million tickets, 79,000 users and 448 projects in their system. We captured Apache's Jira data by developing a Python based web application that interrogates Jira's REST interface and persists this data in a Postgres database. As some of the key information, namely the change logs, were not available through the REST interface, we also wrote web scrappers to acquire this additional data.

Although we acknowledge that we have no means of asserting that this data from the Apache organization is a truly random sampling of the entire OSSD community, we chose it due to its

tremendous diversity in project size and unmatched historical reach. Our sample set includes seven projects with over 10,000 tickets and 245 with fewer than 200 tickets. The date ranges of activity in this data set exist from October of 2000 through (in our second capture) August of 2015. So if a research effort is to only take from one system, we speculated that Apache's Jira instance would be the most representative set to the OSSD community as a whole.

3.3.2 Workflow in Open Source Projects

In order to appreciate the task distribution processes of OSSD projects, we need to establish a basic understanding of the workflow in such projects to begin with.

In short, a ticket is created because someone has identified a defect or they have an idea for new functionality in the product. There is a subtle and debatable distinction within new work as they are divided into "new feature" or "enhancement." The general guideline is that an enhancement is an improvement on an existing feature. In this work we primarily refer to defects and non-defects.

During the creation process, many attributes can be designated to the ticket such as the planned release version or the person assigned to the ticket. When at ticket is assigned to someone, it is effectively saying that the person in question is the next person planned to accomplish something (usually the development work). Some tickets start their lives with an assigned team member and others acquire an assignee later in life. Further discussions on the different states a ticket may be in at the moment of creation are discussed in section 3.6.1.

After work is complete, a ticket is supposed to be designated as complete by being resolved and/or closed. This distinction is clarified in the next section. However, this brings us to thee examples of an ideal workflow. Whether the ticket starts with an assignment or is in fact never assigned, ideally work is accomplished and someone (usually the person that made the code change) indicates the work is complete as depicted in Figure 8. Notice sometimes we achieve completion without an assignment.

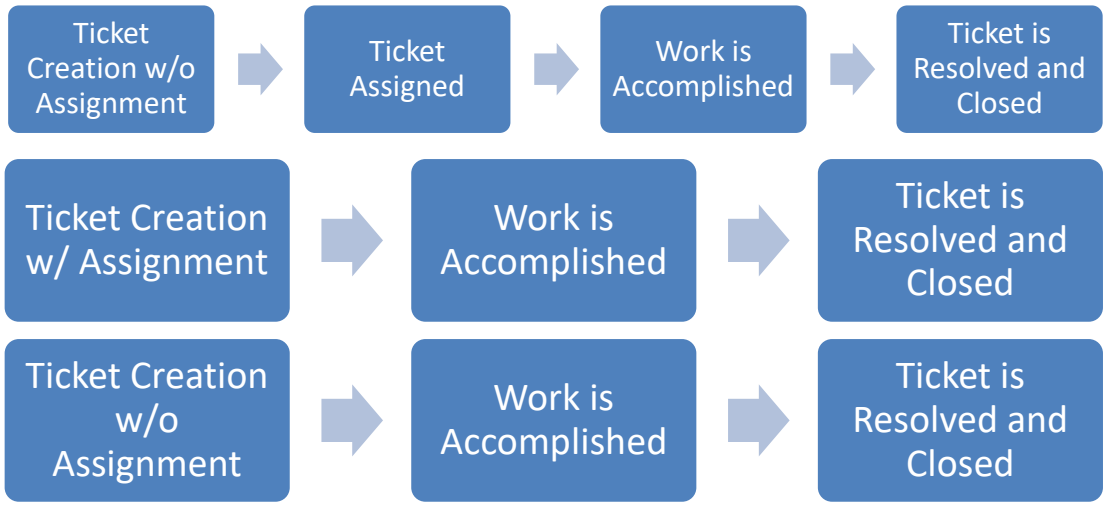


Figure 8 - Three distinct examples of ticket processes that lead to completed work

Although a thorough study of workflow within a Jira ticket's lifecycle is beyond the scope of this article, we also present five examples of non-ideal workflow in the interest of a portraying a deeper understanding of what might go wrong. We include an assignment being changed (or tossed as mentioned in section 3.3) as this often represents wasted effort and the case of a reopened ticket, but our focus in this paper is on when a ticket does not reach competition status at any point and how this may be related to that state of being assigned to a participant. We now present how this data pertaining to the different states our ~500k tickets have been in was utilized to investigate this problem.

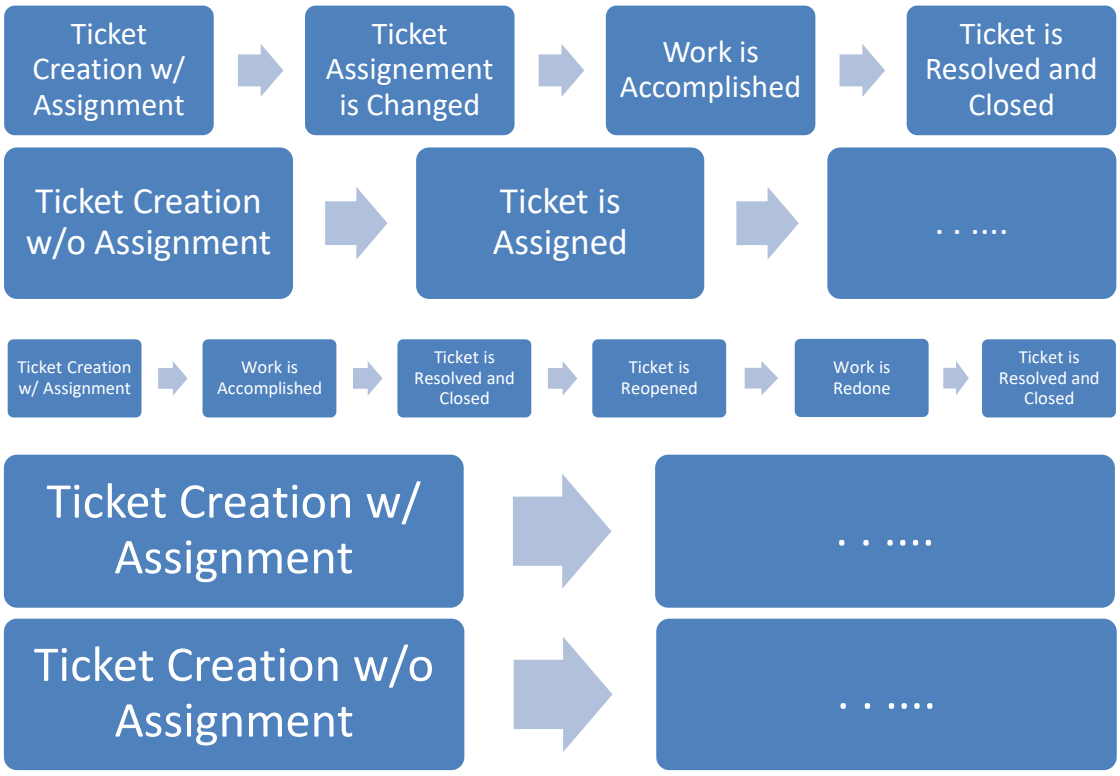


Figure 9 - Five distinct examples of less than ideal workflows, some of which lead to unfinished work

3.3.3 Ticket Ownership and Routing

Of course we are also not the first to investigate the assignment process. Many studies focused on the often disruptive event of an assignment changing from one participant to another (Guo, Zimmermann, Nagappan, & Murphy, 2011)(Jeong et al., 2009)(Chen, Wang, & Liu, 2011)(Wang, Zhang, Yang, & Wang, 2013) This is a concern for many reasons, not least of which are the delays associated with these assignment changes (Chen et al., 2011).

Methods varied. (Jeong et al., 2009) introduced the use of Markov Chains to capture the assignment history of a given requirement. Through this advanced analysis of work flow within a project, they were able to establish networks of developers and realize team structures from the context of ideal work assignments. In (Shao, Chen, Tao, Yan, & Anerousis, 2008) they also used a Markov models to investigate the, “possibility of improving ticket routing efficiency by mining ticket resolution sequences.”

Of course, like our efforts, many (Shao et al., 2008) focus on the event log data and completely ignore the content of the tickets. In (Wang et al., 2013), they argue that ticket assignment and

tossing are a form of collaboration that supplements commenting and other forms of communication. To demonstrate this, they developed “a framework for representing and analyzing developer collaboration in bug repositories based on heterogeneous developer networks.”

Outside of the tossing issue, some set out to improve initial assignments. By taking into account a developers previous contributions and the vocabulary found in the source code, the verbiage in the bug report was used as a predictor for the best assignee in (Matter, Kuhn, & Nierstrasz, 2009). In (Crowston, Li, Wei, Eseryel, & Howison, 2007) they used inductive content analysis to establish that “self assignment” was the most common task-assignment mechanism used in OSSD projects.

So much work has been done on re-assignment of tickets (tossing), some on predicting the best initial assignee, and others made great contributions by gaining insight into how OSSD teams collaborate with regard to assignments. However, we have not found any work that focused on the process of assigning and re-assigning tickets through workflow mining methods. So we have set out to establish what sort of insights this approach might bring to this area of OSSD organization.

3.4 Success Criteria

The major challenge to any sequential pattern mining attempt is how to go about reducing all the relevant data into abstractions such that the when the common sequences are discovered, some subset of them pertain appropriately to the subject you are pursuing. This needs to involve the preceding events (antecedents) and the resulting events (consequent events). As we were investigating all events involving a ticket assignment, we needed a means to measure the eventual success, or lack thereof, of each ticket in an objective way. The algorithms of sequential pattern mining also do not allow for continuous variables. At the very least, measures must be reduced to ordinal or categorical variables.

After investigating a few options, we chose to focus on indicators that show whether or not development efforts were taken after a requirement was written. Specifically, the event we are

primarily interested is when the ticket was declared finished. The challenge is that there is more than one way to indicate this and no one can agree on the proper way. It is largely based on team preference. Either way, an official status is changed to either Resolved or Closed. In this study, when measuring a specific sample of tickets, we refer to the status of being finished or completed as “closed” and the percentage that has either one as the “Closed Rate.”

RQ1: How often are either of these two statuses given and therefore how often are tickets Closed?

Within our data set of 530,982 tickets, 51.3% have been marked closed, 57.7% were marked resolved and 27.5% were given both statuses at one point in time. The overall closed rate of the entire history of Apache is 81.6%.

When a ticket is marked as either closed or resolved, ideally this means a solution is presented. Unfortunately, many additional scenarios exist. A decision maker can decide that the new requirements or the description of the defect are not valid or a duplicate. Specific to defects, a decision maker can also simply mark the ticket “cannot reproduce.” The ticket can also be marked with the blunt and feckless mark of “won’t fix.” And some tickets are simply “in-progress” because the ticket is claimed and the solution is being sought but said solution is not yet complete or, especially in certain projects, the ticket can also simply be ignored. The ideal scenario is that the resolution is marked “fixed” because code was presented to make the enhancement, add the new feature or solve the defect.

Aside from being ignored, all of this information pertaining to how the ticket was eventually received by the decision maker (who is often the person who presents the first solution) is obtained by the field of “Resolution Type.” Of the 433,059 tickets that had ever been declared closed, only 0.6% were lacking in a resolution type so we assert it is a reliable means to determine under what context the ticket was closed. As the sought after “fixed” resolution type indicates, amongst other things, that the requirements in the ticket were worthy of some ones effort, they represent a likely workflow that will be different from the other rejected tickets. Therefore, they are worthy of special consideration. So this rate of tickets being fixed and the previously explained closed rate are our two “closure measures.”

With this fundamental success criteria established, we are prepared to ask :

RQ2: What is the relationship between the practice of assigning tickets and the completion indicators? OR Does assigning a ticket increase its odds of ever being closed?

In the ticket population as a whole, ~73% are assigned to a user at some point. Of the population of tickets that are assigned, ~90% receive a closed status at some point as opposed to the population of unassigned tickets who achieve a closed status at a rate of 58.5%.

Of this same population of assigned tickets, 77% eventually are marked with a *fixed* resolution as opposed to 32% in the unassigned population. We speculate, based on this very simple Boolean based comparison, assigning a ticket may have an even stronger association with tickets that avoided the non-fix resolutions. We think this might be partially due to some teams filtering out the non-valid or less than worthwhile suggestions from the populous without bothering to assign them to everyone. Queries on the non-fixed yet closed tickets support this. 12.9% of assigned tickets were non-solution closures meanwhile of the not assigned tickets, 26% were non-solution closures.

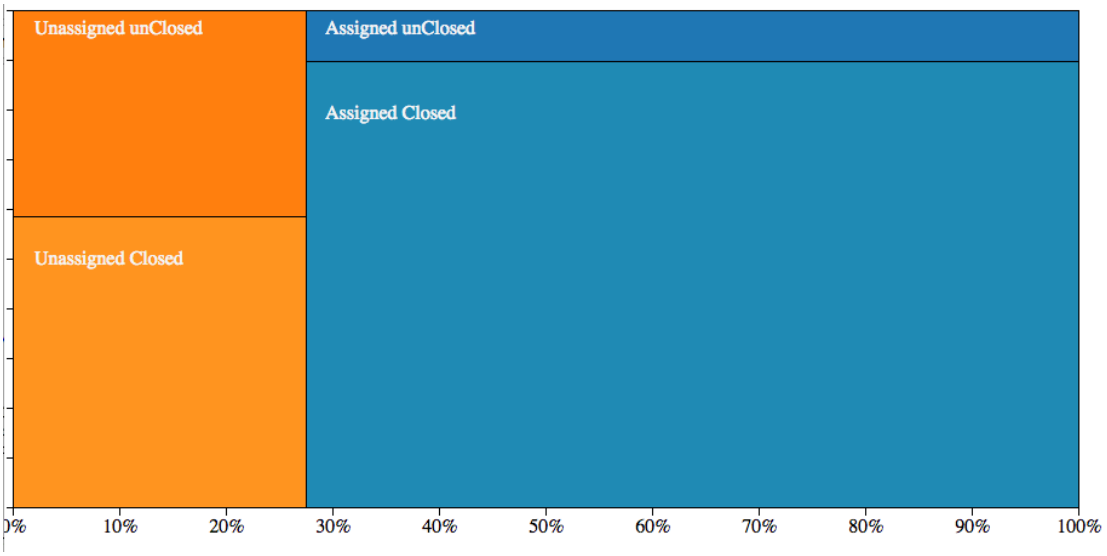


Figure 10 - Our mosaic plot depicts the proportion of tickets that are assigned at some point in their history and the proportion of tickets that are closed at some point. The assigned tickets have an overwhelmingly larger proportion of tickets that are eventually closed

These findings led us to suspect there may be a relationship between the eventual resolution status of a ticket and whether a ticket was ever assigned. The following research question then followed:

> **RQ3: Is a ticket being assigned because it avoided a non-solution resolution or is a ticket more likely to be considered worthy if it's assigned?**

That is to say, one possible reason there seems to be a correlation between assignment and our success measures is that tickets are sometimes determined to not be worthy of the team's efforts are effectively rejected before it becomes worthwhile to assign the ticket to anyone. Alternatively, could a ticket be less likely to be given a non-solution resolution if it is assigned to someone first?

Before providing a full explanation for how we went about investigating this question in phase 2 it is important to emphasize that an empirical study such as ours will not be able to prove cause

and affect relationships. However, we were able to establish statistical associations that may give decision makers insights or lead to further study.

3.5 Phase 2: Workflow Mining

The deeper insight we were able to gain in this second phase was inspired by the realization that our data is ripe with temporal information. Be leveraging this, we hoped to begin to address questions such as:

What came first? The assigning or the neglecting?

To establish statistical relationships based on precedence brought us to the conclusion that we would gain a lot of insight by leveraging the data mining technique: Sequential pattern mining. Sequential pattern mining is a subset of the powerful data mining technique called Association Analysis[7]. The general objective of this kind of data mining is to find associations, or "rules," for when the occurrence of two events are considered "interesting." The classic is example is:

Bathroom Tissue w/ Gorgonzola Cheese vs Diapers w/ Beer

In the mid-90s, it is rumored that a "major Midwest based retailer" mined their transaction data to find an unexpected relationship between the purchase of beer and diapers. "Unexpected" can be defined in many ways but quantitatively, if customers buy bathroom tissue X portion of the time and Gorgonzola Cheese Y portion of the time but there is not significant association between the two, then we would expect that

$$X * Y = Z \qquad \text{Equation 1}$$

Where Z is portion of the time customers buy both bathroom tissue and Gorgonzola Cheese because the two are independent. In market basket data, often burgers and buns are given as the opposite example. If a customer buys hamburger buns, we will assume they are much more likely to be buying burgers because the two items are associated with each other. In other words:

$$X * Y < Z \qquad \text{Equation 2}$$

The unexpected association that this "major Midwest based retailer" found was that, like buns and burgers, beer and diapers were associated together. This is often brought up in data mining

[7] Note the family of methods is called "Association Analysis" and not "Causation Analysis!"

circles because, had they not applied sequential pattern mining, no one might ever have associated the two.

As sequential pattern mining involves temporal information, we can take into account which event happened before the other as a means to consider whether the preceding event (antecedent) influences the probability of the second event (consequent). For example, what if an online retailer wanted to predict which customers might buy a toddler's sippy cup for the purposes of promoting such a product. Of course this could be tackled using a variety of predictors, but let's just say the group wanted to only use previous purchases as a predictor. Let's designate Y to be the resulting event in question and all potential predictors to be $X_{1...k.}$ We are then interested in studying the Sequential Rule:

$$\mathbf{X_{1...k} \rightarrow Y} \qquad \textbf{Equation 3}$$

One approach available is to note that Event Y occurs a% of the time in all cases but when X precedes it then Y happens some b% of the time and a happens to be less than b, then we claim that X may have a positive influence on Y.

$$\textbf{RelativeSupport(Y)} < \textbf{RelativeSupport(X->Y)} \quad \textbf{Equation 4}$$

So in our investigation of task distribution processes, we are essentially seeking items that appear unusually common together relative to how often we would expect them to (AKA high support) relative to other related patterns. In terms of workflow, this criterion might identify process steps that are often seen together. Utilizing sequential pattern mining might be the most efficient way to identify these patterns, especially when precedence *may* be a factor. In other words, we are especially interested in events that may be unexpectedly influencing another. This is most commonly measured by the "confidence" of a rule.

An example of confidence is the count of:

$$\mathbf{|X \text{-}> Y| \,/\, |X| = 40\%} \qquad \textbf{Equation 5}$$

This states out of all the times X occurred, Y has followed X 40% of the time. If you were to compare it to another relationship also involving the antecedent

$$\mathbf{|X \text{-}> Z| \,/\, |X| = 80\%} \qquad \textbf{Equation 6}$$

You could claim that when X occurred, Z followed twice as often as Y did.

3.5.1 Our Munging Process

In any pattern mining (or association analysis) effort, the data must be abstracted into a list of "items." In the case of sequential pattern mining, and in particular workflow mining, these events might be thought of as events. Each event captures information about what occurred in the process at that precise point in time. We referred to this process/action/effort as translation. It might sound straightforward, but in fact this was the area we devoted the most effort as there were so many ways to approach this task. In the case of our initial example on retail analysts, somewhere in the process of discovering the infamous beer and diapers relationship, the "Midwestern retail analysts" had to decide:

> . . were they looking for items purchased along with "Huggies Diapers Overnight Size 3", "All Huggies Diaper Products", "All Diaper Products," "All Baby Doody Management Products", or possibly even "All Baby Products (Non-Elmo Related)."

This is a decision often referred to as the choice of granularity (Jiawei Han, Kamber, & Pei, 2012) (Liu, Zhang, & Xiong, 2014). So during many of our translation attempts we were focused on determining the appropriate details to capture and, of course, much of our efforts were pointed at the events surrounding the assignment of a ticket. However, we were also interested in the conditions before and after an assignment event so we considered many additional abstractions.

3.6 Results

Our abstractions in many cases proved insightful and in a few cases did not but we reveal both groups so as to create a cohesive story of how we arrived at our final insights.

One important caveats: within our data set these is one project (Flex) that has anonymized user data for the ~8yrs that it was owned by Adobe, presumably to protect the privacy of its employees. Since the user as an actor is so important for this study, we just removed that project. Therefore, the set of projects used are sometimes referred to as the "non-Flex projects." As they are present in many of our results, we first present the "create state" results in Table 4.

3.6.1 Ticket State at Creation

We make several observations from the statistical association of the state of a ticket at the time of creation and our success criteria. Some indicators had a much stronger impact in terms of

percentage change on the proportion of tickets that were fixed rather than simply proportion that were closed. Meanwhile, in many scenarios the attribute in question at the create state of the ticket had a similar impact on closing the population of tickets versus fixing them.

The strongest example of a large delta between the impact on closing and the impact on fixing was the effect of an initially assigned release version. There is a 10.6% increase in the proportion of tickets that reach a closed status but a 22.3% increase in the proportion of tickets that are fixed. Similarly, 4.1% less of all the non-Flex tickets were closed without a release version assigned at create and 8.8% less tickets were fixed without such planning in place at creation. Interestingly, this effect is even stronger if the ticket type is a defect and almost as much so if a ticket is an improvement, but this effect is different for new features. As stated, the difference between an improvement and a new feature is not set in stone but the suggestion is that a new feature is a major undertaking to provide the user with a completely new option in how they use the product meanwhile an improvement might be as simple as a larger button. Defects are closed 13% more often and fixed 24.3% more if they have a release version assigned at the time of creation. Meanwhile, an assigned release version at creation makes very little (0.9%) improvement in a new feature being closed but an 8.1% improvement in it being fixed. But the proportion of new feature tickets without a release version at the time of creation were closed was 12% less than the overall population and fixed 18.5% less.

Defect	Improvement	New Feature	W/ Initial Assignment	W/o Initial Assignment	W/ Initial Version	W/o Initial Version	Closed Confidence Level	Closed Relative Support	Fixed Confidence Level	Fixed Relative Support
							80.96%	80.95%	64.55%	64.55%
X							81.91%	45.26%	63.60%	35.15%
	X						79.14%	18.19%	64.73%	14.88%
		X					74.24%	4.96%	57.63%	3.85%
				X			79.37%	50.72%	60.07%	38.39%
			X				83.91%	30.20%	72.60%	26.13%
						X	77.58%	55.69%	58.87%	42.26%
					X		89.52%	25.06%	78.92%	22.09%
X				X			82.29%	29.69%	60.83%	21.94%
X			X				81.42%	15.55%	69.01%	13.18%
	X			X			75.25%	10.98%	59.23%	8.64%
	X		X				85.97%	7.21%	74.34%	6.23%
		X		X			70.77%	3.25%	51.26%	2.36%
		X	X				81.97%	1.70%	71.78%	0.00%
X						X	78.55%	32.14%	57.74%	23.63%
X					X		**91.48%**	12.99%	80.23%	11.39%
	X					X	75.40%	11.85%	59.19%	9.30%
	X				X		87.24%	6.31%	76.67%	5.54%
		X				X	71.18%	3.37%	52.64%	2.49%
		X			X		81.70%	1.58%	69.79%	1.35%

Table 4 - The proportion of tickets that both closed (left results) and were fixed(right results), based on different create state attributes and combinations thereof. Defects tend to fair best and if a ticket starts with an assignee, it is far more likely to be fixed and closed.

Assigning a ticket to a user at the point of creation also increases the closure measures of Jira tickets at Apache by 5.7% greater closing proportion and a 20.9% greater fixing rate over not having an assigned user. Interestingly this was not true with defects, having almost not impact on closing and a 13.5% increase in the fixed proportion. In contrast, improvements and new features were 14.2% and 15.8% more likely to be closed respectively. Fixing percentages were greater at 25.5% and 40% more if a person was assigned to improvements and new features respectively.

Another observation we made from Table 4 is that nearly all the create-state attributes and their combinations had a much bigger impact on the fixed proportion rather than the closed proportion. The extreme case was having an initially assigned user for a new feature. This subset of the general ticket population had marginal better (1.3%) close rate but a tremendously better fix rate

with an 11.2% increase. The gap in impact between close and fixed rates is nearly eight times greater than the impact of close rate alone. Interestingly, there was a similar gap between the close and fix rate effects of having the planned release assigned to the ticket of a new feature at time of creation. So making this decision at the time of creation only slightly (1.5%) increases the rate of closing but the fix rate is improved by 8.1%. We speculate these two findings may be related to the workload involved in a new feature. Perhaps waiting for a volunteer is a much higher risk in this ticket type category.

The exception to this observation of fix rates being affected more than close rates was the case of defects with an assigned user at creation. This state marginally increased the close rate of defects by less than 2% but *decreased* the proportion of fixed defects by 5.8%. We speculate this is related to the proportion of defect suggestions that are closed with non-solution resolutions but it also led to many questions about how volunteerism is different for defects versus non-defect tickets and these questions will be addressed later.

Although we find these discrepancies between ticket type fascinating, it is important to note that some of these observations pertain to a rather small samples size. Over 400k of the 496k non-Flex tickets were eventually closed, meanwhile only 7850, or 1.58% of the non-Flex total, pertained to new features tickets that had a release version assigned to them at the time of creation. So as the literature tells us, we must a bit cautious that these findings are not spurious (or due to random variance.)

Of course some of the less impactful factors are worth noting simply because they are so wide spread. A ticket simply pertaining to a defect had a closing rate of 8.3% lower and for fixing, it was 10.7% less common.

In Table 4, we also note one of the few contradictions to this preference for redetermination. For defects only there seems to be a marginally (and possibly spurious) increase in close rate for tickets that do not have an initial assignment but a 13.5% increase (60.83% vs 69.01%) in the proportion of fixed tickets. So in the case of defects, an OSSD team might be better off waiting for an assignment rather than assigning it at creation but we will address that in greater depth in section 3.6.8.

This first example of our event abstraction was an efficient means to find some of the ticket attributes at the time of creation that were the most influential toward our success criteria. As we explain in our next section, precedence in this case was a very simple concept because the create state and the completion measures cannot occur at the same time. In fact, we could have acquired these results using queries and did not require sequential pattern mining. However, the remaining patterns discovered were not so simple.

