Quantifying and Mitigating Turnover-Induced Knowledge Loss: Case Studies of Chrome and a project at Avaya

[†]Peter C. Rigby, [†]Yue Cai Zhu, [†]Samuel M. Donadelli, [§]Audris Mockus
[†]Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada
[§]Department Electrical Engineering and Computer Science, University of Tennessee, Knoxville, USA
first.last@concordia.ca, audris@avaya.com

ABSTRACT

The utility of source code, as of other knowledge artifacts, is predicated on the existence of individuals skilled enough to derive value by using or improving it. Developers leaving a software project deprive the project of the knowledge of the decisions they have made. Previous research shows that the survivors and newcomers maintaining abandoned code have reduced productivity and are more likely to make mistakes. We focus on quantifying the extent of abandoned source files and adapt methods from financial risk analysis to assess the susceptibility of the project to developer turnover. In particular, we measure the historical loss distribution and find (1) that projects are susceptible to losses that are more than three times larger than the expected loss. Using historical simulations we find (2) that projects are susceptible to large losses that are over five times larger than the expected loss. We use Monte Carlo simulations of disaster loss scenarios and find (3) that simplistic estimates of the 'truck factor' exaggerate the potential for loss. To mitigate loss from developer turnover, we modify Cataldo *et al.*'s coordination requirements matrices. We find (4) that we can recommend the correct successor 34% to 48% of the time. We also find that having successors reduces the expected loss by as much as 15%. Our approach helps large projects assess the risk of turnover thereby making risk more transparent and manageable.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: [Metrics]; K.6.3 [**Software Management**]: [Software development; Software maintenance; Software process]

Keywords

Quantitative Risk Management, Mining Software Repositories, Knowledge Distribution, Truck Factor, Successors, Turnover

ICSE '16, May 14 - 22, 2016, Austin, TX, USA

ISBN 978-1-4503-3900-1/16/05...$15.00
DOI: http://dx.doi.org/10.1145/2884781.2884851

1. INTRODUCTION

Software projects have a variety of risks that range from uncertainty about future technologies and user needs to simple programming errors. Individuals who create software transfer their, often tacit knowledge, into the inner workings of the system making it difficult for others to maintain. It is, therefore important to evaluate the risk of developers leaving the project. Developer turnover has a wide range of reasons, including job dissatisfaction from boredom or conflict, alternative job opportunities, and personal issues [17]. Organizational changes in the business environment and globalization may lead to turnover in the form of outsourcing and offshoring practices [25, 21]. While new developers may bring insights and enthusiasm, the departure of developers negatively affects both quality and productivity in a software project creating substantial risks [20, 30]. The aim of this paper is to quantify the risk of developer turnover and investigate ways to mitigate this risk to make projects more resilient.

Risk management involves the quantification of the impact of a risk and the probability of it materializing. We measure the impact of turnover via knowledge loss (source files abandoned after developers leave). We adapt techniques from financial risk management and use historic knowledge loss data to obtain the probability that a certain level of losses will be sustained. We run simulations to determine alternate turnover scenarios. In historical simulations we randomly choose the set of developer who leave, but keep the number of leavers at historical levels. In a Monte Carlo simulation we vary the number of developers leaving the project to simulate disaster or 'truck factor' scenarios. Combining software mining techniques with quantitative risk management, we estimate and simulate how at risk (risk exposure) a software project is to developer turnover and examine ways to reduce the risk by suggesting successors.

We perform two case studies on large software projects because the risks from turnover are trivially assessed on small projects, *i.e.* if a small group of developers leaves, the entire project is abandoned. We chose one open source project and one project in a traditional closed-source environment to measure the differences in their risk profiles and answer three research questions.

1.1 Research Questions

RQ1: Loss distributions: What is the historical knowledge loss distribution and the size of large unexpected losses?

While turnover in moderate amounts may bring innovation to the project [2], high turnover rates have been associated with lower quality because of loss of expertise [20], lower productivity because newcomers may not be as productive as the developer they replace [30, 19] and may introduce more defects [12]. In other domains, turnover has been shown to be associated with higher incidence of health problems in a study of mid-level managers and executives in the Canadian civil service [1]. Furthermore, maintaining abandoned code may lead to higher incidence of serious defects [20] because the project survivors are likely to lack the understanding of the design and structure of the abandoned code. We, therefore, gauge the rate of turnover in projects.

In particular we calculate the loss distribution, expected loss, and two measures of unexpectedly large losses: Knowledge at risk (KaR) and the expected shortfall (ES) [18]. Unexpectedly large losses describe the right tail of the loss distribution. With most losses being small, the expected loss may not appear to be large. However, large unexpected losses, such as when a core developer leaves, can threaten the success of a project. We examine the level of risk from the perspective of expected and unexpected knowledge loss and compare the projects to each other. In particular, our risk analysis allows project managers to estimate the number of developers they may need to replace in any quarter, *i.e.* three month period.

RQ2, Turnover simulations: How susceptible is the project to alternate historical loss and disaster scenarios?

The historical loss distribution describes losses that actually occur. Financial risk assessments required at large banks involve a historical simulation and often a Monte Carlo simulation of loss scenarios [18]. We replay the losses sustained each quarter and use stratified random sampling to select developers to leave. This historical simulation gives us a sense of how bad the actual loss is compared to how bad it could have been. Some recent studies have evaluated disaster loss scenarios, which are known colloquially as the 'truck factor' [24, 29, 26, 10]. These previous works have only quantified how large disaster scenarios can be but not how *likely* they are to occur. For risk management the impact (loss) needs to be weighted by the likelihood of an event. Previous work on disaster scenarios, however, does not consider the likelihood of the event. Following financial models, we incorporate the most likely disaster scenarios (*i.e.* those that occur at least 1% of the time) to enable risk management.

RQ3: Successors: Can we mitigate the impact of turnover by suggesting the developer who should take over an abandoned file?

Turnover is inevitable; for example, in a report by the job website *PayScale* the average tenure of a new developer at Google and Amazon is only slightly more than one year [23]. It is important to find ways to mitigate the effects of high turnover on code ownership by identifying the best candidates to take over the maintenance of abandoned code.

A development team faced with a departing developer can hire a new developer or select an existing developer to maintain abandoned code. Reserving abandoned files for incoming developers would be effective in keeping existing developers on the code they already own, but it may be risky because newcomers may not understand the project, let alone the specific code that was abandoned [30]. To reduce risk of errors it may be critical to have experienced developers maintain more central code instead of leaving it without an owner or in the hands of newcomers. We first investigate which option was chosen in the projects we studied.

We then use a modified version of Cataldo *et al.*'s [8] coordination requirements matrix to recommend developers who have worked in similar areas to the abandoned file. We measure how many possible successors exist for each abandoned file in the system and evaluate how well our technique predicts actual succession on abandoned files. We also determine how much the risk of turnover is reduced by having available successors. Our results quantify, for project managers, the reduction in risk attained by assigning co-developers to risky files.

The remainder of the paper is structured as follows. In Section 2 we describe our data and methods. In Section 3 we introduce quantitative risk management, including the historical loss distribution and measures of unexpected loss. In Section 4, we simulate alternative historical losses and disaster or 'truck factor' scenarios. We assign a probability to each loss thereby allowing teams to plan for higher probability losses instead of unrealistic maximal losses. In Section 5, we mitigate losses from turnover by recommend possible successors based on co-change relationships with abandoned files. We determine how well we can predict successors and if these successors reduce the risk involved in turnover. In the final sections, we discuss threats to validity and conclude the paper.

2. DATA AND METHOD

To study turnover, we select two large projects one industrial project and one open source project. The industrial project is from Avaya and has been developed for over twenty years. We will refer to this project as **Avaya** throughout this paper. The Avaya project is in the domain of core telephony and networking and is sold to small to medium sized businesses. The code is mostly written in the C language as cost concerns require efficiency of an embedded platform. The product has over 5M SLOC with the development team primarily located in the UK, Romania, and India.

The second project is **Chrome** a Google-lead project that is open source. In contrast to the Avaya project, Chrome is a webbrower designed for end users. Chrome development is conducted in public but the practices it uses mirror those used by Google internally and many of the developers are Google employees. Chrome has been under development since 2008 and this study follows it up to the present. Excluding third party tools that are part of the Chrome source repository, there are 3M LOCs. Since there are a large number of peripheral contributors on OSS projects, we define the core team using Mockus *et al.*'s measure of the top developers whose combined effort is 80% of the development work. We stratify the developers into core and non-core to limit the volatility introduced by transient developers.

2.1 Identifying Developers

On Chrome, the author and committer of the change are the same and are identified by his or her email address. A single developer may have made commits with multiple email addresses. To resolve multiple address to a single individual, we use Canfora *et al.*'s [7] name aliasing tool. We added an additional cleaning stage where diacritics are converted into their ASCII form as their tool cannot handle these characters (*e.g.,* ŏ is converted to o). At Avaya, developers use a company assigned username when they commit changes to the system, so a manual check was sufficient to correct any variations in duplicate usernames.

We need to know when a developer joins and leaves a project. For OSS projects, we do not have the official records of when a developer joins and leaves a project. We use developers first commit and last commit dates to indicate when they join and leave a project, respectively. Although Avaya keeps records of when developers join and leave the company, the records of when they join and leave a project are not kept. Since our results are at the project level, we use the same commit data strategy for estimating joining and leaving times for the Avaya project.

2.2 File Ownership and Abandonment

For this paper, the amount of knowledge lost when a developer leaves is dependent on what code the developer created or maintained. We do not factor in non-code contribution, such as those made by a user experience designer or architect. Determining code ownership and the related concept of developer expertise has received considerable attention in the software engineering literature. Previous studies consider each commit to a software artifact as a unit of developer's ownership and expertise [22, 4]. Further elaboration of ownership is the degree of knowledge model where later commits decrease the expertise of developers with earlier commits [13]. These measures have a serious shortcoming because the number of commits is not always representative of the knowledge in the system that must be maintained. For example, with a commit-based approach the deletion and addition of lines are counted equally. However, the deletion of a line removes knowledge that must be maintained, while an addition increases the maintenance burden. As a result, a commit-based approach is inaccurate when assessing the amount of source code knowledge that must be maintained when a developer leaves a project. For example, a team could delete the leaving developers module reducing the maintenance burden to zero.

Instead of a commit-based approach, we use a blame-based approach. The blame function present in version control systems determines the person who last changed a line of code. In this way we are able to follow the ownership trend at a finer granularity of each line of code in the system. We limit our analysis to source files only, for example, on Avaya we only consider '.c' and '.h' files. We consider a file to be abandoned when 90% of the lines in the file have been abandoned. We also use the blame-based ownership approach to simulate historical-based and disaster scenarios. The simulation methodology are presented in Section 4.

2.3 Definitions

We measure turnover at quarterly intervals *i.e.* in four month periods. We use the following dependent definitions of ownership, abandonment, and knowledge loss:

1. *Line ownership* is calculated using git blame to determine the last person changing a line of code (See Section 2.2).

2. A *developer leaves* during the quarter of his or her last commit. We exclude the last year of historic data to avoid categorizing an up to 364 day leave of absence as departure.