3.6.2 Changes in Assignment After Creation

In the workflow of our tickets, there are three major forms of assignment change:

- an initial assignment when a ticket currently does not have an assignee at the moment
- a change in assignment during which one assignee is removed and another is added
- an assignment removal when a ticket goes from an assigned status to a non-assignment status

We wanted to know the influence of these assignment updates on our two success criteria and assembled a summation in Table 5.

	Closed		Fixed	
	Confidence	Rel. Support	Confidence	Rel. Support
<All>	80.96%	80.96%	64.55%	64.59%
New Assign	70.15%	25.65%	74.19%	27.13%
New Assign, Changed	55.10%	**1.67%**	63.43%	1.92%
Changed	61.08%	3.98%	66.42%	4.33%
Remove	**32.52%**	2.03%	**37.66%**	1.52%

Table 5 - Depicting the confidence and relative support of the three assignment patterns and one combination (initial assignment and a change) for our two success criteria of closure and fixing. We note that an initial (new) assignment historically appears to increase the fixing rate but lowers the closing rate.

Much of these historical statistics seem to align with those determined with simple queries such as Figure 10. If a ticket did not have an assignee when it was created but then acquires one, it is more likely to have a solution proposed and the initial assignment row follows that. Also, if that assignee is then changed, the probability that the ticket is fixed then decreases.

However, in the case of the closure rate, our initial pattern analysis appears to lower the percentage of tickets closed when an assignee is added. This is counter to intuition and the finding in the previous section that a ticket that begins its life with an assignee is over 4% more likely to be closed.

Elaborating on Figure 10, Table 6 depicts the portions of closed and fixed percentages taking into account whether a ticket has "ever" been assigned at any point in its history versus not. These also appear to contradict the findings in Table 5 and we set out to understand why.

		Closed	Fixed
		Confidence	Confidence
All Tickets		80.96%	64.55%
Never Assigned		60.50%	**32.17%**
Assigned Eventually		88.70%	77.04%

Table 6 - We readdress the history of assigned vs not assigned tickets as with Figure 10 but this time with pattern mining. We note that the results seem to contradict each other and we set out to determine why in the next section.

3.6.3 The Not Events

According to (Han, Kamber, & Pei, 2012a; Tan et al., 2005), the confidence measure is based on the probability that B will occur, given that A has. Because we are focusing our efforts on sequential pattern mining and not a form of association analysis that is void of temporal data, A would need to occur in an item set prior to B.

$$\textbf{confidence}(A \Rightarrow B) = P(B|A) \qquad \textbf{Equation 7}$$

This is often translated using the count measure, sometimes depicted as:

$$|A| = \textbf{the number of tickets that involve the event A in any order/ any instance} \qquad \textbf{Equation 8}$$

Therefore we calculate equation 7 with:

$$\textbf{confidence}(A \Rightarrow B) = |A \cup B| / |A| \qquad \textbf{Equation 9}$$

where the numerator asserts that A precedes B. In order to completely appreciate this measure, we must note that all of the instances of A include instances of

1. B following A.
2. no Bs following an A
3. A *following* B without another A preceding B (AKA reverse scenario)

If A is our assignment event and B is our success measure, then scenario 1 is the numerator and all three scenarios in sum are our denominator for the confidence measure. Meanwhile, the sum of scenario 1 and 3 make up all the tickets where both an assignment and a close status occurred. If that is used as the numerator and all instances of assignment are the denominator, then the ratio in the left portion of the mosaic plot of Figure 10 is explained with these scenarios as well. Therefore, our mosaic and confidence measures are in fact measuring different relationships between the action of assigning a ticket and our success measures.

Another interpretation of our confidence measure asserts that of the tickets where an initial assignment occurs, a fix event will follow it 74.19% of the time. This may be considered insightful, but we argue that there are other approaches that bring us more insight into how important it is to assign tickets to users. So our altered research question is:

> **RQ4: "Does a ticket assignment after the creation of a ticket reduce the probability of the ticket not being completed or fixed?"**

This is an example of measuring the pattern involving the ***lack*** of an event. So for every one of our ~496k non-Flex translated ticket histories, if it lacked a fix event or a closure event, it received a new event indicating so at the end of its transaction list. We suggest this alternative translation may bring more useful insights as depicted in Table 7. As a simple comparison, we depict in the first row our "base case" of all tickets in the data set that were not from the Flex project.

Antecedenat Event	Closed		Not Closed		Fixed		Not Fixed	
	Confidence	Support	Confidence	Support	Confidence	Support	Confidence	Support
<All>	80.96%	80.96%	19.00%	19.00%	64.55%	64.55%	35.41%	35.41%
New Assign	70.15%	70.15%	6.39%	2.34%	74.19%	27.13%	18.52%	6.77%
New Assign, Changed	55.10%	**1.67%**	6.68%	0.20%	63.43%	1.92%	17.73%	0.54%
New Assign, Removed	0.47168	**1.14%**	20.94%	0.50%	0.51494	1.24%	37.64%	0.91%
Changed	61.08%	3.98%	9.31%	0.61%	66.42%	4.33%	20.35%	1.33%
Removed	**32.52%**	2.03%	40.50%	2.52%	**37.66%**	1.52%	67.97%	4.24%

Table 7 - Our first example of the not-events shows how the lack of success may be a better reflection of outcomes as the antecedent events cannot occur after said indicator. We also note the disadvantage of smaller sample sizes in the case of this study.

This first demonstration/evaluation helps show how effective these new measures are to investigate the effects of assignment changes. The original observation that an initial assignment dropped the closure rate to 70% is misleading due to the occasional instance of an assignment *after* the finish event but the new measure of not-closing shows a reduction in the undesirable occurrence by nearly 2/3rds and the not-fix rate fell by almost half. According to the confidence measures of this new approach, even changing the assignment is beneficial and it's easy to speculate that is because someone is still assigned to the ticket.

Although the original approach could have implied it was even more impactful, removing an assignee such that there is a period of no assignment doubled the not-close rate and nearly doubled the not-fix rate.

Clearly removing an assignee is an indicator that a ticket is much less likely to be completed or fixed but let's remember the selection bias in effect here. That someone changed their mind in committing to resolve a ticket might be an example of poor process and making process suggestions to the OSS community is our primary goal. But it might also be related to the importance of the requirement. Perhaps the need was not properly evaluated when the initial

assignment was made (likely volunteered for). Perhaps the defect had been resolved or the need met in another way after the time the user volunteered.

In other words, an assignee being removed from the ticket might often be due to the value in the request depicted in the ticket changing in the eyes of the person that the ticket was assigned to. Therefore this is not necessarily a process choice that should always be avoided. Most importantly, this form of selection bias may apply to many of our findings which is why we assert the goal is to find statistical associations and not causes.

As another demonstration of the use of association rules, we also depict {init, changed} vs {init, removed}. According to the definition of an association rule, the only precedence being asserted in:

{new assign, removed} → {Closed} **Equation 10**

Is that "initial assignment" and "assignment changed" happened before "Closed." So in reality, an assignment might be removed and *then* an initial assignment could have occurred before the close thus the ticket had an assignment when closed. This is probably why, in the case of our not-closed event, the association rule has a similar result to the overall population.

Similarly, a rule such as this is asserting what *did* happen before the success measure and how often. Other rules need to be used to learn how often other events happened *in addition.* In other words, in some of the cases of the rule depicted in equation 10, there were many instances of a change (or other assignment) event also occurring before the close event.

3.6.4 Assignment Updates with or without Initial Assignments

When evaluating the differences in influence over tickets that started with an assignment vs not starting with an assignment, only the *removed* event seem to differ significantly as depicted in Table 8.

Apparently, removing an assignee from a ticket that did *not* start with an assignment is not nearly as bad/detrimental as one that did start with an assignment even though some one would have had to be added before the remove event. In other words, perhaps a ticket that started its life with an assignee is experiencing a greater risk by having an assignee removed than a ticket that

had removed a participant who was assigned later in the tickets history. We notice this is especially true with the measure against not-fixing. The general population has a not-fix rate of ~35% but a ticket that starts with an assignment has only a 27% chance of *not* being fixed. Yet if a ticket in this same pre-populated list experiences a remove at any time, our historical data shows that the not-fixed outcome happens 80% of the time or nearly 3 times as often. This may be very significant because this might indicate that some one that is assigned in the beginning might not be as committed. Meanwhile a temporary keeper of the ticket has less of a negative effect on the prospect of future efforts by others. Is this an argument against automatic assignment at the time a ticket is created? Should we discourage automatic assignment?

It is worth noting that a removal after a pre-populated assignment in the not-closed or not-fixed population is roughly four times as likely as with tickets that do not start with an assignment. However, this is a small sample size relative to many of the observations we make in this study and should be taken with a grain of salt.

W/ Initial Assignment	W/o Initial Assignment		Closed		Not Closed		Fixed		Not Fixed	
			Confidence	Support	Confidence	Support	Confidence	Support	Confidence	Support
		<All>	80.96%	80.96%	19.00%	19.00%	64.55%	64.55%	35.41%	0.00%
	X	<ALL>	79.37%	50.72%	20.57%	13.15%	60.07%	38.39%	39.87%	25.48%
	X	Removed	43.65%	**0.85%**	22.99%	0.45%	43.89%	0.86%	40.47%	0.79%
X		<ALL>	83.91%	30.20%	16.09%	5.79%	72.60%	26.13%	27.40%	9.86%
X		Removed	**27.55%**	1.16%	**48.05%**	2.02%	**12.98%**	0.55%	**80.58%**	3.39%

Table 8 – We compare the impact of a remove event against tickets that start their existence with an assignee, without an assignee and the general population of tickets

3.6.5 Relative Assigns

We speculated that workflow mining doesn't just have to be about *what* was done but *by* whom. Thereby, another investigation into the effectiveness of this data mining technique was what we deem as: "relative roles."

Aside from the current team lead, there is very little formal indication of roles, let alone titles, in the Apache Jira system. Perhaps in future efforts, supervised learning might match some of the

attributes of OSS contributors to attributes of team members of commercial teams where participants have titles. In this manner we might be able to keep track of the actions taken by participants in terms of whether they are filling the role of those that focus on requirements, development, quality, or management.

As an alternative to this larger picture, we looked at the roles played by participants in one particular ticket. Effectively, we wanted to investigate whether an assignment action was taken by the same user that created, attached to, resolved, or closed the ticket. We also abstracted the user "other" if the assignment action was taken by a user that filled neither of these roles. This event was particularly of interest as we had hoped it would give insight into a user that might not be doing the specific work of this ticket but was a part of leadership in the project as a whole. We hoped it would give insight into management.

However, there turned out to be notable flaws with this approach that may brought lessons learned to future abstraction approaches. The first is simply that if a ticket is not closed then it does not have a closer. Therefore, of the transactions with a closer assignment, there were no "not-closed" events. Similarly, there might be a resolver in a ticket that was not fixed but only if it was given a non-solution (non-fix)resolution so this brings little insight. With these caveats aside, we did gain a bit of insight with this assigned-by-XX approach overall.

The first observation from Table 9 was that a ticket that was assigned-by-creator has a much better chance at being completed than the general ticket population. This scenario reduces not-fix and not-close rates by almost half. Of course, in this case we are not including the assignments made at creation as this is implied to be done by the creator. We find this influence on the outcomes interesting because all the assignment types are represented: new-assignment, change-assignment, and remove-assignment. However, when we take into account that new-assignments are 74% of the assignment events for non-Flex projects (after creation), the presumed impact on removed-by-creators is understandably not as great.

EVENT	Not Closed		Not Fixed	
	Confidence	Support	Confidence	Support
<all>	19.00%	19.00%	35.41%	35.41%
Assigned by Creator	10.12%	1.20%	18.76%	2.23%
Assigned by Attacher	5.69%	0.45%	10.97%	0.87%
Assigned by Resolver	0.00%	0.00%	12.93%	3.97%
Assigned by Closer	0.00%	0.00%	17.70%	2.57%
Assigned by Other User	**34.87%**	3.40%	52.68%	5.14%

Table 9 - Based on the roles the assigning user played in a given ticket, we observe the creator has a very positive influence historically, as does the more obvious user that attached a file. The attachment was likely a proposed solution.

Assigned-by-attacher is also an intuitive indicator that the a ticket is headed toward success as we find 70% of Apache attachments have a file type that is *very likely* a proposed change, as opposed an attachment that might pertain to additional information about the requirements such as a screen shot. Some examples of these very code oriented file types are Java, diff, and patch. If we also assume all txt and zip files are proposed solutions, although in reality some of them may be requirements oriented, the ratio of solution related attachments climbs to 87%.

The anti-pattern rates of assigned-by-other did appear, at first glance, to indicate that outside forces should be discouraged from interfering with ticket workflow. A great call for developer freedom was about to ensue when we realized there is a fatal flaw with our abstraction logic. In reality, for a ticket that was not resolved and closed, we have speculated that *in many cases* an "other" user is more accurately described as "intended to close" or "intended to resolve" user but just hadn't gotten to it yet. We can support this suspicion because of the tickets that had a closed event, 84% also had an assignment event by the same person who did the closing. This led to the overwhelming suspicion that for a very large portion of the time users are assigning tickets to *themselves* with the intention of closing or fixing a ticket. We suspected this other 16% of tickets that had assignments and were closed (at some point) might be operating, at least partially, under the long arm of management. As contrasting the impact of both of these circumstances may have a profound effect of ticket workflow, we address this in section 3.6.8.

3.6.6 Relative Assigns by ticket type

Despite these disadvantages to our relative assignment approach, we were able to identify additional circumstances that gave use greater insight into assignment processes. In Table 10 we note that an assignment change that is made by the creator is more positively impactful if it pertains to a defect that other ticket types.

Defect	Improvement	New Feature	Assign Event	Not Closed		Not Fixed	
				Confidence	Support	Confidence	Support
		X	<All>	25.72%	1.72%	42.33%	2.83%
		X	Assigned by Creator	**15.76%**	0.15%	**26.15%**	0.25%
		X	Assigned by Other User	35.97%	0.22%	51.38%	0.32%
	X		<All>	20.84%	4.79%	35.27%	8.11%
	X		Assigned by Creator	**10.49%**	0.32%	**19.05%**	0.58%
	X		Assigned by Other User	33.81%	0.67%	52.82%	1.04%
X			<All>	18.03%	9.96%	36.34%	20.08%
X			Assigned by Creator	**8.50%**	0.46%	**16.87%**	0.91%
X			Assigned by Other User	36.93%	2.22%	55.85%	3.35%

Table 10 - Building off of our work of relative roles, we note if the user that creates the ticket is the same participant that makes the assignment change, there is a positive influence on outcomes and this is especially true for defects.

3.6.7 Relative Assigns with Create State

In Table 11 we also evaluated how these assignment changes differ from our two large categories of tickets: those that started with an assignment and those that did not. To highlight the difference in the two groups, we also show all the init (new assignments) and change assignments by all participants. It would seem that although new-assignments and change assignments in general reduce not-fixing rates and especially not-closing rates of the *pre-assigned* group as seen in Table 11, the overall assign-by-creator actions produce very similar results to the general ticket population. Perhaps changes by the creator once a ticket is assigned at creation tend to be neutral but if it's the first assignment a ticket has had, the creator either does well by the ticket by solving the problem themselves or knows who is ideal to do so. It is important to note these specific groups are a very small portion of our ticket population, many in Table 11 are well under 1%, so these results are more sensitive to random chance and should be taken with a grain of salt.

The other insights we gained from Table 11 are that the assigned-by-other group of users is devastatingly harmful, at least according to our historical data, toward the closing and fixing of tickets if the ticket started its life with an assignment. Could this be partially due to a large portion of remove-assignments? As less than 13% of all assignment actions are removes, this is unlikely the primary contributor of this finding. In the case of tickets with the prepopulated assignee but a lack of success, we know that a change was made to that original assignment and then the positive outcome was not achieved. So are these "other people" assigning the work to unwilling participants or were they volunteering to do the work but then never did it. We will now dive into our most fruitful abstraction to ascertain.

W/ Initial Assignment	W/o Initial Assignment	EVENT	Confidence	Support	Confidence	Support
	X	<all>	20.57%	13.15%	39.87%	25.48%
	X	New Assignment	6.28%	2.27%	18.40%	6.64%
	X	Assigned by Creator	**7.39%**	0.74%	15.50%	1.55%
	X	Assigned by Attacher	5.27%	0.38%	10.43%	0.75%
	X	Assigned by Other	**24.03%**	1.46%	**39.11%**	2.38%
X		<all>	**16.09%**	5.79%	**27.40%**	9.86%
X		New Assignment	11.80%	0.05%	25.18%	0.11%
X		Assignment Changed	11.60%	0.41%	22.58%	0.80%
X		Assigned by Creator	**24.64%**	0.46%	36.16%	0.67%
X		Assigned by Attacher	9.01%	0.07%	15.36%	0.12%
X		Assigned by Other	52.19%	1.88%	74.90%	2.69%

Table 11 – We review our relative assign events relative to whether the ticket started with an assignment to find that, compared to new-assignments and even change-assignments, an assignment event by the creator is not terribly helpful if the ticket started out with an assignee. Also, assign-by-other events were devastating to tickets that started out with an assignee.

3.6.8 Volunteerism in Open Source Software

As mentioned in the previous section, we believed that the more insightful patterns may lie in not who the assignment change is made *by* but *to* whom the ticket is now assigned. Also as we

suspected, these are often the same person as 33.6% of tickets involve an assignment update where a user is assigning a ticket to themselves.

In the first three rows of Table 12 we observe that tickets with both assign-to-self and assign-to-other fair much better than the general population which include tickets with no assignment. This supports previous findings. More importantly, a margin of just over 3% seems to indicate that both non-completion anti-patterns are less common in tickets with assign-to-self over assign-to-other. This margin is similar throughout most of our other categories of ticket type except for new features where assign-to-other seems to be particularly less effective than assign-to-self. Could new features require more work and thereby require greater self-driven commitment?

We also observe that the margin of impact between assigning to self vs other is not nearly as impactful with tickets that had an assignment at creation. Could the effectiveness of volunteerism be hindered by the practice of pre-populating the assignment simply because (in most cases) someone has to remove themselves from the assignment in order to allow another participant to volunteer? In general, starting a ticket with an assignment has been shown to be associated with better outcomes but if someone volunteers after that fact, perhaps there is a lessened sense of ownership. We also observe there were significant differences on the assign-to categories based on ticket type. Defects seem to be especially impacted by assign-to-self, reducing the not-closed anti-pattern by nearly 75%. Also, tickets that are put in the system without an initial assignee also seem to benefit especially from volunteerism having brought the not-closed group down by about 70% meanwhile those that started with an assignee were only improved by 41% in close rate.

It should be noted, as before, that some of these observations are made with a rather small sample sizes. This is especially true because we have decided to focus on the lack of a positive outcome which, fortunately, is a much smaller samples set. To put any concerns of spurious results to rest, we conducted a 1 sided proportion test on the assign-to-self and the assign-to-other events[8] for all tickets and then the defects only. The ~3.5% delta (relative to all tickets with this assignment event) proved to give p-values of 8.8×10^{-14} and 4.8×10^{-8} respectively and we

[8] Assign-to-other leading to a not-closed scenario had only a 4250 sample size. Assign-to-other for defects leading to a not-closed scenario only occurred in 2024 tickets

take this into account with all sample sizes of this nature and deltas of this approximate size for previous results as well. In nearly all cases presented, our sample size is sufficient to justify declaring that these difference in confidence measures as significant.

Defect	Improvement	New Feature	W/ Initial Assignment	W/o Initial Assignment	EVENT	Not Closed		Not Fixed	
						Confidence	Support	Confidence	Support
					<ALL>	19.00%	19.00%	35.41%	35.41%
					Assign to Self	6.19%	2.08%	18.14%	6.10%
					Assign to Other	9.66%	0.86%	21.33%	1.89%
X					Assign to Self	4.78%	0.91%	17.95%	3.41%
X					Assign to Other	8.12%	0.41%	20.91%	1.05%
	X				Assign to Self	7.34%	0.56%	17.15%	1.31%
	X				Assign to Other	10.45%	0.20%	20.77%	0.40%
		X			Assign to Self	10.65%	0.24%	23.12%	0.51%
		X			Assign to Other	15.94%	0.10%	28.89%	0.18%
				X	<ALL>	20.57%	13.15%	**39.87%**	25.48%
				X	Assign to Self	5.72%	1.77%	17.82%	5.51%
				X	Assign to Other	8.77%	0.62%	20.43%	1.44%
			X		<ALL>	16.09%	5.79%	27.40%	9.86%
			X		Assign to Self	11.22%	0.30%	21.46%	0.58%
			X		Assign to Other	12.91%	0.24%	24.67%	0.45%

Table 12 – We contrast volunteered (assign to self) tickets versus assigned to other to show the historically positive association with volunteerism, especially with regard to defects.

3.7 The Inadvertent Insights from Sessions

An early consideration to abstracting the events was whether asserting the specific order of every event was relevant. If a user bothers to change the priority of a ticket just before assigning it to someone, is that specific subsequence important? Borrowing from the research on web sessions, perhaps all actions taken during that specific users session on Jira can be grouped the way market researchers group all items a customer buys on the same trip to the store together. During one specific visit, is it relevant whether the diapers or the beer were placed in the cart first? Probably not. But that more beer and no diapers will be placed in a different cart a couple

months from now might be. So in many instances, order is quite relevant but not down to the minute.

We speculated that a similar reduction in temporal granularity might increase the confidence we have in our findings. In reality, many of our findings were quite different under this approach and after some investigation the cause became obvious.

In literature, a sample transaction without sessions or itemsets might be depicted as such:

A, B, C, B, D, E, F **Equation 11**

And given the minimum requirements of the interestingness rules there might be enough examples within the sequence database/set, such that we might discover rules such as:

D → F **Equation 12**

In contrast, if you were to group events in sessions, there might be transactions such as these:

{A, B}, {C, B, D}, {E, F} **Equation 13**

{A, B}, {C, B}, {D, E, F} **Equation 14**

Note, both have the same events in virtually the same sequence and the transaction in equation 13 would support the equation 12 rule because D does occur before F. However, the transactions in the vein of equation 14 will not simply because D is occurring, as we refer to it as, "effectively simultaneously" as F, as opposed to preceding it. Therefore, it does not support the rule in equation 12.

So in the case of our study, if we were especially interested in the impact of event D on the success measure F, the session groupings could be argued as misleading. However, if we treat the anti-patterns of not-closing and not-fixing as always in their own pseudo-session at the end of every transaction, this concern is mitigated.

With this essential lesson learned established the original success measures of closing and fixing were greatly affected by this grouping if they were considered the consequent of events quite simply because they often occurred in the same session as the antecedent in question. In other words, if D and F are the subject of study and they can be in the same session, then using sessions led to misleading confidence measures. However, the stark contrast in some of these results highlighted in Table 13 indicate how often some of our antecedent events of choice

occurred at effectively the same time as the target consequent and this lead to some intriguing research questions. Although a full investigation of the impact of sessions on the many insights gained is beyond the scope of this paper, the previously mentioned area of self-assignment was particularly insightful. We note this is especially true with new defects and follow up with that observation in section 3.8.1.

Defect	Improvement	New Feature		Closed		Fixed		Closed w Sessions		Fixed w sessions	
				Confidence	Support	Confidence	Support	Confidence	Support	Confidence	Support
			<ALL>	80.96%	80.95%	64.55%	64.55%	72.69%	70.19%	57.33%	55.36%
			Assign to Self	69.86%	23.51%	74.81%	25.17%	47.01%	15.28%	45.42%	14.76%
X			Assign to Self	69.67%	13.24%	74.68%	14.20%	45.07%	8.27%	56.36%	30.08%
	X		Assign to Self	68.65%	5.22%	75.35%	5.73%	46.66%	3.43%	56.91%	12.63%
		X	Assign to Self	69.29%	1.53%	70.78%	1.56%	51.13%	1.09%	52.23%	3.37%

Table 13 – We can see a significantly different confidence measure when we group events into itemsets based on user sessions for both the close rates and fix rates of different ticket types. We note this is especially true with new defects.

3.8 Delays

The realization that so many tickets (approx. 10% ~ overall percentage) were fixed in the same session that someone volunteered to own them (assigned to self) led to the research question:

> **RQ5: Do the tickets that are not closed or not fixed in the same session as the session in which a user assigns the ticket to themselves exist in a incomplete state longer?**

When a ticket is closed during the same session that it was volunteered for, we deem this to be "claim and close" as depicted in Figure 11. We dub this habit of volunteering for a ticket but holding the ticket for some time before completing the work as "claim and hold" as Figure 12 depicts.

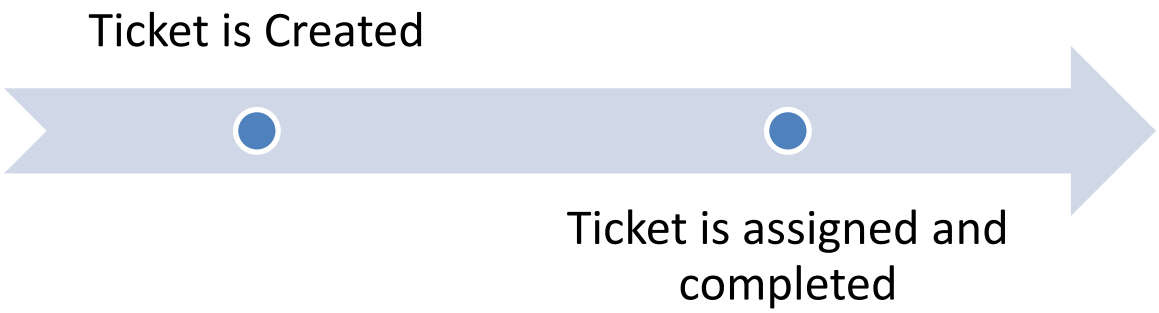


Figure 11 - A timeline showing the point in time a ticket is both assigned and completed sometime after the point it was created.

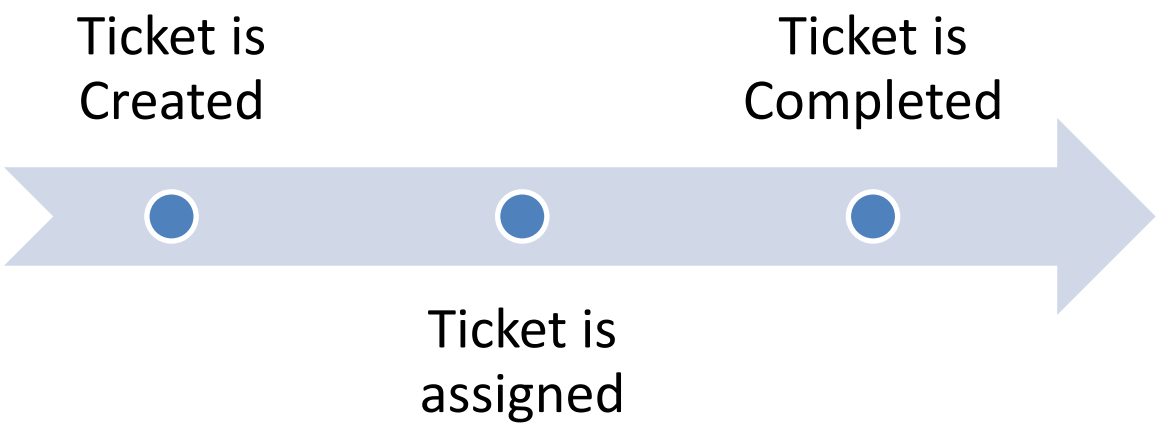


Figure 12 - A timeline depicts that the time of completion is past the time a ticket is assigned and both are after the time of creation.

We tabulated this completion or fixing time (T_{cp} or T_{fx}) by taking the time stamp from the completion event (t_{cp}) or fixing event (t_{fx}) and subtracted the creation time stamp (t_{cr}).

$$T_{cp} = t_{cp} - t_{cr} \quad \text{or} \quad T_{fx} = t_{fx} - t_{cr}$$

By comparing the “claim and hold” with the “claim and close” groups by this criteria, we discovered that there is at least one down side to the typical open source form of volunteerism. In Figure 13 - From the point of creation to the time of closure, a ticket lives in the system nearly five times as long if the user self-assigns but holds the ticket instead of completing the work during that same session, we observe that the “self-assign with hold” group has a mean “create to

close" time of 490% longer than "self-assign w/o hold." More striking, the median is over fifteen times longer.

Tickets that are never assigned to anyone, on average, take 65% longer to close, than a ticket that was self-assigned without a hold but they still take 66% less time than a self-assigned ticket that was held past the initial moment of volunteering (AKA "with a hold").

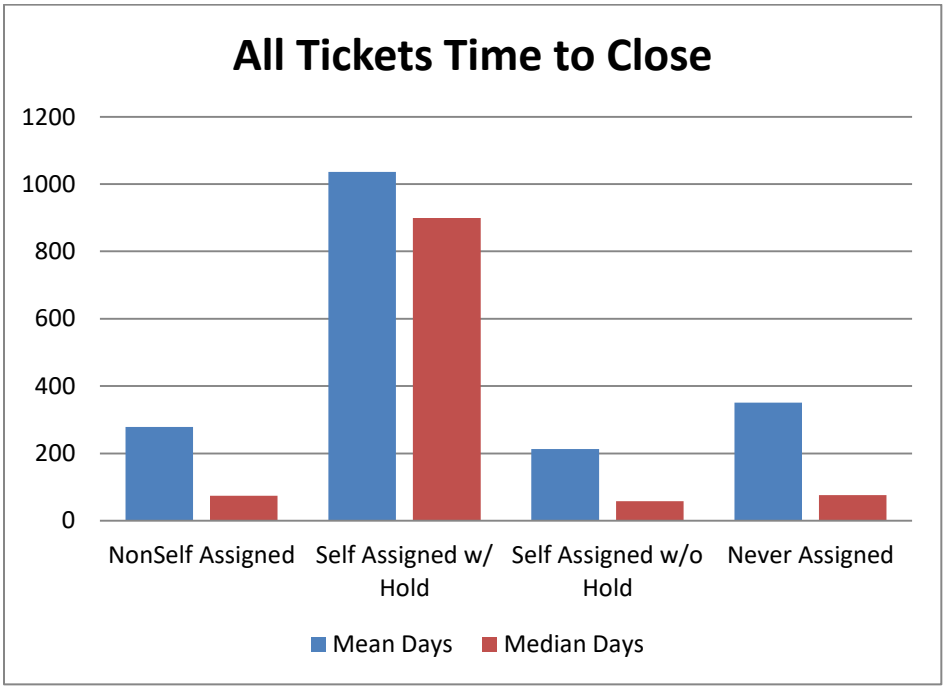


Figure 13 - From the point of creation to the time of closure, a ticket lives in the system nearly five times as long if the user self-assigns but holds the ticket instead of completing the work during that same session

When we look at the difference in cycle time from a standpoint of creation to fixed status, so as to exclude the non-fix tickets, the difference is even more astounding. In Figure 14, we show that the median time to fix for the "without hold" group is 95% less than the hold group. For tickets that are never assigned, the median time to fix is 92% less than the hold group. We also notice the contrast of mean to median measures indicate the distribution of time to fix is heavily skewed to the positive. While the time to closed data for "self-assign and hold" had a mean time that 364% longer than the median time, the time to fixed mean time for the same process pattern is 1768% longer. Clearly there is a very long tail contributing to this vast difference between the self-assign with a hold versus any other group. This lead to discussions about whether a certain

ticket category might be more inclined to attract a volunteer yet also is inclined to experience a delay even after this volunteering. We first investigated defects vs non-defects.

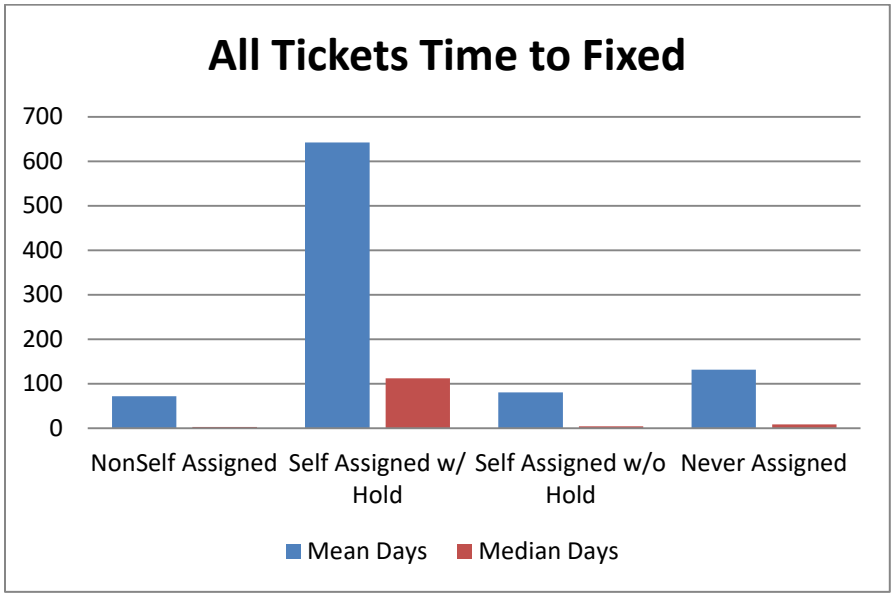


Figure 14 - In terms of tickets that are actually fixed, from the point of creation to the time of fixing, a ticket lives in the system nearly eight times as long if the user self-assigns but holds the ticket instead of completing the work during that same session

3.8.1 Delays Considering Ticket Type

In Figure 15, we observe that in categories of non-hold, both self-assign and nonSelf-assign, the non-defects seem to take just a bit longer to achieve a fixed status, however in the case of "self-assigned with a hold" the defects stay in the system much longer than the non-defects. Could this be because some are not as important as others? We know from our first effort that roughly half of our tickets are defects. Could enough of them be low priority enough to create the long tail in delays?

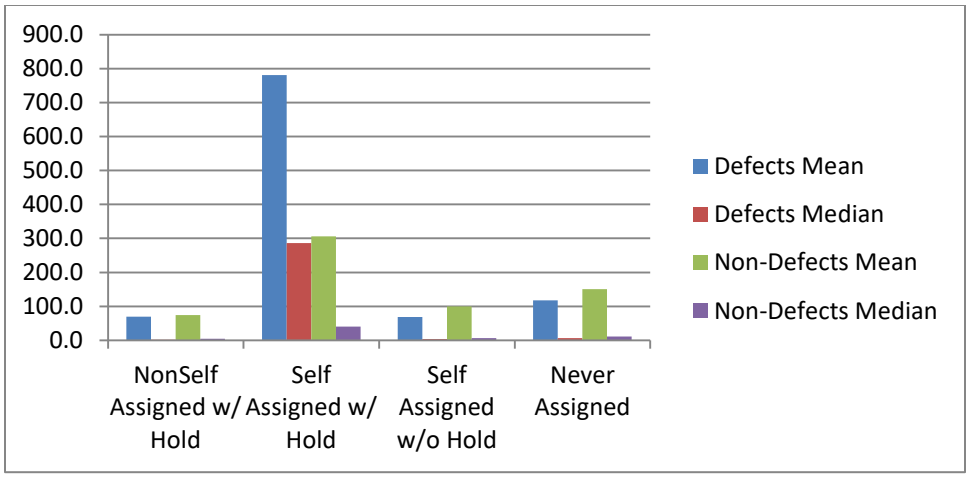


Figure 15 - Comparing Time-to-Fix for Defects vs Non-defects we see that there is a much greater difference with regard to defects

When looking at the five priorities made available in Apache's instance of Jira (Trivial, Minor, Major, Critical and Blocker) in Figure 16, there was a notable impact on cycle-time. With the exception of Trivial, which had the lowest median time to fix and a mean time to fix roughly equivalent to the 2nd more important priority, the priority categories showed a distinct difference in response time based on their proclaimed importance. However, this difference was still dwarfed by our discovery that a volunteer that holds a ticket after self-assigning it.

Percentage wise, the minor and "critical"[9] showed the greatest difference in ratio to the without hold categories (1091% and 1396%). Perhaps many minor tickets are just not terribly important as the non-hold categories show and this is just accentuated when volunteerism is involved. Regarding the critical tickets, we speculate the extreme mean measure is partially due to the great variety of tickets that fall under this category. A full investigation of the use of priority designations in OSSD teams is beyond the scope of this article, but we speculate that some tickets are marked as critical however only a few people are qualified to resolve or fulfill the ticket properly. So, although many are critical tickets are probably resolved in a timelier manner due to their importance, some have to wait for the right person to become available. If a person knows they are one of the few qualified, they may volunteer but simply not have the means to get to it for

[9] Critical is lower than "Blocker" so Critical is a P1 in a P0-P5 system.

some time. This scenario we speculate on could be identified in specific case studies and may be the source of great delay in the OSSD community.

Of course, these critical bugs that are not resolved immediately after being self-assigned represent 0.1% of our bugs and the critical bugs that were self-assigned but resolved immediately were five times as common but still a very small portion of our dataset. So we acknowledge there is some element of randomness in these figures.

One of the possible reasons the “major” ticket are not differing from the non-hold or non-assigned tickets to the same degree as the critical and minor tickets might simply be that there are a lot more of them. As 58.5% of our tickets are self-assign w hold tickets are Major priorities and 27.2% are minor, there might just be lower relative variance and a lower percentage of extreme outliers in the major group to keep the mean delay more reasonable. However, we still find this to be an insight that the OSSD community should consider. Throughput might be greatly improved if the community were to mitigate the historical trend of this group taking 609% longer than those not holding or 391% longer than tickets that are never assigned.

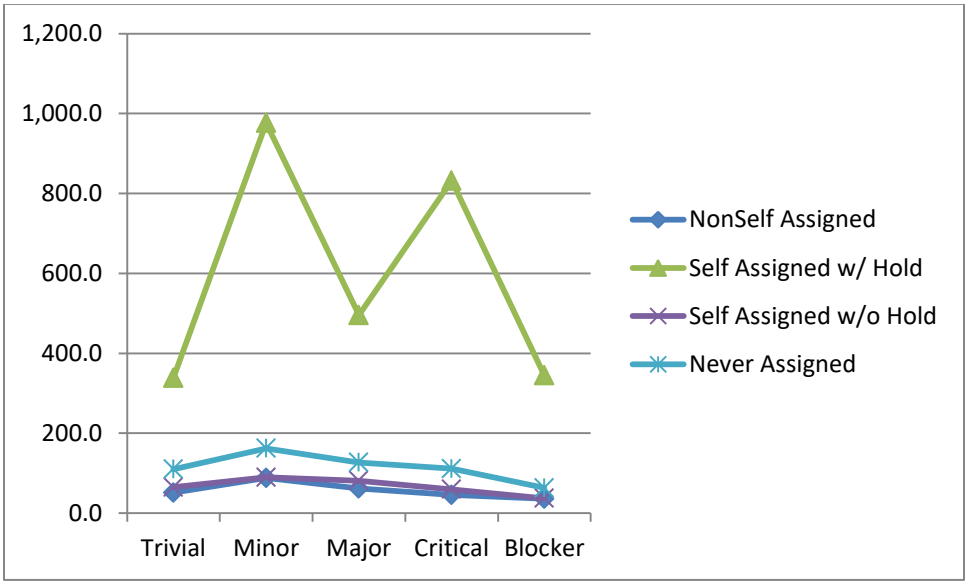


Figure 16 – Contrasting our four process choices against the five priority designations, we can visibly see that in terms of the mean fix times for all tickets, the “self-assignment with hold” represents the greatest delays.

However, even though we were able to show that priority was a small factor in the delay time, we do acknowledge that does not remove the possibility that many of the “claim and fix” are the “easier” bugs. Despite guidance otherwise, one ticket representing a defect, improvements or new features might actually require many days’ worth of work. Therefore, resolving the ticket in the same session is physically and mathematically impossible unless said ticket is broken up into the recommended 2-8 hour chucks.

3.9 Chapter Two Conclusions and Suggestion to the OSSD Community

We presented our two measures for success in the Open Source Software community and investigated their relationship to the concept of assigning work tickets to the participants in this community. We then showed how workflow mining was an efficient means to identify process patterns that tend to be associated with tickets that are completed by our two success indicators. We acknowledged that some of our early work could have been done with elaborate queries such as our work in the state of the ticket at the time of creation where, with rare exception, the sequence was consistent throughout the dataset. It would have to be a very unusual circumstance for the type of ticket (E.G. defect), for instance, to be determined after the ticket is closed. So sequential pattern mining was simply convenient but not required for establishing statistics related to these factors. However, most of the remaining investigations that we undertook needed an efficient means to measure how often a process pattern preceded a positive outcome (or lack of a negative outcome) and sequential pattern mining turned out to be an excellent means to identify and measure these process patterns. However, we acknowledge that, in many cases, the contents of the ticket quite likely influenced the type of processes a ticket underwent. When evaluating the factors that led to the success of a ticket, we must consider that the value of the work proposed was also a significant factor. In any study it would be difficult to measure all factors at once so we chose to focus on finding relationships between process choices and outcomes. We argue that this method allowed us to gain valuable insights and we ask the reader to consider our hypothesis that the processes teams and leaders choose will often have an impact on the likelihood of success.

As we considered numerous approaches to abstracting our data, we were able to report many process related influences and we summarize the most insightful as the following. We discovered that in our process logs, the person that created the ticket has the best record for determining who should own the ticket. In addition, tickets that were assigned to the same person that did the assigning (an indication of volunteerism) also had a higher ratio of completed work than the tickets assigned to someone by someone else. However, well over half of the time a participant volunteers for work, they do not get to complete the work until another visit to the requirements management tool Jira. We refer to this as “claim and hold” and we explained that sometimes this delay represents an exceptionally long period of time. We presented many scenarios, but compared to volunteers who present their solution almost immediately upon claiming the ticket as their own, those that hold the ticket, thus preventing others from volunteering for the work, tend to create a scenario where the ticket exists incomplete for 5 to 10 times as long on average. We investigated the influence of both ticket type (defects and otherwise) and also priority to see that, although these categories influence a tickets life span, this habit of “claim and hold” is still the strongest source of significant delay for the tickets in the volunteerism category; a group of tickets that is normally prone to high completion rates. Therefore, we make the following humble suggestions to the Open Source Software Community. We believe that throughput would increase if there were new means to make the time period of holding a ticket a bit more visible. This could be in terms of reports that summarize all the open tickets in a project and it could also be emphasized with flags on the tickets themselves. So as not to prevent others from volunteering themselves, we suggest there could also be an alternative system where participants express interest in a ticket without claiming ownership until the time approaches where they have time to begin work on it. We do acknowledge that ownership often needs to occur for longer than a Jira session as the work involved requires hours if not days. However, before work has begun, couldn’t the ticket still be up for grabs in many cases? This is clearly a cultural question but it is one we propose every team should address. Alternatively, we propose there are some cases where it is acceptable, probably after a respectable warning time, that a ticket is given back to the unclaimed pool after it has been owned by one participant for too

long. How long that period might be would probably depend on the team and is another choice that should be made collectively.

We encourage the open source community to consider these process improvements but it is also important to realize that it would be very counterproductive to discourage volunteers through public shaming. All four authors of this paper have participated in numerous volunteer projects outside of software and, anecdotally, we see a common theme. Volunteers might be willing to give their time for a mission that they strongly believe in. However, in most cases they would like to do so on their own time in their own way. If holding tickets is the only way for some OSSD participants to feel a sense of ownership then it is possible that these delays may be a necessary cost of open source software development.

4 WORKFLOW VARIATION WITHIN THE APACHE SOFTWARE FOUNDATION VIA PROCESS MINING

4.1 Chapter Abstract

Requirements repositories manage the state of software requirements as they move from an idea to actually becoming an improvement or fix in a software project. As such, the data pertaining to each event that occurs to a work ticket (ticket) may bring great research value when applying the methods of process mining. We describe our system designed to capture data from any web based requirements repository, verify that data, and then analyze such data. While making suggestions for numerous additional research projects, we demonstrate this research potential by conducting a case study of the many different projects within the Apache Software Foundation (Apache). We show the variety in ticket count, age, and growth rate. We then compare the different workflows within Apache by process mining each project's tickets for various measures of effectiveness (especially with regard to completion) and various indicators for process choices. In order to study literally tens of thousands of patterns among all the teams with enough data to indicate patterns (~85), we also describe a data mining platform we developed that specializes in pattern mining at scale. In this study, we were especially interested in how processes of the

teams vary in requirements updates, assignment changes (especially abandonment) and exposure to volunteerism. We found a negative association between how often a team updates the specifics of a requirement and how often requirements remain unfinished. There is an even stronger negative association between our completion measures and the rate of participants removing themselves as ticket owners within a project. Stronger volunteerism indicators are associated with greater completion rates but not nearly as much with the very largest projects to the point that we suggest the effectiveness of occasional delegation may be a factor in the biggest projects succeeding. We then wrapped up the case study by highlighting some process indicator measures that are common in the teams that score highest on our completion measures.

4.2 Introduction

In order to better manage the risks associated with the unpredictable nature of software development projects, there has been a great need to provide industry with guidance on every aspect of software development management since the industry's inspection. In this pursuit, researchers have many methodological options that include case studies, surveys, simulation modeling and scientific experiments. However, despite its popularity in Software Engineering research (Bai et al., 2011), case study findings are sometimes criticized as not readily generalizable(Tsang, 2014) at least in part because the "sample size in any case study research project is never going to be large enough to qualify for the use of statistical inference (Easton, 2010)." Scientific experiments might provide stronger data but they are usually very expensive to carry out (Zelkowitz & Wallace, 1997) (Ur et al., 2007) in the software development environment primarily due to the value placed on an engineer's time. Simulation models are often the popular alternative but many are not sufficiently based on hard data. Therefore, some have turned to leveraging the data that is created during the development of a software project to strengthen the validity of their research.

For decades, researchers have been harvesting data from any tool involved in the development or management of software products and referring to it as Mining Software Repositories (MSR).

This includes tools that manage the code base and the many changes that occur to it over time; server logs that track the use and performance of the product online; as well as testing servers that confirm the stability of the products intended functions. As we are ultimately interested in team dynamics and project evolution, we chose to focus on the data that are heavy in events and, thereby, workflow information. The approach that we chose focuses on analytics derived from data residing in tools that can help a software development team manage their requirements. There are many names for these systems but we refer to them broadly as "requirements repositories."

We believe the research potential of this type of data is vast and includes topics such as studying project workflow, investigating variations in contributor behavior, predicting project success, and generally allowing researchers to better understand the software development process.

In this paper we choose to focus on process. We seek insights both in identifying the process choices teams make and possible ways to measure the team's effectiveness to move a product forward. Inspired by previous work, we refer to many of these findings as "Project Process Analytics." The more established term of "Software Analytics" focuses on the more broad objective of improving the use or development of software by turning vast amounts of raw data into "actionable insight to inform better decisions related to software (Buse & Zimmermann, 2012)." Our overall objective in this paper is to show that a great variety exists in the process choices made my different software teams and that our Project Process Analytics can be leveraged facilitate discussions about what processes are best for a given team. We do not for one minute argue that this kind of effort will identify cause and effect relationships, but we do hope that the insights we present here will give teams numerous considerations to take into account when evaluating of the advantages and disadvantages of their process choices.

In order to demonstrate our Project Process Analytics, and enable other kinds of studies in the future, we have created a system that acquires, verifies and analyzes data from requirements repositories. Although the system we designed could ultimately store and analyze data from almost any of these requirements repositories as they tend to use similar data, we decided to start by developing the plugins that would collect data from one of the most popular requirements

tracking tools called Jira[10]. The primary reason for choosing this focus is that Atlassian has made their flagship product free for the major Open Source Software (OSS) organizations starting back in 2002. This has allowed volunteers of all different backgrounds to come to know their product and later advocate for its use in the corporate setting. This policy has also made over 13 years of project management data readily available to analysts. In fact, as of January 2015, 12 OSS organizations publicly advertise using it. Alternatively, many academic studies have utilized requirements repository data from the open source tool Bugzilla as it has a feature to export the data easily while the commercial product Jira has no such convenience. This is probably why sampling similar studies is much more likely to be dominated by efforts that use data from Bugzilla rather than Jira.

So the original motivation to develop our system came from the desire to make this tremendous amount of data readily accessible for analysis by researchers. As a starting point, we harvested data from one of the largest Open Source communities, the Apache Software Foundation (Apache), which had 448 projects managed by their public instance of Jira during our first pull of data.

After reviewing some previous work and clarifying the source of our data in section 4.3, we describe requirements repositories and the data within them as a means to explain its research potential in section 4.4 and 4.5. We then explain our elaborate system for capturing this data and how we store it in section 4.6. Section 4.7 shows how Software Project Analytics can profile an OSSD project and it also shows the incredible variety of teams that belong to Apache. We then make a case for why process mining this data will give great insight. Section 4.8 explains why a pattern mining platform was needed to allow for highly scalable and repeatable studies of this or any process mining effort. Section 4.9 is then a study of both process choices teams make and some of the outcomes that may be influenced by them. Before concluding we make suggestions for other research topics that software engineering researchers might undertake with this data in sections 4.10 and 4.11.

[10] JIRA is often stylized in all capital letters, but as it is not an acronym, we chose not to confuse our fellow researchers and keep it simply as "Jira."

4.3 Previous Work

In the pursuit of empirical data of any kind, we are often faced with either an insufficient amount or so much abundance the value has the potential to get lost in the weeds. This is where analytics comes in. In (Thomas H. Davenport, Jeanne G. Harris, 2010) they define analytics as "the use of analysis, data, and systematic reasoning to make decisions." Software Analytics is therefore a method of being certain that better decisions are not lost to the tremendously tall weeds of the software industry.

Although empirical data has been used to study software engineering for decades there still remains a tremendous need for the further development of Software Analytics (Buse & Zimmermann, 2012) as so many decisions in the software industry are made based on gut instinct motivated by experience rather than using tools that base their insight on real data. In (Buse & Zimmermann, 2010) they argue, "the idea of analytics is to leverage potentially large amounts of data into real and actionable insights." We have set out to contribute to this campaign for greater insights into the management of software development.

4.3.1 Mining Software Repositories with Requirements Repository Data

In years past, seeking empirical data from the tools used to manage software projects might also be referred to as "Mining Software Repositories" (MSR) and some of these studies focus on the data within requirements repositories. In many, the text within the tool is the focus such as when (Weiß et al., 2007) created a predictive model to estimate the effort involved in resolving a requirement or in (Lamkanfi et al., 2010) where they attempted to automate the tagging of the appropriate priority to a requirement. Others pertain to the actual use of the requirements repository tool and how to identify poor requirements writing using an automated tool (Schugerl et al., 2008). There is still a lot of work that could be done with the text of the requirements, but ultimately we are interested in the evolution of software projects so we have chosen to focus on the data within requirements repositories that pertains to the countless events that tie a user to the work that they contribute.

Research efforts using this type of data have addressed many questions including some that are policy oriented. (Jeong et al., 2009) introduced the use of Markov Chains to capture the assignment history of a given requirement. Through this advanced analysis of work flow within a project, they were able to establish networks of developers and realize team structures from the context of ideal work assignments. Other research efforts have also addressed team dynamics. In an attempt to study different forms of clarification during the countless conversations throughout the software development process, (Knauss et al., 2012) identified these patterns automatically so that managers or involved stakeholders can be made aware of requirements that should be closely investigated. The research addressing process questions in software development using this specific type of data is very inspiring but we established that the list is still remarkably short and there are so many questions left unanswered.

4.3.2 Open Source Project Evolution

In seeking MSR papers based on requirements repository data, we found that very few of these papers explicitly cite the use of Jira, but some do. (Steinmacher et al., 2013) tracked the activities of newcomers to an OSS project and attempted to link the nature of their interaction with others to find a relationship with the quality of those communications and the likelihood they would become long term contributors.

Using data from the source code repository SourceForge, (Bantelay et al., 2013)(Comino et al., 2007) both attempted to evaluate and track the success of OSS projects, which is a pursuit we find to be multidimensional. (Comino et al., 2007) in particular found that, “the size of the ‘community of developers’ increases the chances of progress but this effect decreases as the community gets larger, a signal of possible coordination problems.”