3. A *line of code is abandoned* when blame attributes the line to a developer who has left the project.

4. A *file is abandoned* when 90% or more of the lines in a file are abandoned. We use the 90% threshold to exclude developers with trivial contributions to a file. Note: a file that is abandoned in one quarter may be adopted by developers making changes in subsequent quarters.

5. The *knowledge loss* in a quarter is the number of files that are abandoned.

In Sections 3 and 4, we describe how we adopt financial risk models by replacing dollar loses, as in writing off bad loans, with knowledge loss, *i.e.* files abandoned by developers leaving in a quarter. This mapping does not violate the assumptions of the financial risk models and, therefore, can be applied to assess risk from knowledge loss.

2.4 File Dependencies and Succession

When turnover occurs and files are abandoned, we want to be able to suggest the most qualified developer to take over the maintenance of the file. Our simple method involves calculating co-changing files using a modified version of Cataldo *et al.*'s coordination matrices [8]. We test whether developers who have changed files that co-changed with an abandoned file are likely to take over the maintenance of the abandoned file. We also quantify the degree to which files that have potential successes reduce overall turnover risk. The details of our approach is described in Section 5.

3. ACTUAL LOSSES AND RISK MEASURES

RQ1: What is the historical knowledge loss distribution and the size of large unexpected losses?

Risk management involves quantifying the *impact* or size of a loss and determining the *probability* that a loss of that size will occur [5]. To model financial risks, large banks track the losses that they sustain from a set of loans. Overtime they are able to understand the loss distribution, which in turn allows them to assess the riskiness of their loan portfolio [18]. Since software development is a knowledge intensive activity, we adapt this methodology to measure knowledge loss from developer turnover on large software projects.

Using the history of the software projects we calculate the loss distribution as the number of files that become abandoned in a four month period, *i.e.* a quarter. Based on this distribution we can calculate the expected loss and two measures of unexpected large losses: the knowledge at Risk (KaR), which measures the size of a loss that has a 5% chance of occurring, and Expected Shortfall (ES), which measures the average size of losses that happen less than 5% of the time.

Historical knowledge loss distribution: In Figures 1 and 2 we see the loss distribution as the number of files abandoned in a four month period. We can see the mean and median loss for Avaya are 209 and 130 files and for Chrome they are 194 and 132, respectively. This means, for example, that for Chrome we can expect that 194 files will be abandoned in any quarter. From the figures it is clear, that the number of abandoned files is not normally distributed with the majority of the values cluster around small numbers of abandoned files per quarter. Although the mean is an overestimate of the median loss, in risk management we are concerned with how big we can expect our losses to be, so we are interested in the right tail of the loss distribution.

Risk from unexpected large losses: The economic models in the 1980's failed to predict the recession, *i.e.* a

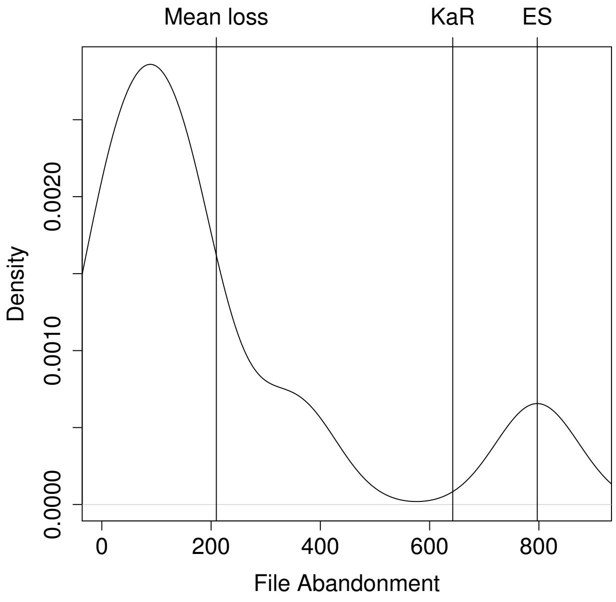

Figure 1: Avaya: loss distribution

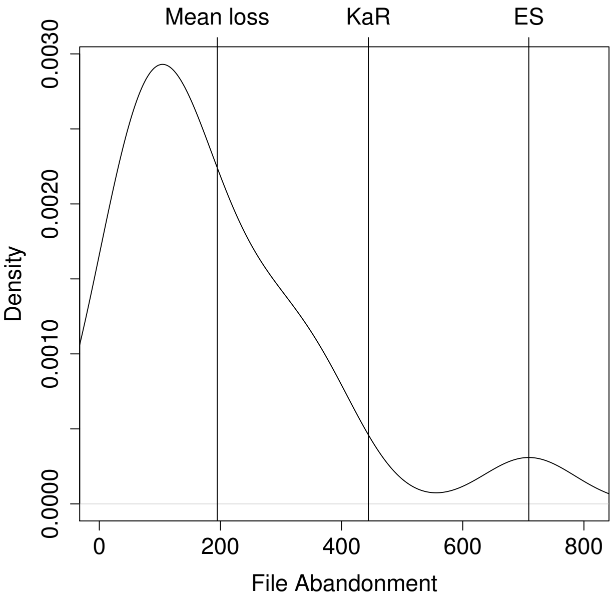

Figure 2: Chrome: loss distribution

series of large losses, as a result, economists developed techniques to predict large losses [16]. As we can see from our loss distributions, Figure 1 and 2, it is not normally distributed. So if we naively assume that we will have the mean loss in each quarter, we underestimate the size of a large turnover event. Since these large turnover events can have disproportionately high risks, *e.g.,* the project may fail from high knowledge loss if we lose core team members, it is critical to understand how often we can expect to have a large loss. We adapt the most common financial measures of unexpected large losses to a software knowledge loss setting.