4.3.3 Process Mining

Data mining is the analytical pursuit of insights from vast quantities of data using methods from statistics, computer science and probability theory. Both the quantity (Krzysztof J. Cios, Roman W. Swiniarski, Witold Pedrycz, n.d.) and format of the data (Han et al., 2012c; Tan et al., 2005;

Ye et al., 2015)) are a part of the challenge which is why the data mining process includes the collection and cleaning of data before the analysis can begin (Baker, 2010).

When data mining techniques are applied to process logs, it is often referred to as process mining (Rebuge & Ferreira, 2012)(W. M. P. van der Aalst, 2016)(W. Van Der Aalst et al., 2012)(Huang & Kumar, 2007) or sometimes workflow mining (Berlingerio et al., 2009; Schur et al., 2015; Silva et al., 2005; W. Van Der Aalst et al., 2004)(Li et al., 2011). In (Sunindyo et al., n.d.) they specify that process mining assumes that the data in the event logs contain:

1. A specific activity
2. A process instance (sometimes called a case)
3. A person performing the task
4. A timestamp

Our data follows these requirements as each ticket, which is out case or process instance, has a series of events (activities) tied to it with a user and timestamp associated with each action. Of course, the order of events within a case (ticket) is important but the order of the tickets within the projects are not just as in (W. Van Der Aalst et al., 2004).

The primary purpose of process mining is to discover, monitor and improve processes (W. Van Der Aalst et al., 2012)(Lou et al., 2013). Discovery can come in many forms but, for instance, we might use process mining to discover the different roles participants play in a team (Checking, 2016). Monitoring a process is important for many reasons, not least of which is a reality check as there is often a surprising gap between what team members and management believe is the process being followed and the processes that are actually utilized (W. Van Der Aalst et al., 2004). It also has been argued by (W. Van Der Aalst et al., 2004) that the goal of workflow mining is to “find a workflow model.” We argue this can be a form of discovery and monitoring all in the hope of improving a process. Our efforts are best described in this way.

4.3.4 The Research Opportunities Abound

Through our investigation of previous studies we have established more work in Software Development Analytics is needed and we have chosen to focus on analytics that can be derived from requirements management tools that we refer to as requirements repositories. We have

also identified a short list of work that, instead of focusing on the text with in a requirement, focuses on using data within a requirements repository that identifies all the changes (events) associated with the evolution of each requirement and who carried out these changes.

Inspiring previous work has addresses team dynamics and tracks, especially within the open source community, the different types of contributions we can identify using this kind of data. But we found this list remarkably short. Therefore, we set out to create a system that would make it easier for the research community to leverage this type of data and to promote this area of software development analytics.

Much of the rest of this paper provides great detail into what kind of data is managed by our system but along the way we also elaborate on the many studies that the data would support. Inspired by previous efforts, we focus especially on the contributors of OSS projects and the dynamics that exist within their projects.

4.4 Requirements Tracking Repositories Data

Not long ago, the details describing new features of an application or software system would usually be articulated in some form of a requirements document, meanwhile the defects discovered in the delivered product would usually be tracked in a “Bug Tracking System” (BTS). More recently it has become common place to attempt to break a new feature into pieces such that an individual developer can commit to delivering the piece in a timely manner. These deliverables are best managed using an application that communicates their details and priority, as well as tracks the person that is assigned to them, and reports their progress just as defects were in a BTS. Therefore, requirements for new features are now often managed in the same applications as the details of the defects discovered in these deliverables. So for our purposes, we are interested in project management data that pertains to new features and defects but we will refer to it all as coming from a requirements repository.

Although utilizing requirements repository data in research data has many challenges, (Canfora & Cerulo, 2005) argue that requirements tracking data from OSS projects will be the most comprehensive because the contributors in an OSS project are completely dependent on the

tracking tool to organize and negotiate requirements. Unlike industry projects, OSS contributors are almost never in the same room and they rarely know one another so an OSS instance of Jira, or another requirements repository, will have a data set that is much more complete than in a commercial setting where verbal communication can make brevity a temptation (Mockus, Fielding, & Herbsleb, 2002b).

4.4.1 The Tables

In a requirements repository, Tickets (sometimes: Work Tickets) are effectively units of work that a software engineer will attempt to deliver to the project. They include a description of what is usually a new feature, an enhancement or defect (bug). Requirements also live in the dialogue represented by in comments, the summary (title) and sometimes attachments.

Tickets belong to projects and are often tagged to involve at least one part of the product called a component. Users write, evaluate, deliver, verify and generally manage these tickets. In most systems, every change to the ticket is tabulated in an event that we generically refer to as a history record. Each of these records has a time stamp and the associated user making the change. In the case of a change of assignment, there is also the new user owning the ticket that is associated with this important change.

The workflow for each ticket varies greatly based on the state of the team, the demand for the "change" being discussed and the policies of that project. We are especially interested in evaluating event driven data as it is our greatest insight into the activities of the contributors.

4.5 An Overview of the Data

In a previous section we explained how most of the data tables related to each other and what their purpose was. We will now explain the most critical fields within these tables so that the significance of this data can be fully appreciated.

4.5.1 Tickets

The focus of this entire system are the tickets, each of which has a long list of "required" fields and quite a few optional ones that bring us great insight, however infrequently. The ID and the

key uniquely identify the ticket. Three time stamps are available: “Open Date,” “Update Date,” and ideally “Resolved Data” but not every ticket gets resolved. Similarly, the user that created the ticket is recorded as well as if the ticket is assigned to someone, which occurs almost exactly 2/3rds of the time at the Apache Foundation.

The fundamental text of a Jira ticket is not our focus, but we do record how many characters are in the tickets title (A.K.A. summary) and its sometimes lengthy description just as with the comments. Every ticket also has to be assigned to a project as well as have an IssueType, priority and status.

The choice for IssueType has many options, but this field is usually used to distinguish “Defects” (A.K.A. Bugs) from “Enhancements” or “New Features.” There are also rarely used custom IssueTypes that are of interest, particularly those that might indicate the use of a certain process.

Priority represents the relative importance of a ticket to other tickets. Officially, the choices are “Highest”, “High,” “Medium,” etc. But these labels are customizable and the choices most teams make on the labels isn’t nearly as interesting as how often they choose the six given levels of priority and who on different teams determines this choice.

Status is the most fundamental way to indicate where a ticket is in its workflow(Atlassian, n.d.). Officially, a ticket starts as “Open” and once completed us usually moved to “Resolved.” When this proposed change is confirmed, it is supposed to be “Closed.” There is also the option to customize the status types.

Once a ticket is moved to “Resolve,” a user needs to indicate what the resolution is. Aside from the obvious “Fixed,” a user might choose “Won’t Fix” if there is reason not to move forward on the issue (E.G. it is no longer relevant), “Duplicate” if the need is represented in another ticket, “Incomplete” if there is not enough information in the ticket, or “Can’t Reproduce” if the user was not able to reproduce the bug with the given information.

4.5.2 Events

The record in the Events table always involves a timestamp, a change to a ticket and nearly always a user. Much of our analysis will pertain to this data that is often excluded in MSR papers.

So far there are 108 different types of changes to a ticket in the Apache Foundation, but the most interesting are changes in Status, Resolution, User Assignment, and Versions as they are key changes in a ticket's workflow. In this paper we focus on the resolution and especially creation events.

4.5.3 Projects

The Project object has a key and name but its effective purpose is to aggregate a list of tickets and the users that contribute to those tickets. As different teams have different ways of going about their work, there are tremendous research opportunities in comparing these projects and their evolution. The challenge in comparing them is that these projects come in all different shapes and sizes.

There are 445 projects in the Apache foundation Jira instance. As of January 2015, there were 6 projects with zero tickets and one with 34,694 tickets. The projects have also been active over many different periods of time. The project "Batik" started in January of 2001 and still has active tickets as of the writing of this article. Meanwhile, of the projects that do have at least a few tickets, some were only active for a week or so. So this leads to our first research question:

> **When studying different OSS teams, how can we appropriately compare projects of such incredibly varying sizes and time periods?**

Another challenge in comparing projects is that some have very different backstories. For instance, Apache Flex is *by far* the biggest project but it is also special because it was a funded project at Adobe Systems for 7 years before being donated to Apache in 2011. For confidentiality reasons, all the users in the project during the Adobe years are tagged together as "adobeJira". Therefore many of our metrics will not be valid in this large project because dozens of developers and countless more contributors are represented by one user tag. We still find this project very fascinating as it is a rare look at the dynamics of a commercial environment, even if some of the data is masked.

4.6 Our Data Capture System

The system that we have developed is capable of acquiring nearly all of the public data in a Jira instance, minus the actual text[s] belonging to requirements[11], and then analyzing said data from a variety of angles using many statistical tools.

The reason we do not copy the actual requirements is primarily to save hard drive space, but also so that we can potentially tell any group that we are not interested in threating their confidentiality, we are primarily interested in studying their team dynamics. But this data is not available in an easy to import CSV or XML file like many of the tools normally used in MSR studies such as Bugzilla. So a major part of the contribution we have made to the MSR community is our development of modules that collect the necessary portions of the data in Jira and stores it in a relational data base. This data base then enables an endless list of studies as it's built within a framework of open source tools that are very popular among researchers leveraging statistical methods.

4.6.1 Tech Choices

For budgetary and repeatability reasons, our system is completely based on open source technology. In the interest of future researchers benefiting from our experience, we explain some of our technology choices, particularly the programming language.

4.6.2 Our Primary Language: Python

The primary programming language for this system was not chosen due to the current skillsets of the team but rather the need to facilitate powerful statistical analysis along with the ability to build a complete application harnessing the web. A thorough investigation indicated there was only one clear choice: Python

Python is a completely free (open source), "Object oriented scripting(Lutz, 2007)" language that was invented in 1991. Many internet sources make the claim that it is very popular among academics, engineers, and data scientists alike when developing projects requiring the analysis

[11] Although we do record how long summaries, description and comments are at the time of the capture

of large bodies of data. However, this can also be said for SAS, R and Matlab, so a thorough investigation was justified. In the interest of future research team's following our example, we present a detailed explanation here.

The inner computational strength of Python comes from a library, NumPy, that was added early in its history to allow for an incredibly efficient manipulation of very large vectors of data. (Van Der Walt, Colbert, & Varoquaux, 2011)(Lutz, 2007). All previous examples of this level of computational efficiency had required a compiled language such as C or C++. However, using a compiled language requires that every time a change is made the developer needs to wait a period, often many minutes, for the code to be converted to binaries. Meanwhile, Python is known for its "rapid turnaround" (Lutz, 2007)(Grandell, Peltomäki, Back, & Salakoski, 2006) as it is technically an interpreted language. This is the main reason that developers prefer interpreted languages such as Python to many other languages in terms of development time. So over the past decade and a half, Python's ability to perform large scale computation along with its popularity among developers has led to other Python libraries being developed within the open source community to support a tremendous variety of analysis almost right out of the box (Grandell et al., 2006).

So when in combination with these mathematical open source libraries, Python is said to have many of the same statistical capabilities of R, SAS or MATLAB (McKinney, 2012)(Nilsen, 2007), while still maintaining its status as a fully functioning object oriented language capable of being used to build enormous systems (Lutz, 2007)(Chudoba, Sadílek, Rypl, & Vorechovsky, 2013). Also, unlike the other infamous mathematical languages such as R and MATLAB, Python is simply very web savvy as the most popular frameworks built in Python are based on the server-browser paradigm. This was important to us because we wanted to leverage the established power of the web browser for our user interface and also offer the option of making parts of this system open to anybody. This ability to support "Programming-in-the-large" is unlike some of the popular early scripting languages of the past like Perl (McKinney, 2012).

Python also has an incredibly enthusiastic and generous community that freely evangelizes knowledge via blog posts and forums(Lutz, 2007). So if we ever do find some sort of

computational limitation, the community can point us to packages that are easily tied to scientific plugins written in binary languages like C, C++ and Java.

That Python has a strong community also means that answers to questions on almost any development challenge will be easily found but even this is less of a concern when compared to other language because Python is universally accepted as one of the easiest main stream programming languages to learn (Zelle, n.d.)(Lutz, 2007)(Grandell et al., 2006). It's also considered one of the cleanest languages, allowing developers to write it incredibly quickly (A.K.A. it is very "writable") (Grandell et al., 2006) (McKinney, 2012). This is primarily because of its simple yet structured syntax but also due to its use of dynamic typing. We contend that academics that need to write code should learn a bit about proper control structures and they probably should learn the difference between a hash table and an array. However, in Python there is no compelling reason to learn additional computer science concepts such as the tradeoffs between a floating point number vs a "double" like you would if they were developing in Java, C, or C++. In Python data types are just a bit simpler. Similarly, the syntax vaguely resembles English with the unusual use of "is", "pass" and "not" so that it is a very easy for one developer to read another developers code. This "readability" is furthered by the community convention of "camel_case" that encourages very descriptive variable names. So in short, as many of the authors of this article are first and foremost Industrial Engineers, Python's reduced learning curve, reputable write-ability and genuine readability was incredibly appealing to us.

So in the end Python was chosen to build this system for many reasons, but primarily for its strong and justified reputation as a language capable of building a large and stable system while simultaneously being capable of satisfying almost any statistically analytical request all while being easy to use.

4.6.3 Our Choice of Framework and Database

A secondary reason for choosing Python was that we would be able to use the very popular framework, Django (Jang-Go). The tradeoffs of a "heavy" versus "light" framework are beyond

the scope of this paper, but the motto of the OSS community that wrote this framework sums it up well:

> "For Perfectionists that have deadlines."

Using a stable and established framework like Django means that, although we would be forced to follow many design patterns proven to work by thousands of developers before us, we would also gain a lot of capabilities right out of the box. But since the original developers of Django had already confirmed that their design choices are wise practices, this group of researchers was happy to follow suite and we encourage others to do so. There is only so much inventing that we had time for and we preferred to focus the invention process on analytics and not ways to map data from SQL to python objects.

We also wanted to start with a traditional relational data base so our obvious choices were MySQL, SQLite and PostgreSQL (AKA Postgres). Our scalability needs pointed us toward PostgreSQL, but further research established that the most current list of data types and the strongest community also pointed is to PostgreSQL. So we were able to move forward with Postgres without hesitation and we hope others do so as well.

4.6.4 The MultiStage Design

Conducting a study using our system requires three phases of activity. As mentioned, Jira does not provide a simple means to export its data. Therefore, the first phase of activity is acquisition of the data through two different types of web requests. Being that this phase literally takes days and is dependent on these sometimes unreliable requests, our second stage is to verify that we have all the data we expected. Lastly, we use the power of all of the python statistical libraries to analyze the data in the last phase.

4.6.5 Acquisition Stage

One of the reasons we were excited to take on the underutilized source of data is that an open source library called "Jira-Python" made manipulating the REST API of Jira much more manageable. REST stands for REpresentational State Transfer. It is a very widely used

"Architecture Style" for web services that are built to serve a very large audience. (Fielding & Taylor, 2000) W3C, the standards board for the web, defines REST as:

> "a software system designed to support interoperable machine-to-machine interaction over a network"

In other words, a REST interface is effectively a means for a server to offer data over the web. In the case of the REST interface that Jira offers, a developer can request most but not all of the data in a Jira instance and, if they have the proper credentials, make changes.

Being that the servers we are requesting data from are owned by non-profit open source organizations, they are not incredibly powerful and must serve the entire community. Therefore, early requests for tens of thousands of records from Apache were rejected. We discovered that requests of about 200 records at a time are usually honored so that is what the initial phase of our development was all about: Creating a robust module of code that accepts a response representing 200 records and reliably inserts then into our data base while recording the progress made.

The REST interface for Jira was originally created so that organizations can tie their Jira instance to other tools such as their code repository (A.K.A. software configuration management (SCM)), email or automated testing servers. Of course, being a for profit company, they have had to prioritize the development efforts of this REST API to provide access to certain data types. Therefore, some Jira data is not available through the REST interface likely because it would almost never be relevant in the corporate setting. So in the second half of the acquisition phase, we use "web crawlers" to obtain the remaining portions of the data.

A web crawler (A.K.A. Web spider or an ant[12]) is a program that parses the same response that a web browser receives when a user is surfing the web. These are most famously used by the internet giants Google or Yahoo to index and catalogue the entire open web, but in this case we are requesting and parsing the code that would normally be sent to the web browsers of the more typical Jira user.

[12] An "ant" is an automatic indexer

These crawlers are a viable option because the committed Python community has more than one library available that will turn one web address into one large file of HTML code. Other open source tools then allow a very efficient parsing of this file to painstakingly identify the data amongst all the code that is normally used by the browser to render the user interface. So obtaining this part of the Jira data is a lot more work, but it also allows us to obtain data rarely used in these types of MSR studies. Specifically, it allowed us access to the Events table (on the tab labeled "History") on each ticket as well as the details of the attachments associated with each ticket. As much of our analysis is based on the events data, this was well worth the investment.

4.6.6 Verification of the Data Stage

One of the challenges involved with both of these acquisition steps is that they are dependent on responses from servers, both web and REST. They also take a tremendous amount of time. The most optimized version of the REST portion processes about 8,000 tickets per hour and the latest spider processes about 6,000 tickets per hour. So any sort of manual monitoring of this process is not scalable and simply too expensive. However, the integrity of this system is completely dependent on the data acquired being exactly the same as the data living in the Jira instance being studied. Therefore, we designed into the system a series of verification techniques so that we can always be confident that what is assumed to be in the data base, is in fact in the data base.

In the interest of contributing to the open source python community, we designed a configurable and reusable[13] global logging system so that errors and warnings could be written to a universal set of files by using just one line of code. This way it is incredibly easy to identify a run-time error, usually caused by an unexpected data format, collect all of the details of when and where the error happened, and write it to a log file to be studied later. Another reason we liked this approach was that since Python is a language that supports the exception handling design pattern, the error can be dealt with and the execution of the program can continue. We also

[13] Of course, this small bit of reusable code can be used by any one and we named it "Sammy Logs" after the lead developers first child.

reported errors at different levels of concern so that all the errors were “available” at all times, but the most critical were most prominent.

So during the development stage, the routine was to initiate one of the steps of the acquisition stage and then come back hours later to analyze the logs. Of course, in the beginning errors came up right away and were resolved quickly. Eventually it would take hours before an unexpected response such as a very uncommon data format would cause an error. This was the primary reason we needed this logging approach as waiting around for an error was not the best use of our time.

The other method we used to verify our acquisitions was by tracking progress in metadata. The primary tables of pertaining to the Users, Tickets and Projects all have supporting tables that are effectively statistics on each row of their parent table. Maintaining expected total record counts and verify our progress in capturing this primary data are among the most important uses of these meta data tables. For example, a request can be made to the Jira REST API to determine the total tickets in a project. As this is a rather expensive call, we would like to only make it once so we record it in the “Project_Stat” table. Then at any time, we can compare this figure to the total tickets we have in the system for that project and reconcile any discrepancies.

Another example of leveraging the metadata is during the spider phase of acquisition. As the spiders are background processes, our logs might have been the only way to determine if an error occurred. Unfortunately, most of the effectiveness of the logs covers runtime errors like unexpected data formatting. If a server gave an incomplete response, we might not catch this in the logs. However, we do know that every ticket has at least one Event record because Jira always indicates the create event. Therefore, the verification of one event record proves that a response was given to the spider for that ticket. Also, at the beginning of the spiders parsing of the web response, we are able to determine how many total “change Events” should be parsed based in the HTML structure. So this is another example of a piece of metadata that is stored for future verification. If a ticket has fewer Event records than were initially in the page that was parsed, we have a discrepancy to resolve.

In addition to verifying that we are not missing any data, we also want to be sure we do not have too much. This is why we designed another form of "after the fact" verification to double check that all fields in the three primary tables (users, projects and tickets) that should be unique, are unique. If we actually created two objects for the same entity in Jira, we would want to determine how this happened.

This duplicate scenario is unlikely but plausible because our system is over 40,000 lines of python code (as of April 2015). We have every intention of being sure to only create an object when we know it doesn't already exist, but with all that code, mistakes could happen. So for instance, in the case of a project record, there should be no other records with the same key, id or name. We therefore confirmed this with very simple functions that run in a fraction of an hour to confirm this and all other assumptions about the Ticket and User tables.

4.6.7 Analytics Stage

The findings of this paper were discovered using a long list of scripts in the analytics stage. Powerful Python libraries were utilized such as statsmodels, numPy, SciPy, and especially Pandas. The common pattern we utilized was to allow Django to provide a highly readable query to fetch a large set of records and NumPy or Pandas would then manipulate this data to produce statistics. The tools provided by this community made some of these findings so simple, it was often easier to rewrite a script than to find it again, hence our biggest regret: proper documentation and greater modularity in code organization.

4.6.8 Data Base Design

In order to appreciate the full potential of this data, it's important to have a basic understanding of the relational data base schema that we designed to capture and analyze this data.

4.6.9 Primary Tables

Although our system captures the JIRA data in a few dozen tables, there are four that are the most crucial: Tickets, Projects, Users and Events (A.K.A. "Histories"). A Project is a required foreign key in a Ticket as all tickets belong to one and only one project. As every Ticket has a

user that created it, most should be tied to a user that is assigned to it, and ideally most should have a user that has resolved it. A Ticket is a required foreign key in an Event record because an event is moot without a Ticket to tie it to. An Event nearly always is tied to a user object[14] so User is also a foreign key in an Event record.

4.6.10 Secondary Tables

We refer to all the tables that are used repeatedly per ticket or project but with only one or a few attributes as secondary tables. The best examples of these are the many components or versions tied to each project and the many attachments or comments tied to each ticket.

A version represents how a project is periodically delivered to the public. It's effectively a long list of code changes, each of which is tied to at least one ticket. Although the code change events managed in a Source Control Manager is not currently a part of the system we created, the event that represents the change of code and the confirmation of its validity at resolution is a critical part of many of our analytics. These code changes and the tickets they resolve are grouped into versions that the world then requests to install in order to use these applications. So each project has somewhere between zero to hundreds of versions associated with it.

Most projects are broken up into components so that some sanity can be brought into the large code bases be designating a level of modularity. Users are known to specialize (Hayashi et al., 2013) in certain components because in most projects, understanding the entire code base would be virtually impossible. So most tickets are tagged with at least one component, but since many enhancements or defects pertain to more than one part of the code base, sometimes many components are tagged on a ticket. However, it's important to note that a component only pertains to one project.

Every ticket in Jira has the option to attach additional information about the enhancement or defect in the form of a screen shot, written document, or other bit of information clarifying a requirement. Also, on many projects, tickets will be tagged with some or all of the code changes associated with the resolution (often called a patch) through the use of an attachment. We

[14] Unless the user is for some reason deleted, it would seem.

always have a time stamp and almost always have a user associated with these attachments so an attachment record is a very powerful piece of insight into the evolution of a ticket.

As we are not yet involving emails as so many MSR papers have (Baysal & Malton, 2007)(Cerulo, Ceccarelli, Di Penta, & Canfora, 2013)(Bird, Gourley, Devanbu, & Gertz, 2006), our primary source for any insight into the communication within the team is the comments. Comments are associated with a user, a time stamp and a string of text. As we are not focusing on the actual text of the requirements for reasons previously explained, we store the length of the comment instead of the actual string.

4.6.11 Meta Data Tables

As mentioned, we use metadata tables to tabulate progress in acquiring the data for verification and tracking purposes, but we also store statistics that are both expensive and highly reusable. For instance, in order to determine if a ticket as *ever* had a status of re-opened[15], but not necessarily *currently* in a status of re-opened, we need to search every Event record for this ticket. Django queries will manage the somewhat verbose SQL normally required for this request so risks in the coding effort are not the concern, but the time cost of repeated calls to this query is. So we store the answer in the metadata table for a minuscule hard drive cost but a significant processing time gain.

4.6.12 System Limitations

As with many other OSS MSR studies (Gousios & Spinellis, 2014), it's important to note that we have made no attempt to protect the privacy of the users involved. However, the data we are using is readily available online at Apache's instance of Jira. We are simply copying it into the form that is most efficient to study. But that is another limitation. This data is not live. It took over a week for the acquisition module, in its most optimized form, to copy all 496k Jira tickets in the 448 projects associated with the Apache Software Foundation in January of 2015. So our copy of the data is current to that point in time and any changes to the tickets after that are not

[15] A ticket that was declared resolved but then reconsidered for an alternative solution is "reopened."

present. However, since we are studying almost 14 years of data, we believe this downside can be managed.

Another caveat is that there are blatant missing numbers in the ticket key sequences and we have every reason to believe these are due to deleted tickets. So admittedly this issue represents missing data. Of course, only a select group of people have the power to remove tickets from existence so we believe it is only done for good reason. We speculate that the primary reasons that this occurs is when a ticket written that is actually a duplicate of another or simply a poorly written ticket. These tickets are simply going to take up more time to ignore than to just remove. Either way, we believe they are removed for rational reasons and not to remove evidence of a controversial requirement. So they are almost negligible.

4.7 Project Analytics

As mentioned in section 4.6.8, we see great research potential in comparing OSS projects by studying the requirements repository data associated with them but a major analytical challenge is that the projects come in so many different shapes and sizes. Our approach to managing this challenge is to establish as many different ways to describe the projects in the hope that we might classify said projects and the data associated with them. If we were to then compare projects that are more alike, we will have greater confidence in any statistical inferences we make about their differences. We call these specific types of software analytics: Project Analytics and they come in many forms.

4.7.1 Ticket Counts

When observing projects just based on their ticket counts, to say that the distribution is asymmetrical is a gross understatement. Of the 448 projects in the Apache Project, 113 have less than 50 tickets yet 100 have more than a thousand tickets including 23 that have more than 5,000.

A histogram utilizing the logarithmic scale in Figure 17 shows the number of projects with various ticket count ranges in both the created and resolved criteria. This figure excludes the two projects over 10,000 tickets. Note the first column represents over half the projects, 290

specifically, with less than 500 tickets. In fact the mean ticket count per project is 317.64 and the median is 305.90. Upon discovering this wide disparity, immediately we began asking ourselves:

What might allow a project to become ten or 30 times as large as the average size project? What is special about those large projects?

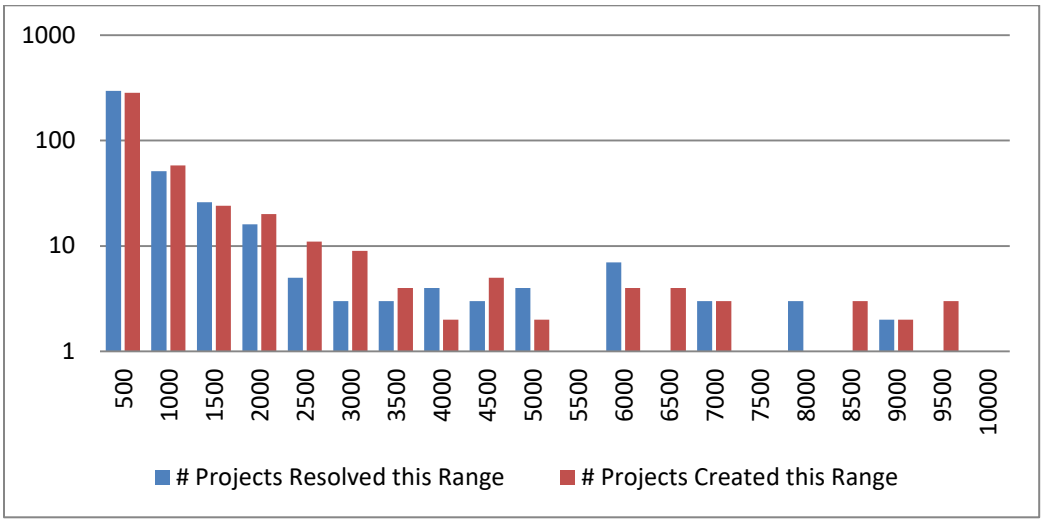


Figure 17 - Histogram of project size by ticket counts, excluding the largest two

The list of reasons why some projects produce many times more work than others could not possibly be covered in one paper, but we propose the use of more than a few analytics that might help differentiate projects which could lead to studies on how they evolve so differently.

4.7.2 Project Life Span

In another Project Analytic, we make note of the first and last change to tickets in a project in order to determine the "Project Life Span." Of course many projects are still under development and we don't really know how effectively active a project is if it only had one user make one suggestion in its final months. Those caveats aside, we find Fig. 2, a histogram of Project life Span in annual quarters, intriguing as the distribution is not nearly as skewed as the ticket count distribution was. In fact, if we consider that many in the recently started projects (10 were started after the beginning of Q4 2014) are likely to make it to at least a couple quarters then the spike of less than one quarter is understandable. So although the distribution of Project Life Spans is not

exactly uniformly distributed over the 14 years of Apache history, it is fascinating how relatively evenly distributed it is.

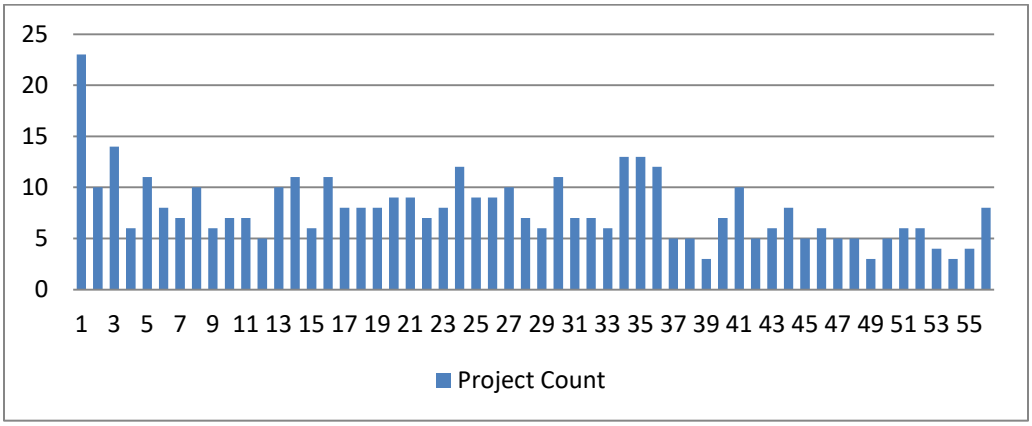


Figure 2 - Histogram of Project Life Spans in Quarters

This unexpected lack of skew led to our strong suspicion that there is a tremendous variance in the *rate* that tickets are written, and especially resolved, even when comparing projects of similarly size. In Figure , we have an adaptation of the previous histogram but with the three major tiers separated. We do see, of course, that of the projects of less than 100 tickets (designated as “small” projects), more of them are less than a few years old but it’s surprising how many are a decade or older. And although many of the projects with over 1000 tickets are many years old, a few are less than five. So we conclude that the rate at which these projects grow and resolve their issues does vary tremendously. An elaborate study would be required to definitively conclude that all projects become busier as they age.

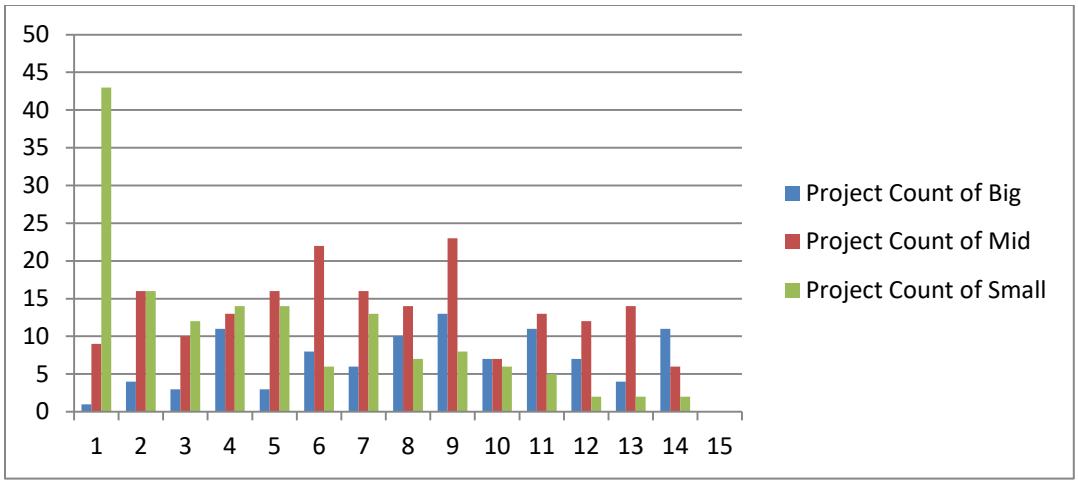


Figure 3 - Project Life Span in years by project size groups

However, if a project with 8,000 tickets wrote tickets at the same rate as the 113 projects with less than 50 tickets, we would have many more short lived projects so we hypothesized that there is some relationship between project size the rate of tickets being written. This suspected relationship would also support our suspicion that the large projects have a tendency to need to "get off the ground" or "gain momentum" at some point early in their history.

This proved to be true as we did find a weak relationship between the *total tickets* written and the *frequency that they are written*. Of course, this is formulating a measure of tickets written per month averaged over the entire time that tickets are being written for a given project. It is easy to assume that the rate has varied throughout the project's history.

Our attempt at simple linear regression analysis first identified some extreme anomalies, particularly a handful of projects that had 50 or a few hundred of all of their tickets written in a short period of time. Interestingly, all of these turned out to be very mature projects that had very recently been donated to Apache but did not transfer their previous tickets into Apache Jira. Just as Adobe donated Flex, Yahoo donated Samoa and eBay donated Kylin. All of these projects are represented in Apache Jira but only for a short period. Therefore in this analysis, we removed them as their relatively low ticket count did not at all represent their level of maturity.

Once utilizing logarithmic scales on both axis, we can visually note something of a relationship between the frequency of new tickets being written and total tickets written[16] in Figure .

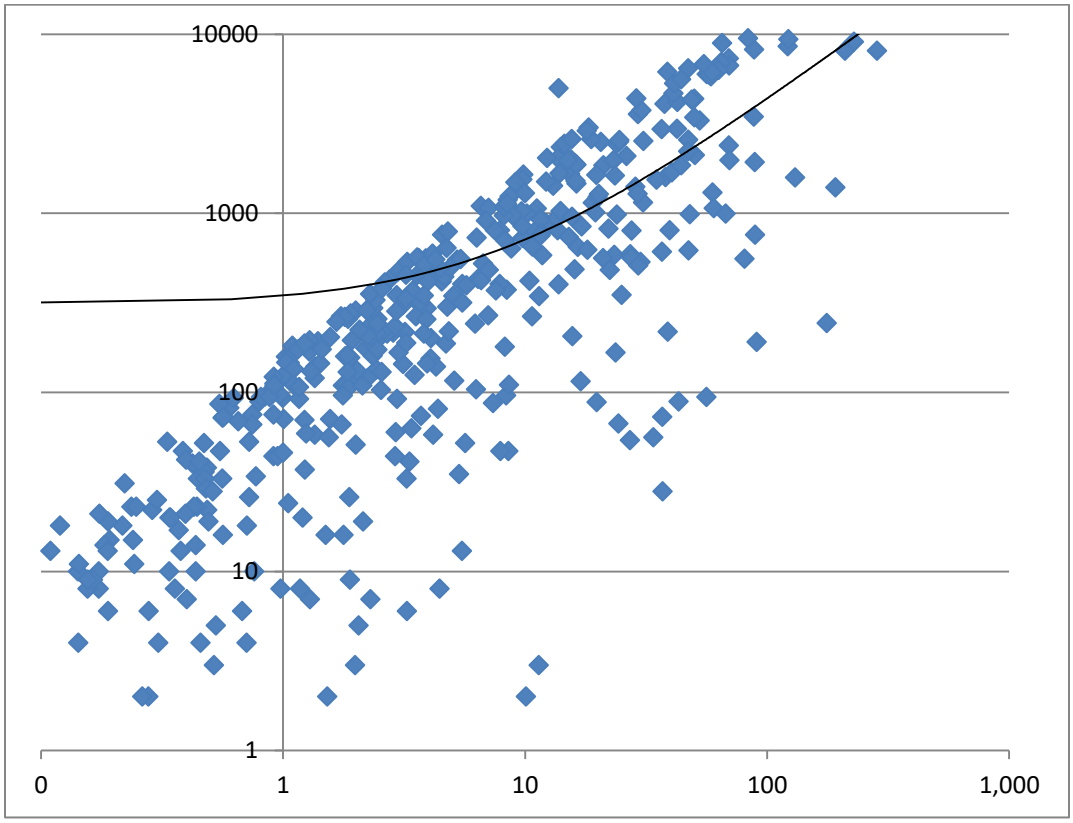


Figure 4 - Scatter plot of Ticket Count vs Frequency of Creation per Month using logarithmic scales

By utilizing the power of open source python packages, we find a linear regression with an R-squared value of 0.620. This is not a tremendously strong relationship but when we consider that the frequency is determined from the entire history of each project and that smaller projects are often slower to gain tickets, the actual frequency of the most recent periods of history in the larger projects might further support this hypothesized relationship. Further studies are much warranted.

To further investigate our hypothesis that the larger projects are gaining momentum, we looked at some of the larger projects and whether the number of tickets written for them were, in a general, increasing. Of the projects with more than 5000 tickets and ignoring the monster that is Flex, six

[16] We did remove Flex from the graph only because visually it is somewhat removed from the rest of the projects but since it has been with Apache for 4 years and appears to include accurate times stamps from its early days, it was included in the regression analysis

projects were very active over the past three calendar years. Charting their new ticket count per quarter showed that only Camel had a small decrease in new tickets over that period. As depicted in Fig. 5 , all others had an overall small increase of tickets or a notable increase in the case of Hive and HBase, two of Apache's most infamous projects.

This is a very limited sample of projects and simply counting tickets per quarter is not very precise but we argue that this is some support for the hypothesis that most successful projects are "gaining momentum" and simply moving forward with increasing speed. Further study would leverage more advanced statistical techniques to identify which projects are in fact trending up and which are possibly down.

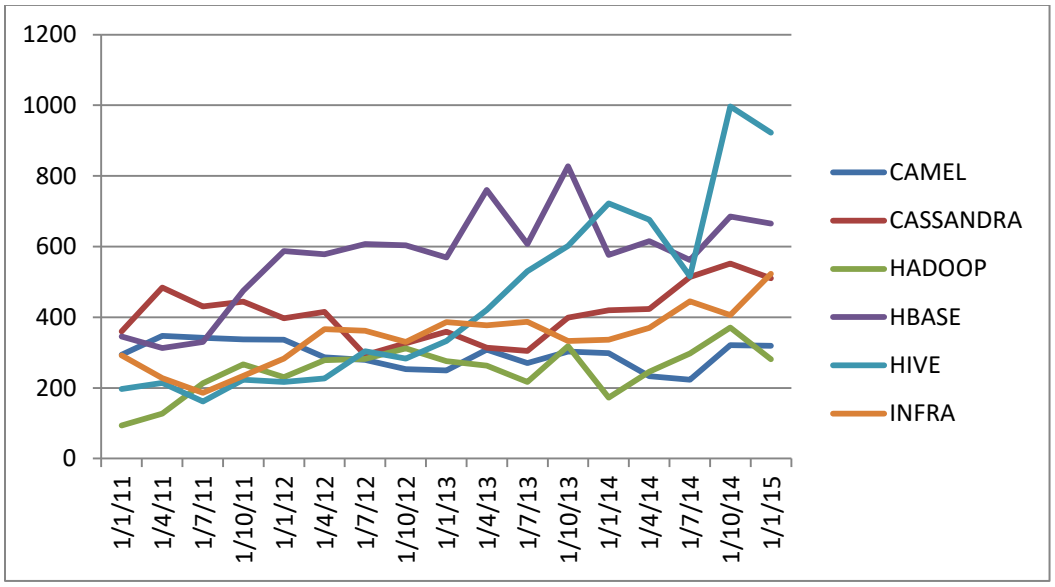


Figure 5 - New ticket counts per quarter of the most active projects in 2011-2014

Of course, the commonality of upward or downward trending projects is not nearly as interesting as the factors that lead to this trend. Are the once successful and now down trending projects losing some of their biggest supporters? What caused this? Admittedly, there could be many factors outside of anything we can conclude from this data alone causing these changes. Such as: Has the market changed such that many in the community are trying an alternate product?

4.7.3 Project Analytics Summary

We have shown a great deal of variety in the projects and teams of Apache and made modest support for the concept of team gaining momentum. If some projects gain momentum and others do not, could there be some objective measures for how a team becomes successful? If so, are there ways to compare the effectiveness of some teams to others to determine best practices? We speculated that the tremendous variety of ticket counts makes this difficult unless we compare the tickets themselves. Within the tickets themselves the great variety is in their content and the processes they undergo. Not wanting to delve into the world of requirements engineering, we set about to designing a process study that would leverage this vast dataset in the pursuit of what processes might be associated with success.

4.8 Our Pattern Mining Management System

In order to undertake this process focused portion of our study in a scalable way, we created a reusable and independent module from our requirements repository data capture system that has the means to allow a researcher to facilitate a pattern mining study with the open source tool SPMF as the provider of the algorithms. This system provides a means to create a queue of experiments with very specific parameters for the state of the data required for a given experiment, the specific pattern mining algorithm with its parameters we are requesting of SPMF and the ability to parse the often vast results set so that the analyst isn't manually reading, in some instances, 10s of thousands of lines of association rules.

The open source tool SPMF was first developed by Philippe Fournier-Viger in 2010 as an open source data mining tool focused on pattern mining. As of 2017, it offers 23 algorithms for sequential patterns alone including: closed sequential patterns, maximal sequential patterns, the top-k sequential patterns, and compressing sequential patterns. Nine algorithms offered cover Sequential Rule Mining, seven for sequence prediction, 34 for frequent itemset mining, two for periodic pattern mining, 31 for high-utility pattern mining, ten for association rule mining, and 4

implementing clustering. It is completely built in Java and the source code is open to the public.[17] The two means to utilize it are through a GUI interface and a command line option. The GUI provides the means to, before launching an experiment, set the options of:

- input/output files
- algorithm choice
- algorithm parameters

This is a great option for getting to know the tool and all the algorithms provided but with all the pointing and clicking, it is not easily repeated over hundreds of experiments. It is also very prone to human error. "What, wait was the minimum support value I used last night?"

The command line provides the same options but with the precarious scenario of requiring perfect strings and numerical values provided for each option or parameter. Clearly this option is best suited in a scripting driven scenario and was the first inspiration for our entire platform.

We could have used many types of algorithms offered by SPMF but we focused our efforts those that provide association rules. We preferred two algorithms CMDeo and TRuleGrowth. In fact, Dr Fournier-Viger allowed us to provide a change to the CMDeo portion of the algorithm such that an additional interestingness measure is provided called lift. Lift is very thoroughly explained in (Han et al., 2012a). CMDeo also is one of the algorithms in SPFM that allows us to limit the itemset size on each side of the association rule. Although there may be scenarios in our data where three antecedent events leads to three consequent events in an insightful way, this would be a rare occurrence as the number of instances of this combination would, in most cases for our data set, be so low that we would be concerned that the results are heavily influenced by random chance. More importantly, the number of association rules that the SPMF algorithm must track during its run grows exponentially as the itemsets get larger. Therefore, when applying a very large dataset to SPMF we often needed to limit the itemsets to two antecedents and two (or often one) consequent to contain the computational complexity. Our system manages which experiments these limits are applied to.

[17] Which allows students to learn the intricacies of each algorithm among other things. Thank you, Dr FV!

Although not the original purpose, our system also became an essential record keeper of all experiments. We allow for a queue of experiments because each experiment is driven by a configuration file and the files can be listed in the JSON file called to launch experiments. Therefore, all the inputs required for SPMF are put on record and tied to output files. This proved a priceless means to maintain one's sanity as experiment after experiment was written. Future efforts will tie these parameters in the database to the most useful rules written so that all information is easily retrievable.

In many instances during this case study, each experiment was not necessarily utilizing our *entire* sequence database as input and we imagine there are many other instances where researchers require the same need. In our data, each case (the entity that all events have in common per sequence) is the ticket and tickets are tied to projects. So in some scenarios we might retrieve all sequences from one specific project. In other experiments we might simply want a random sample of the sequence database instead all 496k non-Flex[18] sequences. Our system provides an easy means to retrieve these specific sequences.

One of the reasons SPMF can provide a pattern mining experiment so quickly is that it does not seek patterns of strings but of integers. The data structures involved are tremendously simpler that way. Of course, as a user it is not terribly convenient or sanity promoting to provide SPMF integers as input and then interpret the resulting rules in that format. So one of the additional inspirations for this module was to provide a means to translate the detail-intense event strings to the integers used in SPMF. Our platform then translates these integer based outputs back to said strings for easy interpretation.

Another option we needed to provide was the ability to ignore all but a specific set of integers during experiments of large scale. This was a transformation we refer to as "masking." In these cases, we might only be interested in certain events as both the antecedents or consequents. Therefore, if we are using a very large input set, we utilize a function that will "flip" all events *not* of interest to a meaningless integer (tabulated as "OTHER" in our data") so that SPMF can

[18] Flex is the project that spent years as a corporate Adobe effort and has masked user data. We needed to ignore it for most of our efforts for this reason.

perform the experiment utilizing a data structure that is miniscule compared to an experiment tracking all events provided. Alternatively, the only experiments that could handle a large data set without requiring a tremendous amount of computing power would be those with rather high minimum support and minimum confidence. The means at which these algorithms use these parameters to contain the memory and processing requirements during processing are best explained in (Han, Kamber, & Pei, 2012b; Tan et al., 2005), but we emphasize that in many cases such parameter restrictions prevent the ability to run many useful experiments. This need to mask the vast majority of the events in a sequence database during experiments of lower minimum support and minimum confidence might be the most extreme case of why utilizing a data managing platform designed for these experiments is so vital to utilizing SPMF in a research effort that requires repeated experiments.

We have cited many reasons this platform is needed while emphasizing the need to manage the state of the integers. Creating the input sequences in this form, and sometimes altering (or masking) all but a few is essential. Translating them back to meaningful strings that had been abstracted from our data was also a tremendous means of saving of time. But in most instances, the output that was now produced, while useful in that it is in some for vaguely resembling a human language, is often massive. So unless you are looking for the most frequent handful of association rules, the team needs to either plan to devote long periods of time to manually parse the rules of interest as we did for many weeks, or our platform can provide numerous options for parsing the results sets that fit the current needs of an experiment.

A full explanation of all the parsing options we were able to provide is best left to a user's manual in future years. However, let us elaborate just on the options we were able to provide without expounding on the specifics. It is based on the idea that one SPMF output file might be parsed into several string based output files based on very different criteria. The three core interestingness measures: support, confidence and lift are an available criteria as well as itemset length. We might allow for two consequent rules in an algorithm run, but then only want to look at a specific instance of two consequent rules in one output file then only one consequent rule in

another. Of course, the string based event labels are the core of this parsing logic and we provide both “AND“ and “OR” logic for the pass/fail logic to determine an association rule’s worth. Lastly, we provide the option to write the resulting association rules to a database table so that they are easily queryable in the future. This became essential once we started keeping track of rules for our 84 largest projects and we imagine many applications will deem it essential as well.

4.9 Project Process Analytics

Now that we have explained our pattern mining platform, we present our process study. We divide our most insightful patterns into two groups, although it can easily be argued that the groups overlap. They are:

1. Indicators of OSSD Team Effectiveness
2. Process Choices Indicators

Overall, our effectiveness measures pertain to a team’s ability to avoid anti-patterns. These are events that, most of the time, represent a hurdle in the workflow that was not ideal. It probably caused a delay or possibly increased the probability of work not being completed. We acknowledge that some of these effectiveness measures are stronger indicators of success than others.

In contrast, we do not pretend to have enough data to know how indicative our process choice indicators are but in a few instances we are able to make the case that teams should consider the relationships we established with the effectiveness measures. In most cases this was simply because the correlation between one of the effectiveness measures and the process indicator was unexpectedly high. In any case, we find that many of our process indicators vary greatly in their prevalence between projects. We argue that just knowing that your project has an extremely high rate of one process indicator and especially one effectiveness measure might facilitate discussions on how work is distributed or how tickets are managed that may benefit a team’s process choices in the future.

To further facilitate these process discussions, we will study our two groups of process patterns using three primary approaches:

- Compare the frequency of one important process indicator or effectiveness measure of one project to the rest of the projects in Apache
- Compare the conditional frequency of a critical event following another critical one, effectively a rule
- Determine the correlation of the frequency of one process indicator with an effectiveness measure

To further our case that some process indicators may be worth discussing within OSSD teams that wish to increase their effectiveness, we also isolate a series of large Apache projects that score quite well on our effectiveness indicators. Anecdotally, we are able to show some of these projects have certain processes in common and we suggest the OSSD community that this into account when evaluating their own workflow patterns.

4.9.1 A Note on the Projects Included

We did not actually seek association form rules for all 445 projects simply because most did not have enough data to justify them. We only wanted to compare projects with enough data that there was little concern that the rules were heavily influenced by random chance. Also, the smaller the minimum ticket criteria, the more outliers we tended to see. We postulate this was due to some projects not gaining enough momentum to have to have any process consistency. But we needed an objective guideline. We chose a combination of support level and minimum ticket count. This was our logic.

We are studying some patterns that might be significant but only exist in as little as 3-5% of their tickets. So in a project that only had 1000 tickets, some patterns would be based on 30 to 50 tickets. At this level, random chance has too big an impact. Therefore we used 1500 as a minimum ticket count in our general studies, but tried to isolate our studies of more infrequent events to 2000 or even 4000. Fortunately there were 29 projects with over 4000 tickets and 85 with over 1500. In all cases, we would not record a rule if the sample was significantly less than 1% of the total tickets in that project. Therefore we are missing rules for some of the smaller of these 84 target projects and we will note this throughout the study. However, we still argue that this is a comprehensive study of the Apache Software Foundation's various projects because all

projects that were able to provide these association rules with strong confidence were included in the study.

4.9.2 Open Source Project Effectiveness Measures

We start with effectiveness measures. For the most part we think of them as outcomes, although usually unintended. They effectively all focus on a lack of anti-patterns, the most glaring of which is an unclosed ticket. When work is complete on a work ticket, or if it is deemed not worthy of a code change, the Jira systems allows it to be closed, resolved or both as we explain in. Based on guidance from the makers of Jira, Atlassian, we consider them equivalent in this study. They both mean that no further work needs to be done.

In Fig. 6, we note that most teams have less than 30% of their tickets open with the most common rate to be about 15%. The Apache wide average is 80.5%, including even the smallest projects. Our most significant take away is that, among the modest to large size projects, some have unclosed rates indicating a very significant to-do list. 20 out of 85 projects have an unclosed rate of more than 30%. The future for all these tickets is unknown but a project with so many unclosed tickets is especially uncertain and we recommend these team's rethink how they manage unfinished work.

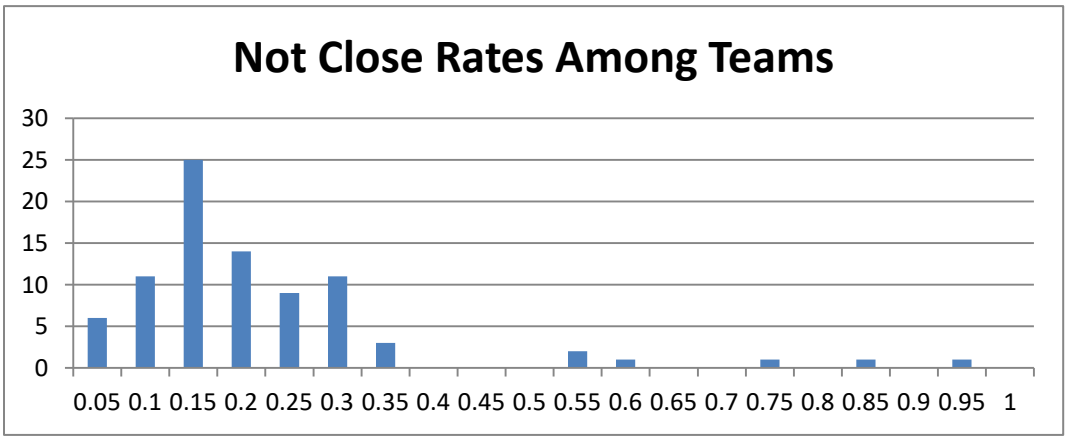


Figure 6 – The frequency of not close rates among teams show that its very common to have incomplete work above 20% and especially above 30%. The teams that may want to evaluate their processes are the 20 above 30%.