Knowledge at Risk (KaR): is adapted from Value at Risk (Var) and is defined over a confidence interval $\alpha \in (0, 1)$. Given a confidence level of α, KaR_α is the size of loss, l, that actual loss, L, occurs with a probability of $1 - \alpha$ [18]:

$$KaR_\alpha = \inf\{l \in \mathbb{R} : P(L > l) \leq 1 - \alpha\}$$

It can be thought of $1 - \alpha$ percentile of knowledge loss distribution. In effect, KaR_α is the quantile at α in the loss distribution. Unless otherwise mentioned we use KaR_{95} in this work. KaR_{95} can be interpreted as the size of loss that has a 5% chance of occurring in a quarter. From the loss distributions in Figure 1 and 2 we see that the KaR_{95} for Avaya is 642. This means that in any quarter there is a 5% chance that 642 of the files will become abandoned. The corresponding KaR for Chrome is 444 files.

Expected Shortfall (ES): The main criticism of KaR is that it can underestimate the size of the largest losses [18]. In effect, KaR_α is the maximum loss that would occur at most $1 - \alpha$ percent of the time. Expected shortfall is the expected value that would occur at least $1 - \alpha$ percent of the time. Formally ES_α can be defined in terms of KaR_α and expected value, E, as

$$ES_\alpha = \mathrm{E}(\{l \colon l \geq KaR_\alpha\})$$

As we can see in the figures, the ES_{95} or mean of the largest losses that occur at most 5% of the time for Avaya and Chrome are 797 and 709 files, respectively.

The longer the right tail of the loss distribution the larger the unexpected losses. The ES is 1.2, 3.8, and 6.1 times larger than the KaR, mean, and median for Avaya. The ES loss is 1.2, 3.6, and 5.4 times larger than the mean, median, and KaR for Chrome. For both projects, the ES is many times larger than the expected loss indicating a long right tail and a potential for large losses.

Previous works on software knowledge loss report losses as percentages of total files or lines [15, 24]. These works examined small projects, but on larger projects even large losses of knowledge can be small compared to the overall size of the project. For example, the ES of 709 files on Chrome represents only 4% of total number of files in the system. Despite this low percentage, it is a daunting task to find maintainers for 709 recently abandoned files. As a result, we follow the norm in financial risk management and report the actual losses instead of the percentage of total loss. In the next section we simulate disaster scenarios and give further examples of why percentages are inappropriate to represent risk from loss. These simulations also make our results actionable for project managers.

4. SIMULATING LOSS SCENARIOS

RQ2: How susceptible is the project to alternate historical loss and disaster scenarios?

We adapt the simulation methodologies used by large banks with the goal of understanding how bad losses could have been [18]. The loss distribution and resulting knowledge at risk depend on the knowledge distribution of the system (*i.e.* the file ownership distribution) and the specific developers who leave. Although we have a limited number of quarters for Avaya and Chrome, 8 and 17 respectively, we are able to use the hundreds of developers and tens of thousands of files in the underlying file co-ownership distribution to estimate the likelihood of extreme events. Through simulation, we manipulate these two variables to understand how the knowledge distribution has changed over time (historical simulation) and how well the current distribution of knowledge withstands 'disaster' scenarios (Monte Carlo simulation). By using the underlying file co-ownership and randomly sampling the groups of developers that leave, we are able to assess the risk in a wide range of scenarios.

4.1 Historical Simulation

For the historical simulation, we hold the file co-ownership distribution (*i.e.* knowledge distribution) and the number of developers who leave constant, but vary which developers actually leave. In this way we understand whether the actual loss was better or worse than what we could expect based on the file co-ownership distribution and the number of people who leave. Since the distribution of work on the development team is skewed, we stratify developers into core and non-core (see Section 2). It is important to stratify developers because different types of developers have different reasons to leave. Specifically, we do the following:

1. we use the knowledge distribution of each quarter
2. we use the number of core developers, l_c, and non-core developers, l_{nc}, who leave for each quarter
3. we use stratified random sampling [9] to choose the same number of leavers, $rnd(l_c, l_{nc})$. Stratified random sampling ensures that we randomly pick the number of core and non-core developers who actually left in a quarter. Since more non-core developers leave, if we use uniform random sampling, we would randomly pick an unrealistically large number of core developers as leavers. This would inflate the risk. We run 1000 simulations for each quarter
4. we compare the actual knowledge loss in a quarter to the simulated loss in terms of expected loss, KaR, and ES

In Figure 3 and 4 we see the simulated historical losses for Avaya and Chrome. The vertical line represents a 95% confidence interval. The value at the 95th quantile is the simulated KaR and the triangle represents the simulated ES. The actual loss is represented by an a cross. For Avaya, we see that the actual losses are always within a 95% confidence interval of the simulated random losses, indicating that the actual knowledge loss cannot be statistically differentiated from the randomly simulated losses. Quarters 4, 6, and 8 show a trend toward a higher actual loss than the simulation mean indicating that in these quarters important developers left. In contrast, for Chrome, the actual loss is always at or below the simulated mean indicating that in all cases developers with lower importance left, and in some cases the result is statistically lower (*e.g.,* quarters 15 and 16). On Chrome, it appears that core developers that are likely to cause larger knowledge losses are less likely to leave.

Comparing the actual losses to the unexpected losses, we see that the most extreme simulated ES for Avaya and Chrome is 1114 and 1099 files, respectively. This represents a 5.3 and 5.7 times larger loss than the expected actual loss. Compared with the actual ES of 797 and 709 the largest simulated ES is between 1.4 and 1.6 times larger for Avaya and Chrome. The simulations clearly show that the losses could have been much worse than they actually were. This indicates that the more files a developer owns (that are not co-owned by others) the more that developer has invested in the project and the less likely he or she is to leave. Future work investigating this relationship is necessary.