When a ticket is marked as either closed or resolved, ideally this means a solution is presented. Unfortunately, many additional scenarios exist. A decision maker can decide that the new

requirements or the description of the defect are not valid or a duplicate. Specific to defects, a decision maker can also simply mark the ticket “cannot reproduce.” The ticket can also be marked with the blunt and feckless mark of “won’t fix.” And some tickets are simply “in-progress” because the ticket is claimed and the solution is being sought but said solution is not yet complete or, especially in certain projects, the ticket can also simply be ignored. The ideal scenario is that the resolution is marked “fixed” because code was presented to make the enhancement, add the new feature or solve the defect.

Aside from being ignored, all of this information pertaining to how the ticket was eventually received by the decision maker (who is often the person who presents the first solution) is obtained by the field of “Resolution Type.” Of the 433,059 tickets that had ever been declared closed, only 0.6% were lacking in a resolution type so we assert it is a reliable means to determine under what context the ticket was closed. As the sought after “fixed” resolution type indicates, amongst other things, that the requirements in the ticket were worthy of some ones effort, they represent a likely workflow that will be different from the other rejected tickets. Therefore, they are worthy of special consideration. So this rate of tickets being not-fixed represents the percentage of tickets that are still with an uncertain future as with the unclosed tickets and the percentage that have been deemed not worthy of the proposed effort. This second scenario explains why, in Figure 18 we see much higher rates than the unclosed rates. In addition, we note that some teams manage to fix 80% or even 90% their tickets. In addition to being more diligent as a team to resolve issues and deliver new features, perhaps they also collect less bogus requirements? This figure notes that not only are their quite a few teams with not-fix rates above the most common of 35% (the mean is precisely 34.5% in all of Apache), but there are 17 above 50%. Although many of these teams are probably already aware of their large to-do list for other reasons, we encourage them to take a look at their requirements collections methods as well. Why are so many suggestions not worth their time?

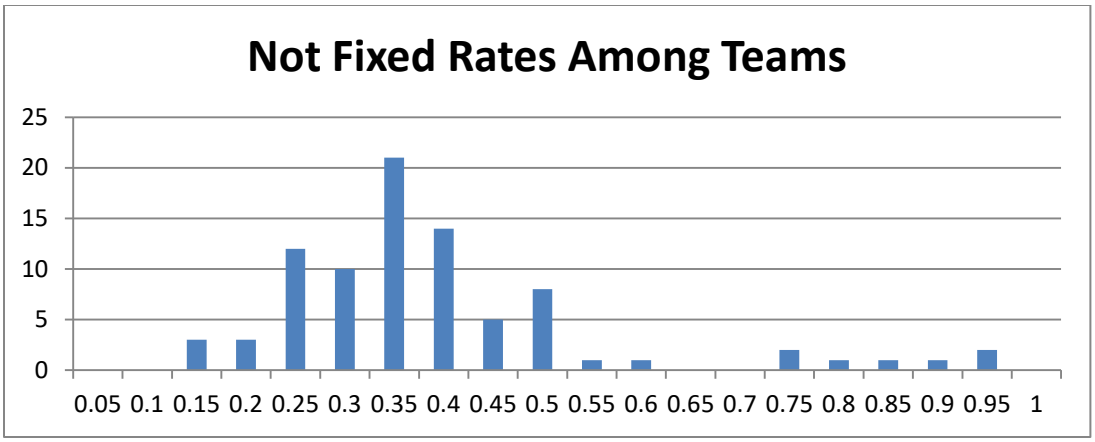


Figure 18 – The not-Fixed rate represents the percentage of tickets that are still with an uncertain future as with the unclosed tickets plus the percentage that have been deemed not worthy of the proposed effort.

In the Jira system, you can add an assignee (sometimes called owner) to a ticket that does not have an assignee, you can change the assignee from an existing assignee, and you can remove an assignee from a ticket that has one. You cannot have two assignees at once. Therefore, removing the assignee is very telling. Unless a user decided to make an assignee change the hard way and remove one, then wait, and then add another; the remove event very often indicates that a ticket will have no one responsible for it for some time, if ever. Did the person give up? Did they become discouraged? Were they fired? The last is an unlikely scenario in these teams based on volunteerism but in any case we argue the incident did not move the project forward and possibly hindered it. Although there are probably some notable exceptions, we consider this process indicator as a signal of likely abandonment.

We note that in Figure 19 the vast majority of projects have less than a 5% removal rate and in Figure 20 we can see that the most common rate is approximately 2% when grouping the rates in half percentile buckets. All projects above these averages might want to take note, but we especially show concern for the 19 above 5%. What are they doing differently than the other 66 that is causing this issue?

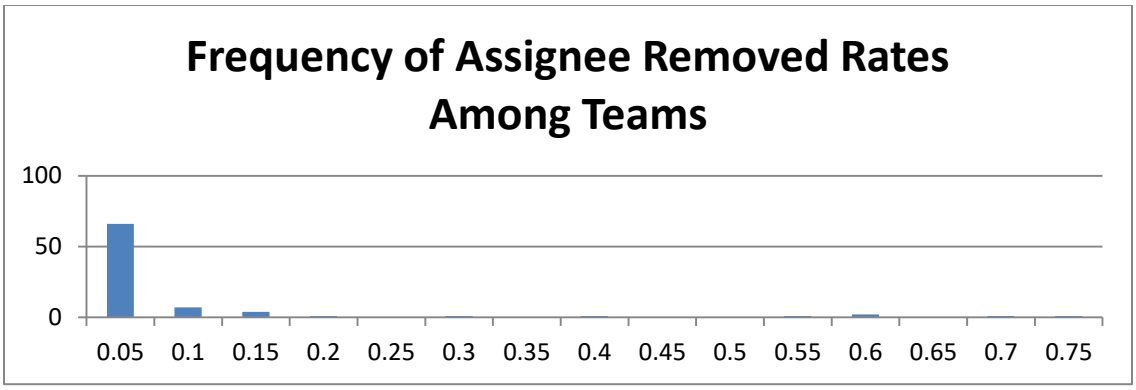


Figure 19 - the rates that assignees are removed from a ticket, what we argue is often abandonment, varies greatly but is quite often below ten percent in these more established projects.

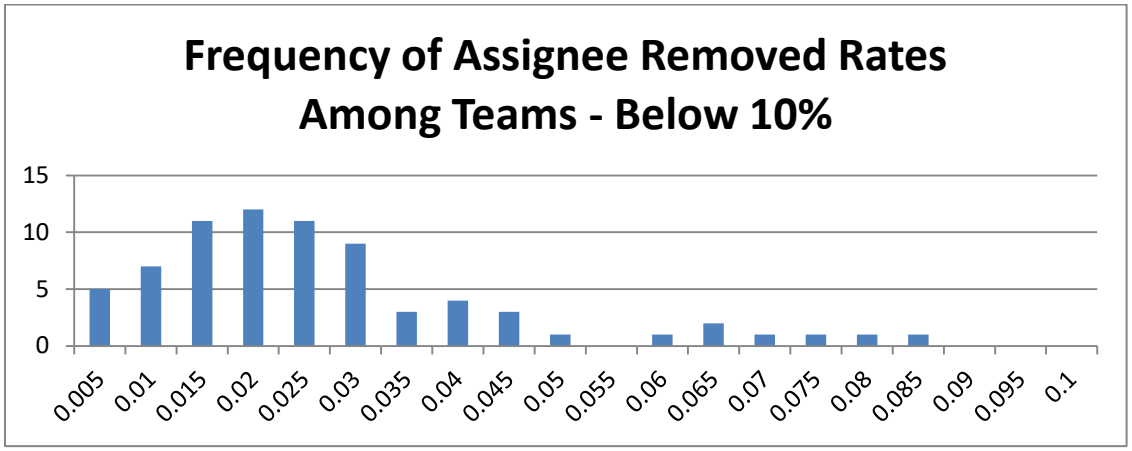


Figure 20 – The same measure as Figure 19 but allowing higher fidelity on those projects with rates below 10%. ~2% is the most common rate at this granularity.

Our last effectiveness measure is supported by previous research to represent delays, wasted effort and sometimes confusion. We are talking about a ticket changing hands from one assignee to another. What is sometimes known as: Tossing. However, we acknowledge that some teams deliberately plan for more than one person to contribute to a ticket. The first might review the validity and value of a set of requirements, the second might perform the work, and the last might verify it. So we will pay homage to previous works by referring to this action as, at the very least, a source of delay and therefore it is an anti-pattern but we acknowledge it might be a deliberate choice in some cases.

Our first observation from Figure 21 was, as there is so much literature on the topic, we would have expected more tossing. But perhaps in a commercial setting this occurs more than at the Apache Foundations and that has been the motivation for previous research efforts. Either way,

we suggest that the 34 projects safely above the organization mode (when utilizing 2.5% buckets) of 5% simply be aware that this is slightly out of the ordinary. If they have policies that encourage a second person to own the ticket after the first then this is not a very strong indicator of any process issue. However, if tickets are changing hands due to poor delegation methods or uninformed volunteerism, we suggest this should be discussed and procedures improved.

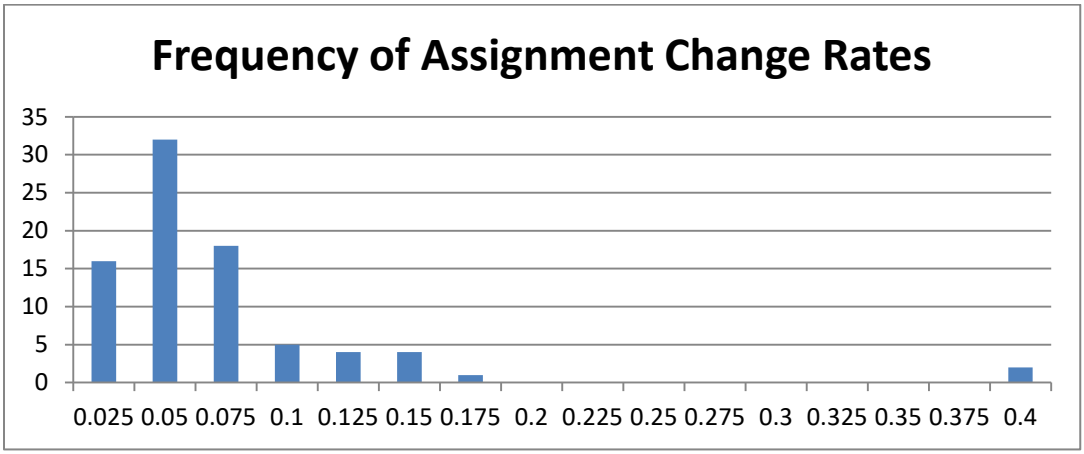


Figure 21 – The frequency of assignee tends to be below 10% in the Apache foundation. Teams above this amount may be deliberately asking two users to contribute but they might also need to discuss better assignment methodologies.

4.9.3 Process Indicators

We contend the previous group of measures are largely outcomes that result from all the choices made and unforeseen events beyond the control of the team. The following measures are more in direct control of the team. For instance, there are a few indicators that we argue show a team's interest in requirements quality. If every ticket is taken as if the text describing the need is flawless, wont some code changes occasionally be made in error, or at least, be controversial? We consider a change in requirements description, summary (title) and in many cases comments as a possible signal of requirements "quality assurance" or QA. The rates that these changes occur vary greatly among Apache teams, as shown in Figure 22 but we note eight of the 85 have the description change above 15% of their tickets. Do the teams that devote more energy to assuring a requirement is sound benefit in some way? We approach this question in section 4.9.6.

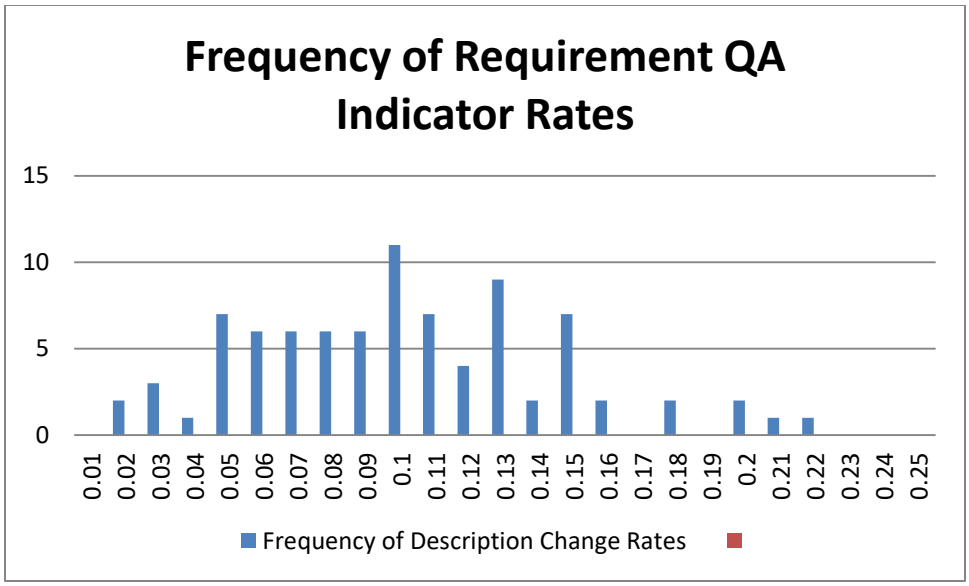


Figure 22 – Our two Requirements QA indicators show that a few teams update their requirements after the ticket is created but for most this is not common. Do the teams that devote more energy to assuring a requirement is sound benefit in some way?

A more comprehensive study on the processes OSSD teams utilize to distribute their work was addressed in our previous work but we take the approach of comparing projects in one specific way here. How often does a ticket include an event where a user assigns the ticket to themselves, a clear indicator of volunteerism. Although ~30% is the most common ratio of tickets subjected to self-assignment when utilizing 5% buckets, a few teams have well over half of their tickets depending on the selfless acts of volunteers. What are these teams doing differently and do they benefit from this high degree of volunteerism?

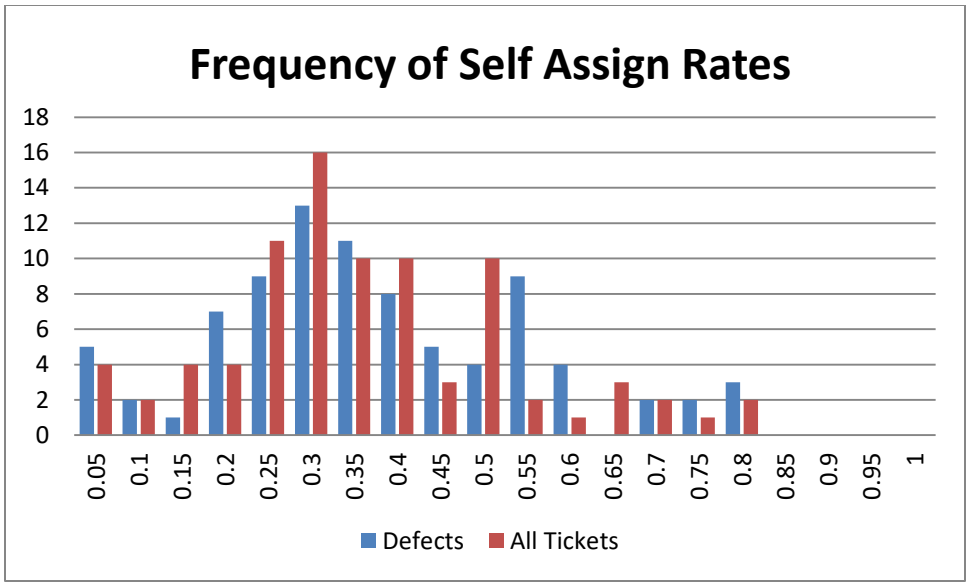


Figure 23 – The percentage of a project's tickets having been volunteered for varies greatly with a few having half or more of their tickets with a self-assign event.

By also isolating the tickets pertaining to defects in Figure 23, visually we see that the higher rates of self-assignment are just a little bit more common in defects. These findings led to many questions such as:

> ***How is workflow different for these teams that have half or more of their tickets subjected to volunteerism versus the teams where this only happens 10 or 20% of the time?***

We address this in section 4.9.7.

4.9.4 Conditional Outcomes and Process Correlations

Although we used our system's capability to query the SPMF produced association rules for these findings, the vast majority of the previous could have been established simply by querying the ratio of the number of tickets that had the event per project versus the number of tickets in that project. However, we were able to query from the database of association rule by seeking the pattern:

> **"Any Ticket is Created" ➔ (event in question) Equation 15**

When applied to only the tickets in that project, the confidence measure for a rule such as this represent how common the event in question is in that project. This is simply because the event

in question will always occur after a creation event (which is always the first event), if it occurs at all.[19]

However, in order to fully utilize the power of sequential pattern mining in this context we should also investigate conditional events. Given that occasional occurrence A (the antecedent) occurs, how often does the B (the consequent) occur? In many instances, we argue that this shows more about how a team reacts to important events through their process choices.

These relationships between antecedents and consequent events led us to many questions about whether some processes were at all influencing each other and could they be at all connected to our effectiveness measures. So we also set out to conduct numerous correlations studies. Effectively, we wanted to ask:

> ***If a team tends to allow X to occur more often than most projects, do they also tend to experience more or less of anti-pattern Y?***

4.9.5 Ticket Abandonment Management

If a ticket losses an assignee without an immediate replacement, how bright does that tickets future look? It turns out some teams recover from this indicator of abandonment much better than others. In our Figure 24 shows that the variance is astonishing. Thirty teams leave a ticket unfixed with more than 50% of these cases of abandonment and with 13 teams, it will happen over 80% of the time. Clearly on some teams, a ticket is most certainly doomed to a no-solution scenario if a person is removed as owner.

[19] To be clear, a second event in the same ticket does not increase the confidence measure.

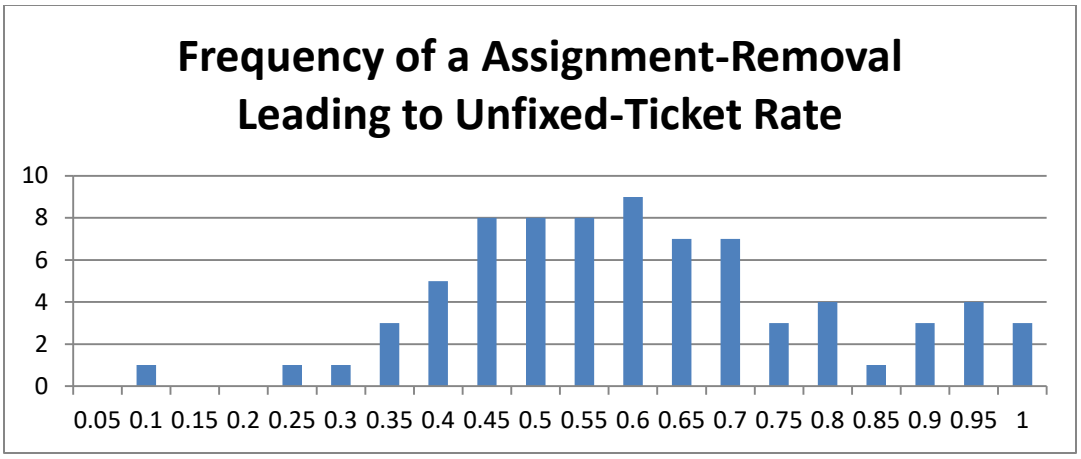


Figure 24 – The conditional rate of a ticket not being presented with a code change based on the event that a ticket loses an assignee; our indicator of abandonment.

It is important to remember that these figures include all the non-fix resolutions mentioned in the previous section so some of these might have been abandoned because they were duplicates or not worthy of some one's time. With that said, we still argue that the great variance indicates some teams may be handling the scenario of abandonment differently.

However, the frequency at which an abandoned ticket was not closed, and therefore a non-fix resolution was not determined yet, might be a more indicative of how a team recovers from ticket abandonment. We note that in 9 of the largest projects that had enough data to support this association rule, 60% or more of the abandoned tickets will *not* be completed or resolved as Figure 25 indicates.

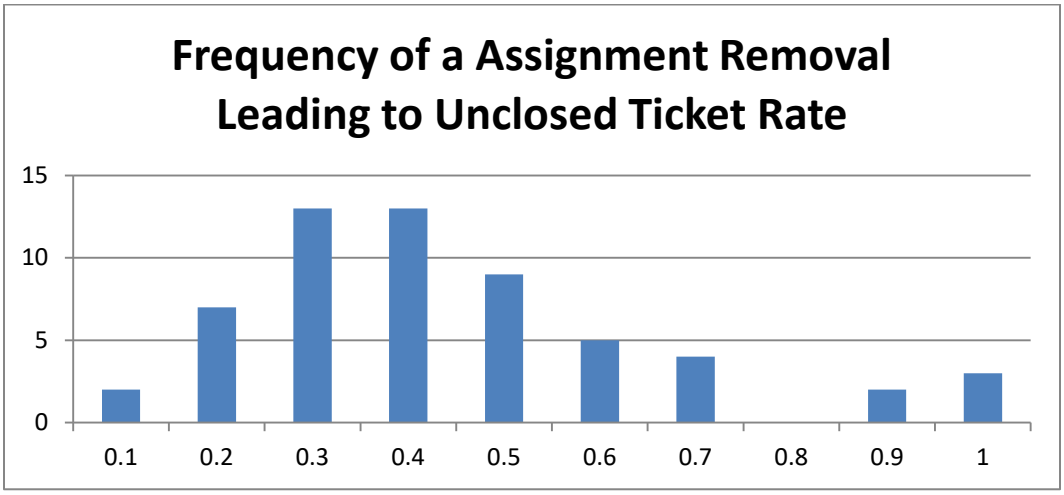


Figure 25 - Given that an assignee has been removed, on some teams the ticket remains unclosed only 30-40% of the time. But in five others, 80+% of these tickets that lose an assignee are not closed.

As 82 of the largest projects had enough data to give us strong association rules, we took that same sample and sought the correlation of ticket removal and our closure rates. We found a correlation coefficient of 0.716 for not-Fix and 0.412 for unClose. In contrast, there was virtually no relationship with an assignee changing. We depicted this in a scatter plot in Figure 25.

It's important to note that the varying confidence measures of the association rules depicted in Figure 25 and these correlations are describing slightly different relationships. As the amount of abandonment increases in a project, there is a moderate relationship with an increase in not closing and a rather strong relationship with not fixing tickets. Meanwhile the distribution association rules is, among other things, showing us that some teams are able to reduce the adverse effects of abandonment better than others.

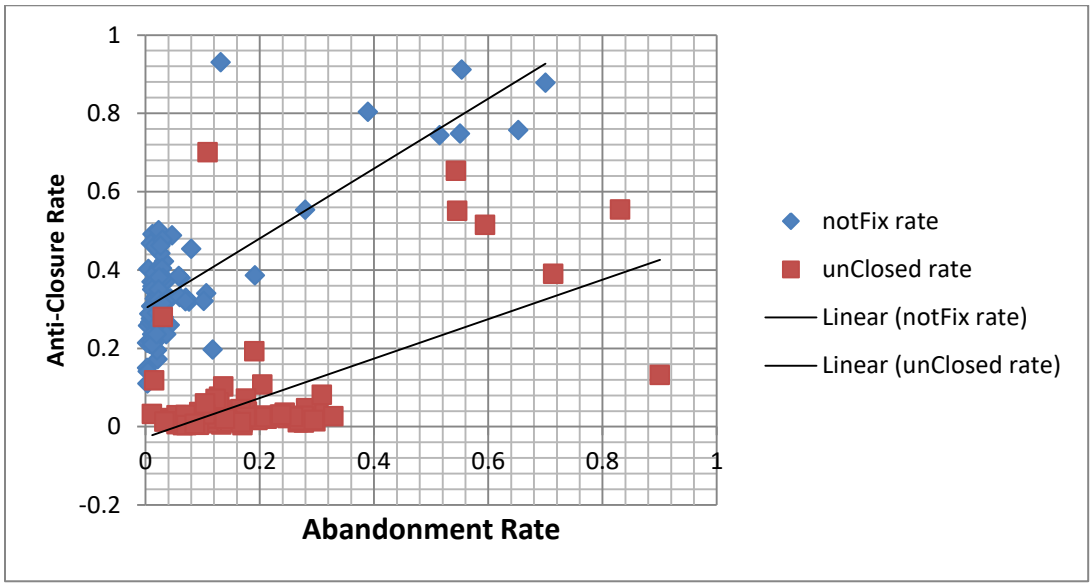


Figure 26 – Each of our 82 largest projects is depicted to show that the correlation between this form of abandonment and tickets not being given a solution is quite high at 0.72 and notable but moderate at 0.41 for unclosed rates.

4.9.6 Requirements Quality Assurance Impacts

As depicted in Figure 22, the degree to which teams alter their requirements, in both the ticket title and the ticket description vary quite a bit and we were curious if there were any relationships that might give guidance to OSSD teams in the future. Not surprisingly, if a team tends to alter the ticket description more than normal, they also tend to alter the title as these are correlated with a coefficient of 0.578. Commenting also seemed to encourage the description change as

they were correlated at a rate of 0.612. The connection to title changes and commenting was much smaller at a correlation coefficient of 0.267146. Perhaps comments are where the details are maneuvered but the main idea often remains unchanged. This is not a study on the evolution of requirements themselves but their workflow, however, future work might attempt to address how often a requirements update is simply a clarification vs a genuine change in need. These relationships draw from 78 of the largest projects that also had enough incidents of requirements changing to justify an association rule. Thus about seven projects did not have enough QA indicators to fit this evaluation.

We also set out to see if completion was higher for teams that collaborate enough to justify requirements changes. What we found interesting is that among our moderate size projects, those that had enough data to justify rules but were still below 4000 tickets (49 fit in this category), there are very weak correlations with changing the description and higher rates of closure (0.247366 for fixing and 0.179440 for closing). However, among the largest 29 projects that had enough requirements QA, there is a moderate negative impact on our completion measures. As shown in Figure 27, the correlation with summary change and not-fixing rate is 0.542344 although with a summary change it is 0.447834. UnClose rate is less impacted with a correlation to description change as 0.375506 and 0.562615 for summary change. Not only the strength but direction of these relationships surprised us enough to include them here. Could requirements undergoing debate be less attractive to volunteers thus reducing the closure rate in these tickets of the largest projects?

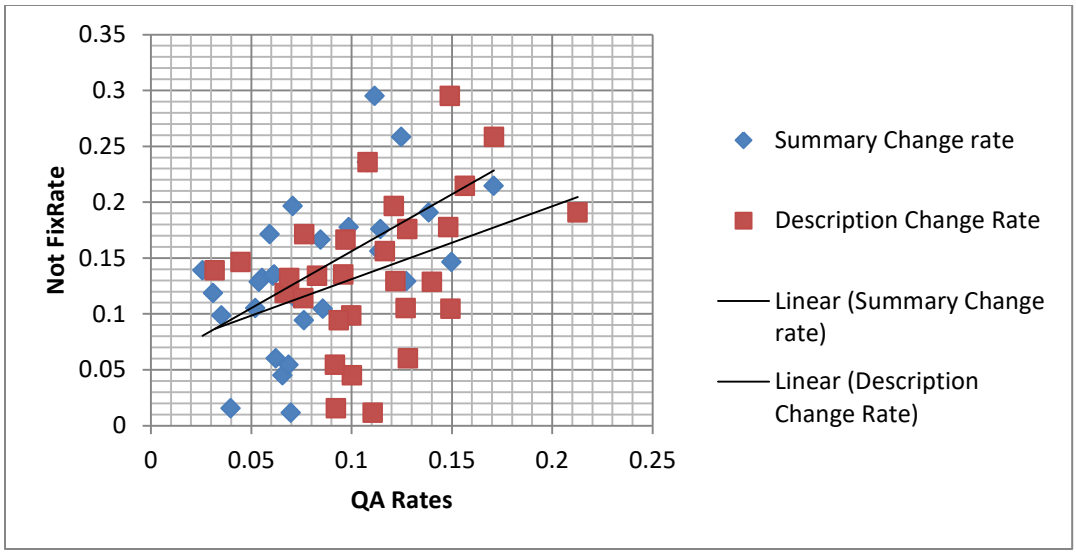


Figure 27 – Among the largest projects there is some indications that changing the requirements reduces the probability that a ticket will become fixed.

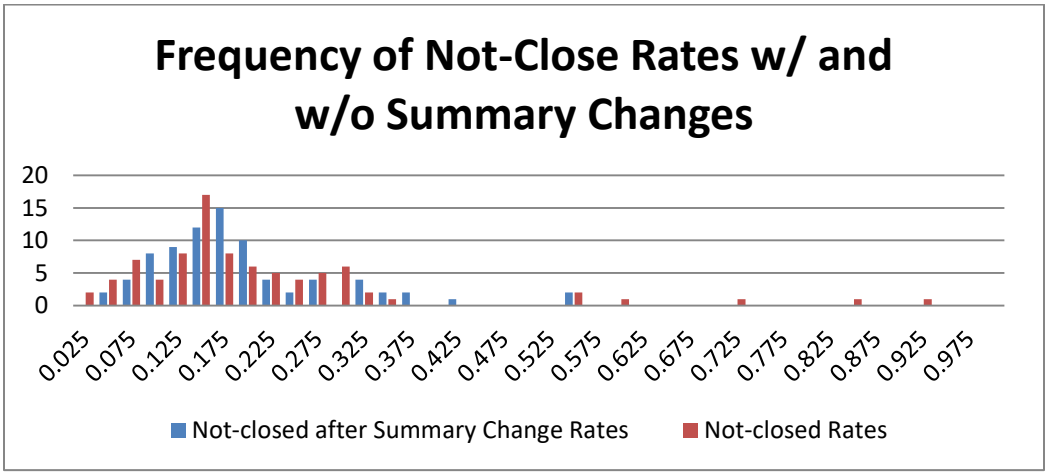


Figure 28 – The conditional closure rates given that a summary change has occurred against the normal closure rates. We see that there are some serious outliers in notClose rate but we also observe that closure rates after summary change tend to be higher, if we exclude the outliers.

To gain deeper insight, we also wanted to get an idea of the distribution of closure rates, given that a summary change has occurred. In the Apache population as a whole, once a summary change has occurred 21 projects have a track record of not closing their tickets at a higher rate than the average 20%. But without this condition of requirements QA, there are 29 projects in this category.

In the process of finding these relationships, we investigated many more. We noticed almost every measure is almost completely independent of ticket count but not our QA indicators. Ticket

count had a correlation with title change of 0.560193 and description change of 0.552188. Is it possible that projects have a harder time evolving if there aren't some product experts willing to clarify needs to the developers?

4.9.7 Volunteerism and Completion

It is well known that OSSD projects are driven by volunteers. As depicted in Figure 23, the degree to which a team depends on the action of such volunteers to assign work to themselves varies quite a bit. We were curious as to how this influences the completion rates. In Figure 29, we depict the -0.426 correlation between self-assignment rate and not-fixing rate. For unclosed ticket rate, it was just slightly stronger at -0.486. What was surprising to us is that the largest 29 projects had a weaker correlation between this task distribution pattern and unclosed rate at -0.323 and virtually no relationship (-0.021 coefficient) with not-fixing. Perhaps these very large projects, some with over 10k tickets, be dependent on volunteers but still find success with higher delegation rates? Meanwhile the projects large enough to give us data strong enough to justify association rules but still remain below 4000 tickets had a stronger (-0.515 with not-fix rate and -0.521 for unclosed rate) correlation with the self-assign rate. Perhaps that is major difference between the moderate sized and large sized projects: The largest projects are still utilizing volunteers at a massive rate but they do not depend on them to distribute tasks nearly as much. Perhaps a modest degree of coordination is healthy for large OSSD projects? We suggest that the largest projects may benefit from some form of delegation and suggest this is a valuable topic for future study.

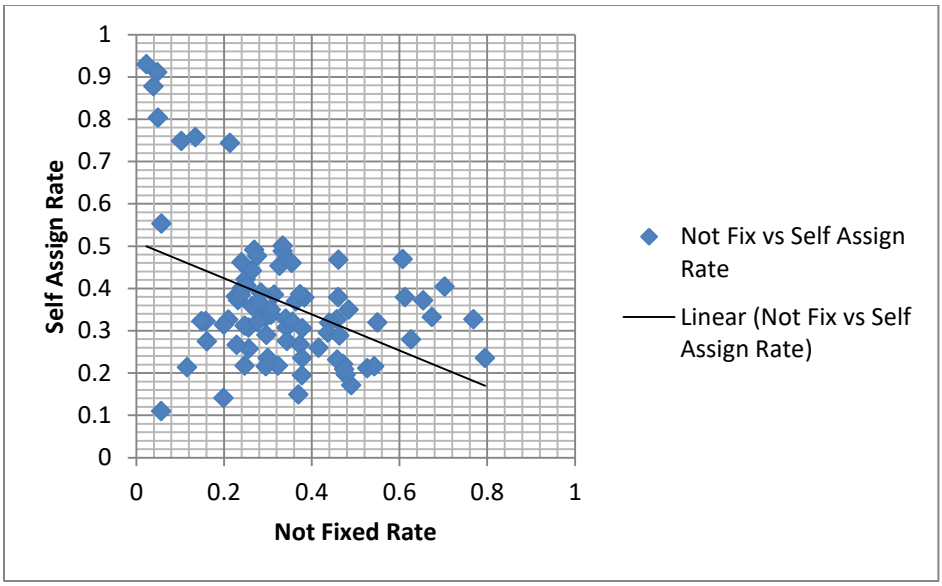


Figure 29 – As the percentage of tickets that have any interaction with a volunteer, designated by a self-assignment event, decreases, so does the percentage of tickets not being fixed in this moderate correlation.

4.9.8 The Model Apache Teams

We have been clear that some of our effectiveness measurements are subjective. We argue that they represent anti-patterns that a team wants to avoid. However, there are scenarios that make these events completely understandable. With that being said, we wanted to know if there were a handful of these well known Apache projects that tend to score significantly better in more than a couple of our effectiveness measures and therefore can be said to represent a model of what other teams should aspire to. If some of these model teams also happen to have some processes in common, or at least many of them tend to use certain processes more than the general public, is that also something the OSSD community should consider?

4.9.9 Requirements QA for Leading Teams

The projects that rank in the top 20 percentile for both of our closure measures seem to rank very low in requirements QA efforts, with some exceptions. Only as a means to approximate where each team ranks in a given process measure, we are showing the average ranking of these top performing eight teams in the two requirements QA indicators. As there are 85 in the entire pool, an average ranking of 56.3 indicates this group is not experiencing as much debate in requirements. In fact, with the exception of Apache Quid, this group rarely changes a ticket's

summary. Although this anecdotal, we argue this supports our previous finding in Figure 17 that connects requirements change and completion rates.

Not-fix Rate	UnClosed Rate	Project Name	Description Change Rate	Description Change Rank	Summary Change Rate	Summary Change Rank
19.62%	1.55%	ASF Infrastructure	9.23%	47	3.98%	69
20.90%	3.33%	Apache ServiceMix	5.33%	71	4.14%	67
17.13%	4.48%	Apache Camel	10.02%	37	6.56%	45
21.09%	6.01%	Apache CXF	12.81%	18	6.23%	48
10.97%	6.56%	Apache Ambari	10.62%	35	5.29%	60
14.91%	7.19%	Apache ServiceMix 4	4.18%	78	4.18%	66
14.04%	9.35%	Apache UIMA	5.63%	69	5.02%	63
23.52%	9.42%	Apache Qpid	9.38%	46	7.63%	32
		Ave Rank		50.1		56.3

Table 14 – The projects that rank in the top 20 percentile for both of our closure measures seem to rank very low in requirements QA, with some exceptions. Although this anecdotal, we argue this supports our previous finding in Figure 27 that connects requirements change and completion rates.

The ability of a team to complete the tasks presented is our primary indicator of effectiveness and there seems to be a few strong indicators of how that status might be accomplished. Among the leading teams, we observe how often tickets are volunteered for and contrast these findings with how often a ticket is completely driven by one person. We argue both signal a team's ability to utilize volunteerism.

We note in Table 15 that with the exception of Apache Quid once again, we observe that all of our completion leaders are extremes in these two forms of volunteerism. Apache Ambari, ServiceMix 4, and UIMA are all dominated by participants that write their own tickets and complete them much more so than other leading teams. Meanwhile, Apache Infastructure, Camel, CXF, and ServiceMix have historically shown success with participants volunteering for work that others have suggested at a rate far superior to most as indicated by the self-assignment indicator. Of course, we are presuming that the teams on ServiceMix 4 and ServiceMix have some history in common. What changed in their leadership to make requirements more self-driven in the newer version? Interestingly, Apache quid is rather average in both forms of volunteerism yet has a low unClosed rate.

Not-fix Rate	UnClosed Rate	Project Name	Single Participant Rate	Single Participant Rank	Self-Assignment Rate	Self-Assignment Rank	Assignement Change Rate	Assignement Change Rank	Assignment Removal Rate	Assignment Removal Rank
19.62%	1.55%	ASF Infrastructure	2.09%	**75**	47.74%	14	2.87%	23	11.80%	76
20.90%	3.33%	Apache ServiceMix	15.14%	54	47.38%	15	2.57%	20	1.19%	19
17.13%	4.48%	Apache Camel	23.89%	30	48.97%	12	3.57%	37	2.14%	40
21.09%	6.01%	Apache CXF	8.39%	63	52.66%	11	2.56%	17	0.74%	8
10.97%	6.56%	Apache Ambari	64.81%	**2**	5.69%	81	1.49%	9	0.34%	**3**
14.91%	7.19%	Apache ServiceMix 4	44.26%	**8**	36.96%	31	3.49%	33	0.32%	**2**
14.04%	9.35%	Apache UIMA	65.90%	**1**	19.91%	73	5.39%	56	0.39%	**4**
23.52%	9.42%	Apache Qpid	18.75%	46	29.94%	45	17.20%	80	3.68%	61
		Ave Rank		34.9		35.3		34.4		26.6

Table 15 - The projects that rank in the top 20 percentile for both of our closure measures have either an exceptional percentage of tickets written and accomplished by one person or they have a rather high percentage of tickets volunteered for, with the exception of Apache quid which is rather average in both areas. Ticket Abandonment was also very low for many in this group.

Our final insights were with regard to task distribution methods in the form of tossing and abandonment indicators. Of course, this is an assignment change or assignment removal as explained in section 4.9.2. Although this group ranked high in avoiding[20] these anti-patterns with an average of 34.4 as opposed to the expected 42.5, we note Apache Quid is once again the process pattern anomaly. Aside form UIMA and Quid, this group have very little tossing. Similarly, Apache Infrastructure and Quid are the notable exceptions to an extremely low rate of abandonment rate. What could be special about these teams that their participants are so conscientious?

So is there one formula for becoming a highly productive OSS team? Absolutely not. But we provide these process profiles of some of the more effective team to provoke discussion among the leaders of OSSD just as we did with effectiveness measures and process indicators. This brief exposé on what some of the completion leaders have in common is quite anecdotal. Although we bring some insights into what they have in common, we bring to light more questions than answers.

4.10 What Would Other Researchers Think?

Through our demonstrations of Project analytics and in particular Project Process Analytics, we hope we've made a strong case that there is tremendous research potential in this data to create

[20] As tossing and abandonment are considered anti-patterns in this research effort, we rank those that avoid them more as "higher" rank. All teams would want to aspire to be #1 in our paradigm!

more useful software analytics. But where might these analytics be used in other forms of research?

4.10.1 Modeling

The great challenge of modeling is to be able to calibrate the relationships between different entities being modeled and then validate the results produced by said model. When it comes to calibrating and validating a model of a software team, it's hard to know where to begin. However, empirical data of this kind is capable of providing metrics that would calibrate simulation as (Münch, 2003) argued. We highlight the following measures that would be difficult to obtain by other means such as:

- Requirements arrival patterns
- Workflow Patterns
- Ticket re-open rates

The wealth of analytics we were able to acquire just from the user behaviors is only the beginning and all of them have the potential to both calibrate and validate a model on OSS projects

4.10.2 Organizational Behavior Studies

We believe a great source of value in this requirements repository data on OSS projects is that since, in most cases, all the work is done by volunteers, the impact of any choices in management strategy, including policy and process, are exaggerated relative to a team that is paid to make contributions.

We believe investigations into these deliberate choices of the highly successful projects and comparing them to the less successful projects from a standpoint of motivation would bring great insight. Although this might come in many forms, we argue that a project that is able to recruit but then maintain a high level of heavy contributors is doing something right. And simply, because empirical evidence is so difficult to obtain in software process improvements studies; we argue that researchers in organizational behavior might also find this data valuable.

4.11 Chapter Conclusion

This effort was both the exploration of a dataset with great research potential and a demonstration of the approach that gives this data the greatest insight: process mining.

Initially, we explained how we obtained and verified our data using various forms of web retrieval including web crawling. From there, we saw great potential in comparing projects to possibly seek best practices. First we showed how much they differ in terms of size and age. We wanted a statistically valid way to compare and decided that focusing on the processes that their tickets undergo had great research potential.

Then we explained how we created a pattern mining platform that would allow for repeated experiments on a large quantity of event logs in a highly scalable way. This module is independent of our requirements repository capture and analysis system and will someday be made available to others as it has great potential for future process mining studies.

Our exploration of processes in Apache brought many insights. First was simply that projects vary greatly in their ability to complete work. They also vary a great deal in terms of task distribution indicators. Some projects suffer from abandonment and others from assignment change (tossing) a great deal more than others. What was remarkable is that, of the tickets that lost their owner (abandonment) some teams are still able to complete the majority of these tasks while others end up neglecting nearly all of such tickets. We showcased other process choices and how they differ between teams. Our areas of focus were requirements changes, what we refer to as Requirements Quality Assurance, and the prevalence of volunteerism in the form of participants volunteering for tickets that others have written.

In taking this approach, we were able to make the case that both team effectiveness in terms of outcomes and the choices they make that might influence these outcomes vary so much there may be a opportunities to establish relationships between the process choices and the results. Although we were not surprised that teams with higher abandonment rates also had more tickets without a solution presented to them, but the strength of the relationship at a coefficient of 0.716 was remarkable in our view.

The efforts toward refining the requirements turned out to have a modestly negative association with completion measures. This was especially true with the larger teams. The less they changed the requirements, the more likely someone will complete the task. We were also able to show a modest correlation between the ratio of tickets involving a self-assigning volunteer and the completion measures. What was interesting is that this was less true for are largest 29 projects and we suggest further study could support the use of delegation in some of the most prominent projects.

Anecdotally, we then showcased the eight teams that rate the highest in both of our completion measures. Interestingly many of these rated low in our indicators of requirements QA and ticket abandonment just as our correlations suggested, but not all of them. Nearly all of them (with one exception) were very high in the category of tickets written and completed by the same person or rather high in the self-assigning indicator. We associate both of these as strong indicators of volunteerism.

In conclusion, it is extremely difficult to prove which choices are best for any one team because there are so many factors involved. However, by using project process analytics to identify that some process choices, especially those sometimes associated with negative productivity, are more common in one project than with other OSSD projects we argue that in itself is valuable to the community in so far as we hope it will provoke a discussion about potentially improving such processes.

5 FUTURE WORK

5.1 Simulation Modeling

Very early in this research effort, simulation modeling was the primary research tool planned to address workflow and this dataset was intended to verify and validate said model. Both the parameters of the model and the outcomes would be compared to the real world dynamics at Apache and we would hope to contribute to research efforts addressing how different process choices impact outcomes. Now that so many processes were studied in depth during these three investigations, a discrete event simulation would give insights that pattern mining alone would not.

5.2 Pattern Mining Education

One of the unexpected benefits of our two investigations utilizing sequential pattern mining was a comprehensive understanding of the factors that lead to pattern mining efforts that are too computational complex for a typical laptop to undertake. We are aware of literature addressing different aspects of this topic of computational complexity but not one that demonstrates all of the factors that we have become aware of and while all using powerful teaching tools such as simulations, equations and demonstrations. We have developed a detailed plan for our next research effort that, leveraging our pattern mining platform, would allow both industry practitioners and students to gain this same level of appreciation for pattern mining such that their data mining skills will be significantly improved. The research plan is as follows.

The common objective in an Association Rule Mining effort is to seek the most frequent patterns present in a data set and many of the algorithms available take advantage of this objective by removing from consideration events and patterns that are not frequent when seeking those that are in order to contain (what we call) runaway computational complexity. However, what if the events of interest are not frequent? What if there is an expensive problem or highly profitable occurrence that needs to be studied but it is infrequent? Fortunately, there are many approaches available to researchers aside from the obvious solution of allocating a tremendous amount of

computing power. However, in order to properly understand these approaches, we need to thoroughly appreciate all the factors that impact the computational complexity of any one given association rule mining effort.

So we would start this future effort by discussing the steps that many common association rule mining algorithms share and some of the data structures involved. After describing the pattern mining tool SPMF we would explain how we augmented it to give more information on the memory and processing demands of any one given run. Then to gain a deeper understanding, we would use simulations to demonstrate how different algorithm parameters significantly influence the memory needs of a given run. We would also show how the iteration counts of the algorithm steps are impacted by these parameters. We also plan to create artificial data sets that have specific attributes to show how the characteristics of data sets also impact the computational complexity of a pattern mining effort.

Finally, we would use this same dataset from Apache to demonstrate these methods by examining two anti-patterns of interest and the many possibly influential events (many infrequent themselves) that we would want to investigate as possible influences to the anti-patterns of interest. We would finish with some suggestions for how these discovered influences might lead to further studies that may themselves lead to policy changes to reduce these anti-patterns in the open source community.

6 Dissertation Conclusion

We set out to make significant contributions to the area of workflow in software teams by conducting three comprehensive investigations of the Apache Software Foundation.

We supported findings that the longer a person is a part of a software development team, the more likely the requirements they pose will be respected by others. We also determined that the first requirements a user proposes are more likely to be a defect than a new feature in the open source community. A large portion of the new feature requirements in Apache are written by a very elite group comprised of less than 1% of the population.

We also investigated how these requirements are distributed to the participants to be implemented. The participants that create the tickets are often those with the best track records for determining who should participate in that ticket. We also gained some insights into the advantages of volunteerism. In addition, we found a disadvantage of volunteerism in that when a participant claims a ticket but postpones the work involved; it is quite likely to create a significant delay. We make recommendations in hopes that fewer tickets will be claimed but not implemented in a timely manner so that OSSD teams will become more effective.

Lastly, we used process mining techniques to look into a variety of process choices and how they may be influencing the outcomes of OSSD projects. Strikingly, there is a negative association between how often a team updates the specifics of a requirement and how often requirements remain unfinished. Some valuable future work might verify that debating requirements sometimes discourages volunteerism. We also verified that higher overall volunteerism indicators reduce incidents of incomplete work but what was surprising is that this correlation is stronger if we exclude the very large projects. In future work, we would like to investigate if many of the largest projects have become so in part due to some more traditional leadership roles being in place and work actually being delegated in some instances.

We also had completely unplanned contributions that we would like to emphasize. The issues surrounding computational complexity in pattern mining was not a research objective but we discovered it to be an important topic and skill of process mining. Although our findings in this area are not appropriate for a dissertation on OSSD workflow, we have every intension of

delivering them to the data science community so that others may gain from our findings. Also, our pattern mining platform started small and expanded such that any future pattern mining efforts will take days instead of months, once the abstractions are established. We did not set out to create an open source tools for highly scalable pattern mining efforts and, as of this writing, our platform is not ready for public consumption. However, with a reasonable amount of effort, our platform could be delivered to the world such that others may benefit. Lastly, our greatest realization is that the abstractions are really where the effort should lie during a pattern mining effort and we hope to contribute to this area of data mining in the future.

We are proud of our contributions as they may lead to improvements in workflow within many OSSD teams, but we present them with great humility because, for any one that has been a long time participant in any software development effort will tell you, there is no one formula for obtaining optimal workflow. There are simply best practices and making any contribution to the industry conversation on best practices is a welcome contribution at that.

7 References

Atlassian. (n.d.). What is an Issue. Retrieved from https://confluence.atlassian.com/display/JIRA/What+is+an+Issue

Bai, X., Zhang, H., & Huang, L. (2011). Empirical Research in Software Process Modeling: A Systematic Literature Review. *2011 International Symposium on Empirical Software Engineering and Measurement*, 339–342. http://doi.org/10.1109/ESEM.2011.43

Baker, R. S. J. d. (2010). Data Mining - Chapter 1 : An Introduction to Data Mining, 112–118. http://doi.org/http://dx.doi.org/10.1016/B978-0-08-044894-7.01318-X

Bantelay, F., Zanjani, M. B., & Kagdi, H. (2013). Comparing and combining evolutionary couplings from interactions and commits. *2013 20th Working Conference on Reverse Engineering (WCRE)*, 311–320. http://doi.org/10.1109/WCRE.2013.6671306

Baysal, O., Kononenko, O., Holmes, R., & Godfrey, M. W. (2016). Investigating technical and non-technical factors influencing modern code review. *Empirical Software Engineering*, *21*(3), 932–959. http://doi.org/10.1007/s10664-015-9366-8

Baysal, O., & Malton, A. J. (2007). Correlating Social Interactions to Release History during Software Evolution. *Fourth International Workshop on Mining Software Repositories (MSR'07:ICSE Workshops 2007)*, 7–7. http://doi.org/10.1109/MSR.2007.4

Benbasat, I., Goldstein, D. K., & Mead, M. (1987). The Case Research Strategy in Studies of Information Systems. *MIS Quarterly*, *11*(3), 369. http://doi.org/10.2307/248684

Berlingerio, M., Pinelli, F., Nanni, M., & Giannotti, F. (2009). Temporal mining for interactive workflow data analysis. *Proceedings of the 15th ACM International Conference on Knowledge Discovery and Data Mining (2009)*, *15*(4), 109. http://doi.org/10.1145/1557019.1557038

Bird, C., Gourley, A., Devanbu, P., & Gertz, M. (2006). Mining Email Social Networks ∗ Categories and Subject Descriptors. *Msr*. http://doi.org/10.1145/1137983.1138016

Buse, R. P. L., & Zimmermann, T. (2010). Analytics for software development. *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research - FoSER '10*, 77. http://doi.org/10.1145/1882362.1882379

Buse, R. P. L., & Zimmermann, T. (2012). Information needs for software development analytics. *Proceedings - International Conference on Software Engineering*, 987–996. http://doi.org/10.1109/ICSE.2012.6227122

Canfora, G., & Cerulo, L. (2005). Impact Analysis by Mining Software and Change Request Repositories. *11th IEEE International Software Metrics Symposium (METRICS'05)*, (Metrics), 29–29. http://doi.org/10.1109/METRICS.2005.28

Cerulo, L., Ceccarelli, M., Di Penta, M., & Canfora, G. (2013). A Hidden Markov Model to detect coded information islands in free text. *2013 IEEE 13th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, 157–166. http://doi.org/10.1109/SCAM.2013.6648197

Checking, C. (2016). Software Development Process Mining: Discovery, Conformance Checking and Enhancement, 254–259. http://doi.org/10.1109/QUATIC.2016.51

Chen, L., Wang, X., & Liu, C. (2011). An approach to improving bug assignment with bug tossing graphs and bug similarities. *Journal of Software*, *6*(3), 421–427. http://doi.org/10.4304/jsw.6.3.421-427

Choetkiertikul, M., Dam, H. K., Tran, T., & Ghose, A. (2015). Characterization and prediction of issue-related risks in software projects. *IEEE International Working Conference on Mining Software Repositories*, *2015–Augus*, 280–291. http://doi.org/10.1109/MSR.2015.33

Chudoba, R., Sadílek, V., Rypl, R., & Vorechovsky, M. (2013). Using Python for scientific computing:

Efficient and flexible evaluation of the statistical characteristics of functions with multivariate random inputs. *Computer Physics Communications*, *184*(2), 414–427. http://doi.org/10.1016/j.cpc.2012.08.021

Comino, S., Manenti, F. M., & Parisi, M. L. (2007). From planning to mature: On the success of open source projects. *Research Policy*, *36*, 1575–1586. http://doi.org/10.1016/j.respol.2007.08.003

Crowston, K., Li, Q., Wei, K., Eseryel, U. Y., & Howison, J. (2007). Self-organization of teams for free/libre open source software development. *Information and Software Technology*, *49*(6), 564–575. http://doi.org/10.1016/j.infsof.2007.02.004

Easton, G. (2010). Critical realism in case study research. *Industrial Marketing Management*, *39*(1), 118–128. http://doi.org/10.1016/j.indmarman.2008.06.004

Fielding, R. T., & Taylor, R. N. (2000). Principled design of the modern Web architecture. *Proceedings of the 2000 International Conference on Software Engineering. ICSE 2000 the New Millennium*, *2*(2), 115–150. http://doi.org/10.1109/ICSE.2000.870431

Gala-Pérez, S. J. (2013). Intensive Metrics for the Study of the Evolution of Open Source Projects, 159–168.