4.2 Disaster Scenario Simulation

Recent works studying turnover have quantified disaster loss scenarios through the 'truck factor' – the number of developers that must leave (*e.g.,* get hit by a truck or bus) before the project becomes unsustainable [29]. However, the truck factor provides only the impact of the worst case loss, not its likelihood. Risk analysis has two aspects, the impact of a loss and the probability that the loss will occur, so we simulate disaster scenarios and illustrate how likely they are to actually occur.

To calculate the truck factor one must consider the number of files that are abandoned when a group of size g developers leave. For a team of n developers this involves $\binom{n}{g}$ calculations. This is infeasible on large projects and even after adding a stopping condition to the truck factor algorithm, we were only able to calculate up to $g = 7$ on Chrome [11]. Previous studies of the truck factor have examined small projects [24, 29, 26, 10]. To calculate the truck factor for a large projects we use a Monte Carlo simulation. It not only provides an approximation of the worst possible case, but also provides estimates of the likelihood for any particular amount of loss that occurs in the simulation. We follow these steps:

1. we use the knowledge distribution of the most recent quarter
2. we vary the group size, g, of developers who leave from 1 to 200 people
3. we select groups of developers to leave at random, $rnd(g)$. We do 1000 runs for each group size.
4. using the actual losses in terms of files and developers and the historically simulated losses we determine the likelihood of each disaster scenario

To make our results easier to interpret we have added two lines to our disaster scenario simulations in Figures 5 and 6. The horizontal line represent the historically simulated ES. The vertical line represents the maximum number of developers we would expect to lose per quarter 1% of the time.[1] We have limited the group size on the x axis to 60 developers because, as we will see later, losses even at this level are highly unlikely. Since Chrome is open source and has many transient developers who contribute little but exaggerate the number leavers, we only consider core developers. For Avaya, we consider all developers as they are paid to contribute to the project.

Risk based on historical developer losses: The figures show that per quarter losses above 15 developers and 12 core developers have less than a 1% chance of occurring on Avaya and Chrome, which means, in terms of files, that

[1] We calculate developer loss using KaR_{99} on the historical number of developers who leave.