Georgios, G., & Bacchelli, A. (2014). Work Practices and Challenges in Pull--Based Development: The Contributor`s Perspective. *Proceedings of the 38th International Conference on Software Engineering - ICSE '16*, (ICIS--R15001), 285–296. http://doi.org/10.1109/ICSE.2015.55

Giuri, P., Ploner, M., Rullani, F., & Torrisi, S. (2010). Skills, division of labor and performance in collective inventions: Evidence from open source software. *International Journal of Industrial Organization*, *28*(1), 54–68. http://doi.org/10.1016/j.ijindorg.2009.07.004

Goeminne, M. (2014). Understanding the evolution of socio-technical aspects in open source ecosystems. *2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering, CSMR-WCRE 2014 - Proceedings*, 473–476. http://doi.org/10.1109/CSMR-WCRE.2014.6747221

Goeminne, M., & Mens, T. (2011). Evidence for the Pareto principle in open source software activity. *CEUR Workshop Proceedings*, *708*, 74–82.

Gousios, G. (2008). Measuring Developer Contribution from Software Repository Data. *Management*, 3–6.

Gousios, G., & Spinellis, D. (2014). Conducting quantitative software engineering studies with Alitheia Core. *Empirical Software Engineering*, *19*, 885–925. http://doi.org/10.1007/s10664-013-9242-3

Grandell, L., Peltomäki, M., Back, R. J., & Salakoski, T. (2006). Why complicate things? Introducing programming in high school using Python. *Conferences in Research and Practice in Information Technology Series*, *52*, 71–80.

Guo, P. J., Zimmermann, T., Nagappan, N., & Murphy, B. (2010). Characterizing and predicting which bugs get fixed: an empirical study of Microsoft Windows. *2010 ACM/IEEE 32nd International Conference on Software Engineering*, *1*, 495–504. http://doi.org/10.1145/1806799.1806871

Guo, P. J., Zimmermann, T., Nagappan, N., & Murphy, B. (2011). "Not my bug!" and other reasons for software bug report reassignments. *Proceedings of the ACM 2011 Conference on Computer Supported Cooperative Work - CSCW '11*, 395. http://doi.org/10.1145/1958824.1958887

Han, J., Kamber, M., & Pei, J. (2012a). 6 - Mining Frequent Patterns, Associations, and Correlations: Basic Concepts and Methods. In J. H. Kamber & J. Pei (Eds.), *Data Mining (Third Edition)* (Third Edit, pp. 243–278). Boston: Morgan Kaufmann. http://doi.org/http://dx.doi.org/10.1016/B978-0-12-381479-1.00006-X

Han, J., Kamber, M., & Pei, J. (2012b). 7 - Advanced Pattern Mining. In J. H. Kamber & J. Pei (Eds.), *Data Mining (Third Edition)* (Third Edit, pp. 279–325). Boston: Morgan Kaufmann.

http://doi.org/http://dx.doi.org/10.1016/B978-0-12-381479-1.00007-1

Han, J., Kamber, M., & Pei, J. (2012c). *Data Mining*. *Data Mining*. Elsevier. http://doi.org/10.1016/B978-0-12-381479-1.00002-2

Hayashi, H., Ihara, A., Monden, A., & Matsumoto, K. (2013). Why is collaboration needed in OSS projects? a case study of eclipse project. *Proceedings of the 2013 International Workshop on Social Software Engineering - SSE 2013*, 17–20. http://doi.org/10.1145/2501535.2501539

Herraiz, I., Gonzalez-Barahona, J. M., Robles, G., & German, D. M. (2007). On the prediction of the evolution of libre software projects. *IEEE International Conference on Software Maintenance, ICSM*, 405–414. http://doi.org/10.1109/ICSM.2007.4362653

Herzig, K., Just, S., & Zeller, A. (2013). It's not a bug, it's a feature: How misclassification impacts bug prediction. *Proceedings - International Conference on Software Engineering*, 392–401. http://doi.org/10.1109/ICSE.2013.6606585

Huang, Z., & Kumar, A. (2007). A Study of Process Mining: Quality and Accuracy Tradeoffs. *… of the 4th Workshop on Business Process …*, 1–38. Retrieved from http://www.personal.psu.edu/axk41/mining-report08.pdf

Jensen, C., & Scacchi, W. (2007). Role migration and advancement processes in OSSD projects: A comparative case study. *Proceedings - International Conference on Software Engineering*, 364–373. http://doi.org/10.1109/ICSE.2007.74

Jensen, J. L., & Rodgers, R. (2001). Cumulating the Intellectual Gold of Case Study Research. *Public Administration Review*, *61*(2), 235–246. http://doi.org/10.1111/0033-3352.00025

Jeong, G., Kim, S., & Zimmermann, T. (2009). Improving bug triage with bug tossing graphs. *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering on European Software Engineering Conference and Foundations of Software Engineering Symposium - E*, 111. http://doi.org/10.1145/1595696.1595715

Knauss, E., Damian, D., Poo-caamaño, G., Cleland-huang, J., & Victoria, B. C. (2012). Detecting and Classifying Patterns of Requirements Clarifications. *2012 20th IEEE International Requirements Engineering Conference (RE)*, 251–260.

Krzysztof J. Cios, Roman W. Swiniarski, Witold Pedrycz, L. A. K. (n.d.). What is Data Mining? In *Data Mining: A Knowledge Discovery Approach* (pp. 3–7).

Lamkanfi, A., Demeyer, S., Giger, E., & Goethals, B. (2010). Predicting the severity of a reported bug. *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*, 1–10. http://doi.org/10.1109/MSR.2010.5463284

Lee, A. S. (1989). A Scientific Methodology for MIS Case Studies. *MIS Quarterly*, *13*(1), 33–50. http://doi.org/10.1017/CBO9781107415324.004

Li, C., Reichert, M., & Wombacher, A. (2011). Mining business process variants: Challenges, scenarios, algorithms. *Data and Knowledge Engineering*, *70*(5), 409–434. http://doi.org/10.1016/j.datak.2011.01.005

Lou, J.-G., Lin, Q., Ding, R., Fu, Q., Zhang, D., & Xie, T. (2013). Software analytics for incident management of online services: An experience report. *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 475–485. http://doi.org/10.1109/ASE.2013.6693105

Lutz, M. (2007). *Learning Python*. http://doi.org/10.1016/0019-1035(89)90077-8

Marlow, J., Dabbish, L., & Herbsleb, J. (2013). Impression Formation in Online Peer Production : Activity Traces and Personal Profiles in GitHub. *16th ACM Conference on Computer Supported Cooperative Work*, 117–128. http://doi.org/10.1145/2441776.2441792

Matter, D., Kuhn, A., & Nierstrasz, O. (2009). Assigning bug reports using a vocabulary-based expertise model of developers. *Proceedings of the 2009 6th IEEE International Working Conference on Mining Software Repositories, MSR 2009*, 131–140. http://doi.org/10.1109/MSR.2009.5069491

McKinney, W. (2012). *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython* (1st ed.). O'Reilly Media. Retrieved from http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/1449319793

Mockus, A., Fielding, R. T., & Herbsleb, J. D. (2002a). Two case studies of open source software development: Apache and Mozilla. *ACM Trans. Softw. Eng. Methodol.*, *11*(3), 309–346. http://doi.org/10.1145/567793.567795

Mockus, A., Fielding, R. T., & Herbsleb, J. D. (2002b). Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *11*(3), 309–346. http://doi.org/10.1145/567793.567795

Münch, J. (2003). Using Empirical Knowledge from Replicated Experiments for Software Process Simulation : A Practical Example. *Empirical Software Engineering*.

Nilsen, J. K. (2007). MontePython: Implementing Quantum Monte Carlo using Python. *Computer Physics Communications*, *177*(10), 799–814. http://doi.org/10.1016/j.cpc.2007.06.013

O'Mahony, S. (2003). Guarding the commons: How community managed software projects protect their work. *Research Policy*, *32*(7), 1179–1198. http://doi.org/10.1016/S0048-7333(03)00048-9

Pinto, G., Steinmacher, I., & Gerosa, M. A. (2016). More Common Than You Think: An In-depth Study of Casual Contributors. *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, *1*(1), 112–123. http://doi.org/10.1109/SANER.2016.68

Radtke, N. P., Janssen, M. a, & Collofello, J. S. (2009). An Agent-based Model of FLOss Projects What Makes Free / Libre Open source software (FLOss) Projects successful ? *International Journal*, *1*(June), 1–13.

Rebuge, ??lvaro, & Ferreira, D. R. (2012). Business process analysis in healthcare environments: A methodology based on process mining. *Information Systems*, *37*(2), 99–116. http://doi.org/10.1016/j.is.2011.01.003

Roberts, J. a., Hann, I.-H., & Slaughter, S. a. (2006). Understanding the Motivations, Participation, and Performance of Open Source Software Developers: A Longitudinal Study of the Apache Projects. *Management Science*, *52*(7), 984–999. http://doi.org/10.1287/mnsc.1060.0554

Schugerl, P., Rilling, J., & Charland, P. (2008). Mining Bug Repositories--A Quality Assessment. *2008 International Conference on Computational Intelligence for Modelling Control & Automation*, 1105–1110. http://doi.org/10.1109/CIMCA.2008.63

Schur, M., Roth, A., & Zeller, A. (2015). Mining Workflow Models from Web Applications. *IEEE Transactions on Software Engineering*, *5589*(MAY), 1–1. http://doi.org/10.1109/TSE.2015.2461542

Shao, Q., Chen, Y., Tao, S., Yan, X., & Anerousis, N. (2008). Efficient ticket routing by resolution sequence mining. *Proceeding of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD 08*, 605. http://doi.org/10.1145/1401890.1401964

Shibuya, B., & Tamai, T. (2009). Understanding the process of participating in open source communities. *2009 ICSE Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development*, 1–6. http://doi.org/10.1109/FLOSS.2009.5071352

Silva, R., Zhang, J., & Shanahan, J. G. (2005). Probabilistic workflow mining. *Knowledge Discovery in Data Mining*, 275–284. http://doi.org/10.1145/1081870.1081903

Steinmacher, I., Conte, T., Gerosa, M. A., & Redmiles, D. (2015). Social Barriers Faced by Newcomers Placing Their First Contribution in Open Source Software Projects. *Proceedings of the 18th ACM*

Conference on Computer Supported Cooperative Work & Social Computing - CSCW '15, 1379–1392. http://doi.org/10.1145/2675133.2675215

Steinmacher, I., Wiese, I., Chaves, A. P., & Gerosa, M. A. (2013). Why do newcomers abandon open source software projects? *2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, 25–32. http://doi.org/10.1109/CHASE.2013.6614728

Sunindyo, W., Moser, T., Winkler, D., & Dhungana, D. (n.d.). LNBIP 94 - Improving Open Source Software Process Quality Based on Defect Data Mining. *Source*, 84–102.

Tan, P.-N., Steinbach, M., & Kumar, V. (2005). *Introduction to Data Mining, (First Edition)*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Thomas H. Davenport, Jeanne G. Harris, R. M. (2010). *Analytics at Work: Smarter Decisions, Better Results*. Harvard Business Review Press.

Tsang, E. W. K. (2014). Case studies and generalization in information systems research: A critical realist perspective. *The Journal of Strategic Information Systems*, *23*(2), 174–186. http://doi.org/10.1016/j.jsis.2013.09.002

Ur, S., Yom-tov, E., & Wernick, P. (2007). An Open Source Simulation Model. *Development*, 124–137.

Van Der Aalst, W., Adriansyah, A., De Medeiros, A. K. A., Arcieri, F., Baier, T., Blickle, T., … Wynn, M. (2012). Process mining manifesto. *Lecture Notes in Business Information Processing*, *99 LNBIP*(PART 1), 169–194. http://doi.org/10.1007/978-3-642-28108-2_19

van der Aalst, W. M. P. (2016). *Process Mining*. *Process Mining* (Vol. 5). http://doi.org/10.1007/978-3-642-19345-3

Van Der Aalst, W., Weijters, T., & Maruster, L. (2004). Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, *16*(9), 1128–1142. http://doi.org/10.1109/TKDE.2004.47

Van Der Walt, S., Colbert, S. C., & Varoquaux, G. (2011). The NumPy array: A structure for efficient numerical computation. *Computing in Science and Engineering*, *13*, 22–30. http://doi.org/10.1109/MCSE.2011.37

Vasilescu, B., Posnett, D., Ray, B., van den Brand, M. G. J., Serebrenik, A., Devanbu, P., & Filkov, V. (2015). Gender and Tenure Diversity in GitHub Teams. *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, 3789–3798. http://doi.org/10.1145/2702123.2702549

Wang, S., Zhang, W., Yang, Y., & Wang, Q. (2013). DevNet: Exploring developer collaboration in heterogeneous networks of bug repositories. *International Symposium on Empirical Software Engineering and Measurement*, 193–202. http://doi.org/10.1109/ESEM.2013.24

Weiß, C., Zimmermann, T., & Zeller, A. (2007). How Long will it Take to Fix This Bug ?, (2).

Ye, N., Baddeley, A. D., & Kopelman, M. D. (2015). *The Handbook of Data Mining*. *Cognition*.

Zelkowitz, M. V., & Wallace, D. (1997). Experimental validation in software engineering. *Information and Software Technology*, *39*, 735–743. http://doi.org/10.1016/S0950-5849(97)00025-6

Zelle, J. M. (n.d.). Python as a First Language. Retrieved from http://mcsp.wartburg.edu/zelle/python/python-first.html

Zhou, M., & Mockus, A. (2012). What make long term contributors: Willingness and opportunity in OSS community BT - 34th International Conference on Software Engineering, ICSE 2012, June 2, 2012 - June 9, 2012, 518–528.

Zhou, M., & Mockus, A. (2015). Who Will Stay in the FLOSS Community? Modeling Participant's Initial Behavior. *IEEE Transactions on Software Engineering*, *41*(1), 82–99. http://doi.org/10.1109/TSE.2014.2349496

Zhou, M., Mockus, A., Ma, X., Zhang, L. U., & Mei, H. (2016). Inflow and retention in oss communities with commercial involvement: A case study of three hybrid projects. *ACM Transactions on Software Engineering and Methodology*, *25*(2). http://doi.org/10.1145/2876443

Appendix

A - OUR OPEN LETTER TO THE OPEN SOURCE COMMUNITY

The benefits of Open Source Software (OSS) are difficult to measure. Some have tried (http://radar.oreilly.com/2012/07/open-source-small-business-report.html) but no one is in dispute that the world of big data, the software industry, and arguably the world economy as a whole benefits when leading OSS projects release new versions of their product. If we could find better practices for managing and leading in such development efforts, wouldn't that increase this benefit?

This was my objective during four papers that made up my PhD dissertation at Arizona State University this past year and I am anxious to share the insights I gained. However, I am sure many will hesitate to invest the time to read the ~150pages, so this is my open letter to the OSSD community where I summarize my findings and make some suggestions.

Dear Leaders, Participants and Casual Contributors of Open Source Software,

Anyone who has participated in more than one Software development project knows that there is no one "correct" way to go about working together and no one person has all the answers for how team should be coordinated. However, after analyzing over 13 years of data involving 448 teams, 500k tickets of requirements from Apache Software Foundation (Apache) we have gained several insights into how some of the ore successful teams operate and we have recommendations for some processes and policies that may benefit many OSS teams.
The main topics I addressed in my research were:

- How do the tickets submitted by massive amounts of casual contributors[21] differ from those from contributors who are more involved?
- Once tickets are written, what have been the ideal means to distribute them based on the ratio of work completed?

[21] those that only contribute a few and often 1 ticket

- How have factors such as volunteer drop off or requirements volatility impacted the completion rates of teams at Apache

7.1 Methodology Summary

With the help of many collaborators, I was able to study all the Jira tickets at the Apache Software Foundations from 2002 – 2015 to find associations between certain process choices and outcomes measured in terms of the proportion of work that is completed. I do not argue that this is the perfect means to measure success as it does not take into account that the age of tickets varies from the given point in time. However, I chose this criterion because in order to utilize sequential pattern mining I needed discrete events within each ticket's history to indicate successes or their absence to indicate an anti-pattern.

Extreme detail on my methods is best explained in my dissertation "In Pursuit of Optimal Workflow Within The Apache Software Foundation."

7.2 Ticket Creation

First some findings on the participant contribution levels

- 59% of ticket writing participants at Apache only write one ticket
- 74% of those are defects
- Of participants that write tickets, the mean count is 5.8
- The top 1% are users resolving more than 110 and creating more than 99 tickets
- The top 0.5% is the only demographic (when bucketed by half percentiles in ticket writing) that create a majority of non-defects

⇨ In short, there is a very large but important demographic of "casual contributors" moving a project forward in addition to a much smaller group usually referred to as the "core contributors."

When a ticket is closed without a code change, Jira provides a means to capture that information whether it's because the ticket was a duplicate of another, was not valid, or simply that the suggestion was not deemed worthy of some one's time. Although there are many legitimate reasons for a ticket to receive a non-fix resolution, we consider the ratio of tickets a user or a demographic writes that are fixed as an indication of ticket writing effectiveness. Just as it was

found to be true at Microsoft in (Guo, Zimmermann, Nagappan, & Murphy, 2010) we found that the more tickets a user writes and the longer they have been associated with a project, the more likely they are to write tickets that are actually implemented.

7.3 Suggestions to the OSS Community:

- A very large portion of defects are written by casual contributors of your products, so make it as easy as possible to receive feedback. Perhaps a simple form that does not require a Jira login would be effective for some projects
- Put much greater faith in tickets from experienced users. Perhaps there could even be roles assigned that would allow some indication that a ticket, especially a non-defect, is written by someone with experience in that project. This could also be a form of recognition to the more experienced, thus adding a small element of gamification.
- Encourage verification of all tickets but perhaps there should be more emphasis on casual contributor tickets. Many great ideas are still coming from the users that just haven't had time to contribute at greater level but historical trends indicate that these suggestions might need a bit more vetting.
- We suggest it is possible to give a small number of core contributors more recognition by designating this ticket verification and management role specifically. Considering trends in the industry, this may be an important role of a product manager in OSS.

7.4 Sharing the work

Utilizing the awesome power of sequential pattern mining [https://en.wikipedia.org/wiki/Sequential_pattern_mining], I was able to find the historical completion rates of tickets under many different conditions. It's important to point out that these are not cause and effect relationships. I make suggestions because some circumstances produced a higher ratio of completed work than other circumstances.

Tickets are significantly more likely to be completed if:

- It is a defect and has a release version assigned at creation
- It is an improvement that is created with an owner

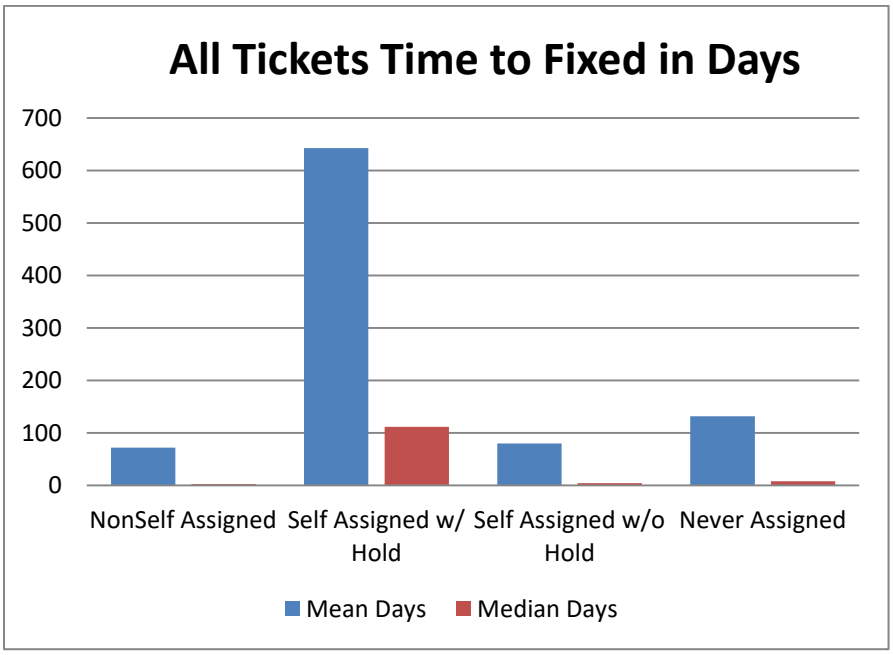
Tickets are significantly less likely to be completed if:

- New Features are not given an owner from day one
- A ticket loses an owner without another replacing it immediately

7.5 Claim and Hold

A major source of delay

An accidental discovery in our sequential pattern mining efforts was the realization that ~10% of tickets are claimed by participants but are not completed for long periods of time.



In short, the median time to fix for the "without hold" group is 95% less than the hold group. For tickets that are never assigned, the median time to fix is 92% less than the hold group. Of course, many extreme outliers are driving the great difference between mean and median but we found the astonishing delay times alarming. Further analysis in my paper shows this delay exists in tickets of all types and "priorities"

7.5.1 Suggestions to the OSS Community:

- The community should consider an alternative means to express interest in a piece of work. If a participant can show interest without preventing others from accomplishing the same task, this delay might be greatly reduced.
- On some teams, it might appropriate to (after a respectful series of warnings) to relinquish some tickets back into the unclaimed pool while bringing attention to their newly found freedom. Participants with the best of intentions may re-claim them but we speculate that many tickets will become completed sooner so as to avoid the loss of status or because new eyes might re-assess a tickets urgency or ask originator to verify it is still needed

7.6 Prevalence of Volunteerism

Tracking the times that a contributor identified a ticket and assigned said ticket to themselves was our primary indicator of volunteerism because it contrasts the times that a ticket in need is assigned to another user, an indicator of delegation. Teams at Apache depend on this form of volunteerism at remarkably different levels and we do not assume that this is the only way to measure volunteerism. However, we did find a correlation between the proportion of tickets a team has that have this indicator and the proportion of tickets that are closed (0.49 coefficient) or fixed (0.44 coefficient). What was interesting to us was that if we exclude the largest projects (in this case the 29 larger that 4k tickets) these correlations become even stronger with close rate (0.52 coefficient) and fix rate (0.52 coefficient). We suspect this is simply because *some* of the largest teams still succeed with a certain degree of delegation.

7.6.1 Suggestions to the OSS Community:

We suggest all teams encourage volunteerism by not discouraging it:

- As mentioned, those that hold tickets for extended periods are preventing others from accomplishing the work in question. Are some of these the most appealing tickets for various reasons? If so, shouldn't the person that is most likely to complete the tasks in a timely manner deserve the best shot at owning it *rather* than the person who sees the ticket first?
- If Requirements clarification can be done in a manner that does not erode the perceived value of the improvements requested, we recommend making requirements as clear as possible
- Provide recognition for rising contributors. We suspect that, if done properly, modest gamification of a team's progress will provide enough visibility for those resolving tickets that ticket completion will increase

We feel these suggestions are especially true for most teams that are not functioning with a solid core team that can accept work from others without feeling less motivated to deliver.

7.7 Requirements volatility

We tracked the ratio of tickets a team has with changes to the title (summary) and details and sought correlations with completion rates.

- Teams that alter the summaries tended to alter the descriptions with a correlation coefficient of 0.58

- Amongst large size projects, there was a correlation coefficient of -0.38 between ticket closing and description change and -0.56 between summary change and closing, although with smaller projects this was the correlations were weaker

We had speculated that the teams that valued the precision of their requirements would have a higher completion rate but the opposite was true. There could be many causes for this trend but we speculate that debates among requirements might erode the perceived value of the suggestions. If there is doubt about what should be done perhaps it is not worth some one's time. There also might simply be social pressure involved in requirements debate. Many OSS contributors never meet in person yet there are social structures in place that carry a heavy influence over their high profile community. If a user wants to make a contribution, it might simply be easier to find a ticket that will not offend a key player on a given team.

7.7.1 Suggestions to the OSS Community:

Just as mentioned earlier, designating a small group of core folks as product managers may give them the recognition they deserve to facilitate requirements verification as stated earlier but also to settle requirements disputes. If tickets are determined to be "settled" we hope that more developers will be interested in completing them.

7.8 Open Letter Conclusion

We were able to establish insights from event logs originating from the Apache Software Foundation using hypothesis testing and sequential pattern mining as well as make some suggestion to the Open Source Software (OSS) community. Although feedback from the general community should be highly encouraged, we point out that these requirements suggestions are not as reliable as those from the dedicated participants and should be treated as such. In a significant portion of tickets, great delays are created by participants claiming a ticket but not delivering. We ask the OSS community to consider rethinking this form of unconditional ownership while still respecting contributor's best intentions. We also suggest giving recognition

to those that manage and clarify ticket requirements is almost as important as recognizing those that resolve tickets as uncertain requirements is associated with incomplete tickets.