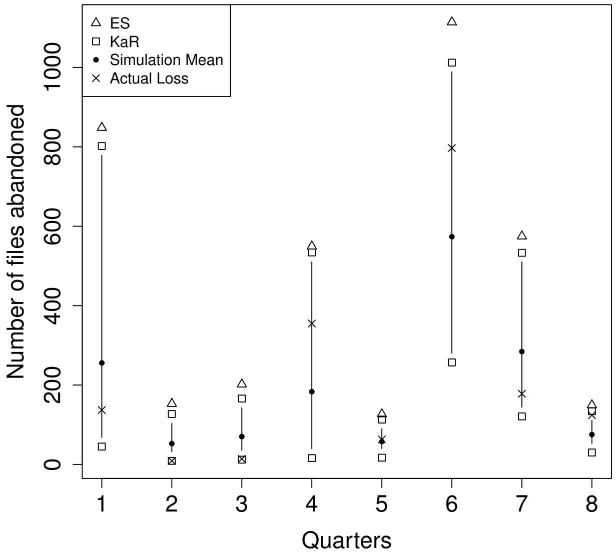

Figure 3: Historical Simulation for Avaya

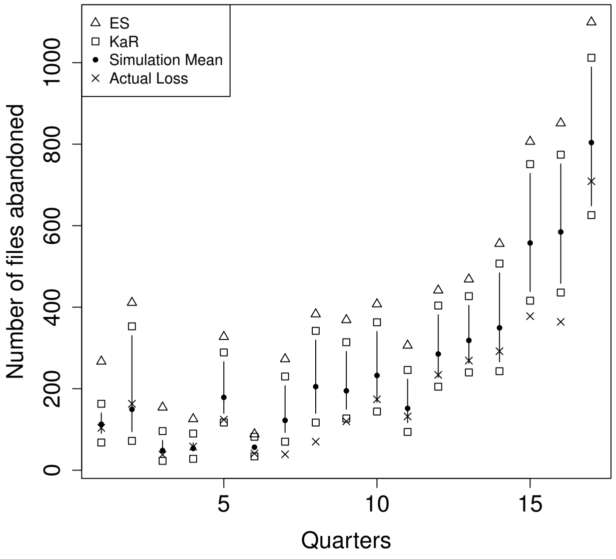

Figure 4: Historical Simulation for Chrome

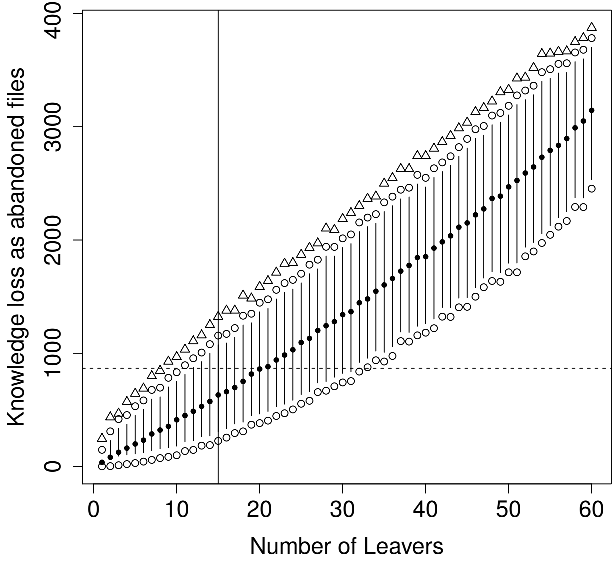

Figure 5: Truck Factor Simulation Avaya

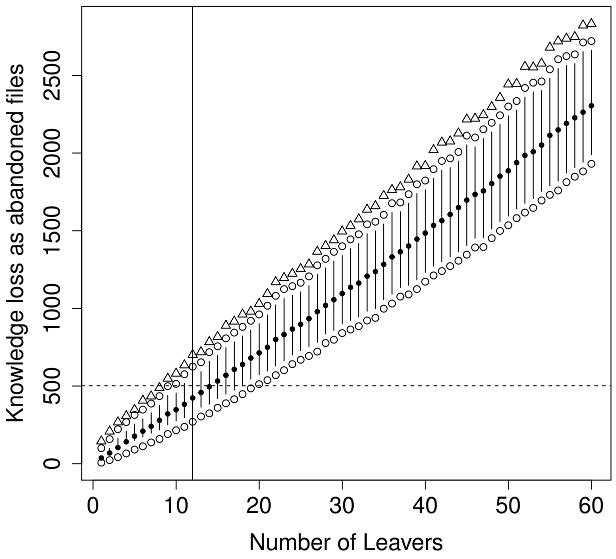

Figure 6: Truck Factor Simulation Core Developers on Chrome

Each figure contains a 95% confidence interval (the bounded vertical lines) and ES (the triangle) for the disaster simulation. Losses to the right of the additional vertical line have a probability of occurring less than 1% of the time. The horizontal line represents the simulated historical ES file loss. The range of 95% confidence intervals that it crosscuts indicates the likelihood of developer losses should be planned for.

contingencies for a disaster ES of 1322 and 701[2] files, respectively should be planned. Although we can simulate what would happen with much larger losses of developers, *i.e.* the truck factor, these catastrophic losses are usually uncontrollable by developers and project managers. For example, such catastrophes could be represented by outsourcing of an entire project or of Google removing funding for Chrome.

In contrast, banks usually hold reserves for losses that have a probability of occurrence of at least 1% [18]. If we use the same threshold, Chrome should ensure that they can deal with losses of 12 or fewer core developers in a quarter and Avaya should ensure that they can deal with losses of 15 or fewer developers.

Risk based on simulated historical losses: The disadvantage of the previous approach is that we use historical developer losses, which may underestimate the risk of unseen events. Another approach would be to use the simulated historical file losses that are based on the knowledge distribution to give a range of the number of developers who may have to be replaced. For Chrome in Figure 6, the horizontal line represents the historically simulated ES. Where the line intersects the 95% confidence interval, we can say that a loss of size ES will occur less than 5% of the time for core group loss of 10 or fewer developers. The same loss would occur at least 95% of the time for a loss of 19 or more developers. As a result, provisions in a loss plan for a disaster loss of at least 10 core developers should be made. Losses of up to 19 developers may happen even though, as we have seen historically, a loss above 12 developers is very unlikely. The corresponding disaster loss 95% confidence interval for Avaya ranges from 11 to 33 developers, however, losses above 15 developers are unlikely (see Figure 5).

As a final discussion point in this section, we again caution against the use of loss percentages. For example, Torchiano *et al.* [26] suggest that a good 'truck factor' threshold is the number of developers that would be necessary for 60% of the files to be abandoned. On the largest project that they studied there were 38 developers and if 7 left 60% of the project would be abandoned (the likelihood of such as loss was not calculated). Using this percentage on a project like Chrome, we find that a minimum of 200 developers must leave the project for 60% of the project to be abandoned. For Avaya to have 60% of its system abandoned the project would need to lose 50 developers. On large projects this 60% abandonment threshold could be renamed 'airplane factor' reminding managers that they should not put all their core developers on the same plane.

5. SUCCESSORS

RQ3: Can we mitigate the impact of turnover by suggesting the developer who should take over an abandoned file?

In the previous section we have quantified the size of loss a project can expect to have with a certain likelihood. This quantification allows teams to plan and prepare appropriately for turnover. In this section, we focus on providing information and suggestions that may help to mitigate developer turnover. Specifically, we look at the level of expertise of actual successors and propose a simple technique to find potential successors. Finally, we determine how much successors reduce the risk of turnover by calculating the reductions in the loss distribution if we adjust for the presence of such "backup" experts who can immediately step in when a developer leaves a project.

Some degree of turnover is inevitable, and when it happens a development team can either hire a new developer or assign an existing developer to the abandoned files. To understand which of these options teams choose in practice, we ask, *how much experience does the developer who takes over maintenance have?*

There is a large literature on mentoring and integrating new developers into software projects. For example, Zhou and Mockus examined the impact of development environment on new developers [31]. Bird *et al.* [3] looked at the survival rate of new developers. Zhou and Mockus examined the amount of time it takes a developer to become productive. Mockus [20] suggested mentors for developers based on past work and Canfora *et al.* [7] suggested mentors based on the email communication network.

Adding a new developer seems an obvious solution to file abandonment. However, on the projects we studied, the experts adopted the majority of abandoned files. In Figure 7 we can see that for Avaya 81% of developers who take over the maintenance of a file have at least one year and often many more years of experience. The corresponding number for Chrome is 54%, and only 16% of files are adopted by newcommers in their first quarter on the project.

Our findings are consistent with Zhou and Mockus's [30] developer learning curve. They find that although the number of tasks a new developer takes on plateaus at three to 12 months, when the centrality (*i.e.* how many files are included in a task) and difficulty are accounted for, developer productivity continues to increase over the entire measurement period of three years. In our study, we see that experienced developers tend to be the ones taking over the maintenance burden of abandoned files. Determining the type of files that newcomers tend to take on would be interesting future work.

5.1 Possible Successors

How many possible successors are there?

Since expert developers tend to take over the maintenance of an abandoned file, we want to determine which developer has the most related expertise. In this work we develop a simple preliminary succession measure. The measure is based on the intuitive idea that if file A has changed with file B, and file A becomes abandoned, then the developer who works on file B will likely know something about the functionality of file A. Our potential successor measure incorporates the commonly used measures of developer experience [22] and co-changes among files [6].

The successor measure involves calculating the developer to file matrix (Dev-File) based on the number of times a developer has changed a file. We then calculate the file to abandoned file matrix (File-AbandondedFile) based on the files that have co-changed with each abandoned file. We multiply the Dev-File matrix with the File-AbandondedFile matrix and are left with the Dev-AbandondedFile matrix. Since there are no developers who have changed the abandoned file left on the project, the Dev-AbandondedFile matrix represents the number of times each developer has changed a file that has co-changed with the abandoned file. We rank possible successors based by the number of files that they have changed that have co-changed with the abandoned file.

[2] The ES for Chrome is smaller than in the previous section because we only consider core developers in the disaster scenario

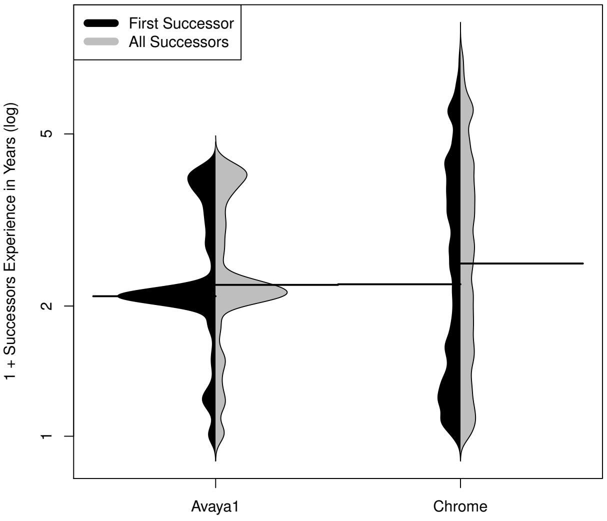

Figure 7: Experience of actual successors

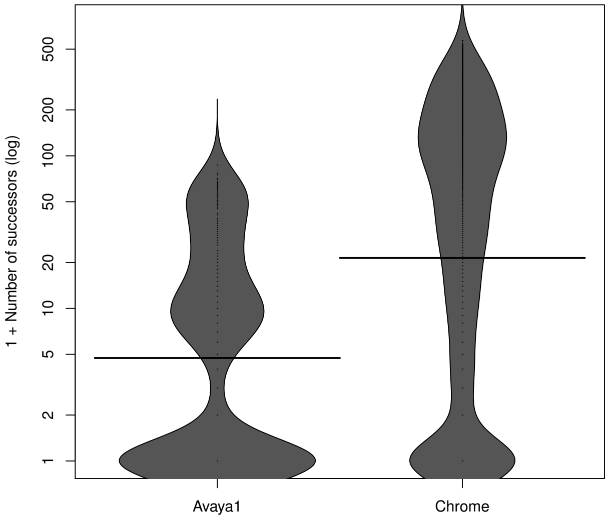

Figure 8: Number of potential successors for an abandoned file

The measure can also be seen as incorporating the first matrix multiplication in Cataldo *et al.*'s [8] coordination requirements measure. We apply the file matrix multiplication only to abandoned files.

On large projects like Chrome it can become very computationally expensive to perform these matrix multiplications. We eliminated commits that contained over 100 files as Hindle and German [14] showed that these commits are misleading because they often represent uninteresting changes, such as changing the copyright for all files in the system. The developer making this massive change is unlikely to understand all the relationships between these files. We implemented a database approach to multiply matrices and only include those files which co-changed with abandoned files making it possible to perform the multiplications on large projects.

The number of files that have at least one potential successor is 56% for Avaya and 77% for Chrome. In Figure 8, we see the median number of potential successor for abandoned files on the Avaya project is 5 and 35 for Chrome. Although many files have no successors, the distribution is skewed towards higher numbers of potential successors. At the 75th percentile Avaya files have 16 and Chrome files have 132 potential successors. The large number of successors for Chrome indicate that some files co-change with many other files. We can conclude that on both projects there is a reasonably good distribution of knowledge because there are a large number of developers who are potential successors for abandoned files. Suggesting hundreds of potential successors may not be helpful, so in the next section we use this measure to suggest the top five successors ranked by the number of files they co-changed with the abandoned file. We evaluate how often our suggested successors modify the abandoned file in the future.

5.2 Suggesting Successors

Can we mitigate the impact of developer turnover by suggesting potential successors?

Projects faced with turnover can assign the developer with the most related expertise, can hire a new developer, or can assign a developer who is available at that time. We evaluate how well each of these scenarios accurately predicts the person who performs future maintenance. We use the measure in the previous section to rank potential expert successors by the number of files they have changed that co-changed with the abandoned file in the past. We suggest a new developer who has no co-change expertise and less than 6 months of experience on the project. As a comparison, we suggest developers chosen at random to simulate the selection of a developer who happens to be available.

For each abandoned file we suggest 5 potential successors and a new developer, who has no co-change experience and less than six months of experience. As a simple control we select 5 "available" developers at random. Since there is usually only a small number of changes to an abandoned file in the future, we consider a suggestion to be correct if at least one of these developers actually works on the file in the future. Suggesting only five developers narrows the search space 18 and 113 times on Avaya and Chrome respectively.

The work that is most similar to ours is that of Mockus who created a measure to find mentors and successors for a developer who's job is being moved off-shore [19]. Much of the complexity of Mockus's measures come from determining who is mentor and who is follower. In our research this relationship is unnecessary because the file is abandoned and no current developer has modified the file. Furthermore, Mockus considers only the ownership matrix (*i.e.* Dev-File), while we multiply this matrix with the files that have co-changed with the abandoned file. Our output suggests who should take over an abandoned file, while Mockus's output suggests which developer should succeed or train a new developer.

To increase the resiliency of the project a combination of these two approaches could be taken where developers who's departure would lead to the largest losses would train the most suitable successors.

When we suggest a successor who actually modifies the abandoned file in the future, we consider our suggestion to be correct. We do not use recall because when a change must be made to an abandoned file, only one of many potential successors actually does the modification. In the control case where we randomly suggest five available developers, the suggestion is correct 10% and 7% of the time for Avaya and Chrome. In the case where we recommend an expert developer who has co-changed related files, the suggestion is correct 48% and 34% of the time for Avaya and Chrome. When we recommend a new developer, *i.e.* a developer with no co-change expertise and less than 6 months of experience on the project, our suggestion is correct 19% and 30% of the time for Avaya and Chrome. Combining the new developer and the expert developer scenarios, we make a correct recommendation 67% and 64% of the time for Avaya and Chrome.

The control case of a random available developer is tied to the size of the project, with the smaller Avaya project outperforming the larger Chrome project. Our expertise measure substantially outperforms the control case. For Avaya where there are fewer new developers than Chrome, we can see that the expertise measure is more accurate. The Chrome project uses many new developers with 30% of our suggestions leading to a new developer taking over maintenance.

5.3 Successor Risk Mitigation

Our successor measure is a simple modification of Cataldo *et al.*'s [8] coordination requirement measure. As a result, we can provide a preliminary test of the conjecture that teams that have more collaboration among developers have a reduced risk of knowledge loss from turnover. To quantify and test the truth of this statement, we exclude abandoned files that we correctly predicted a successor from the loss distribution. These files were not at risk to abandonment because even when the owning developer left, there was a successor who had related knowledge on the project who could perform maintenance tasks. The successor loss distribution shows a reduction in the size of ES by 4% and 7% for Avaya and Chrome compared to the actual loss distribution (See Figure 3 and 4 in Section 3). The corresponding reduction in expect loss is 7% and 15%. For the median loss the reduction is 26% and 25%. Although the unexpected loss sees a moderate reduction in size, the median loss sees a substantial reduction. We suspect that the reason why the improvement is less pronounced in ES is large losses occur because a developer hoards a file and does not have any potential successors when he or she leaves. Our preliminary results suggest that reducing the amount of hording can drastically reduce the risk of loss, as evident in the difference between ES and median loss. This result is promising for future work because a simple successor measure based on co-changed files can reduce risk of turnover when there are potential successors. Future work is necessary to propose advanced successor suggestion algorithms and to provide managers with a clear understanding of the risk reduction of discouraging hording and encouraging co-development and collaboration. This work could also include measures of how central a file is to the project [30] and how many many developers know each of the languages that the system is written in [27].

6. THREATS TO VALIDITY

We have examined only two large projects so it is unclear whether our findings will generalize to other settings. We have, however, introduced a risk assessment methodology that can be applied to other large projects that must keep track of large turnover risks. We explicitly excluded small projects as they usually depend on a single or small number of developers. In this case, there is no need to quantify the risk as it is already clear that if any of these developers leave the project will fail. This size limitation is apparent in the previous works on turnover and in particular the 'truck factor' [24, 29, 26, 10].

Our analysis depends on accurate file ownership information. Both projects in our study were migrated from another version control system at some point in their history. The ownership information is removed when the migration occurs. The `blame` function will attribute the entire system to the developer who made the migration commit. Since the attribution is incorrect on these lines of code, we exclude them from our analysis. We also begin our analysis two years after a migration to ensure that we are studying a substantial portion of the codebase.

We consider a file to be abandoned when more than 90% of the lines in the file have no owner. Other works in code ownership have used similar percentages, such as Bird *et al.* [4]. Previous works on developer turnover have considered a file to be abandoned only when all lines are abandoned [26]. On OSS projects we feel that this is a too liberal definition of ownership as many minor contributors have contributed a few lines to a file, but have much less knowledge than a core developer who may have contributed hundreds of lines to the same file. Compared to these previous works our work is more conservative (*i.e.* our loss may be larger), which is appropriate when accessing uncommon events like large unexpected losses.

7. CONCLUSIONS

The tight relationship between the author and the authored source code makes software development susceptible to knowledge loss when authors leave the project and abandon their code. The newcomers who replace them tend to be less productive and more prone to making errors modifying an unfamiliar codebase. This, in conjunction with modern business practices that do not invest in worker tenure and globalization with its tendency to move work to low-cost locations result in an environment where developer turnover is so high that it may pose substantial risks for project survival [20].

Risks in software projects are often difficult to quantify and the impact and probability of the risks are left for developers to estimate. For example, a common technique is to give each project risk a ranking on a ten point scale from 1, unlikely, to 10, very likely [28]. In contrast to these rough estimates, in this work, we have adapted the financial risk management measures (value at risk) to a developer turnover context (knowledge at risk) and quantified the risk from turnover through the historical knowledge loss distribution. We found large unexpected losses to be 3.8 and 3.6 times larger than the expected past losses for Avaya and Chrome, respectively.

A historical simulation based on the loss distribution and the number of leaving developers discovered that the losses could be even larger with simulated large losses being 5.3 and 5.7 times larger than the actual expected loss for Avaya and Chrome, respectively.

We ran a Monte Carlo simulation to examine the likelihood of disaster or 'truck factor' scenarios. We found that given an unexpectedly large loss, it may be necessary to replace from 10 to 19 core developers for Chrome and from 11 to 33 developers for Avaya. However, the extreme losses suggested by the 'truck factor' simulations were unrealistically high with a likelihood of occurrence of less than 1%. For example, the threshold suggested by Torchiano *et al.* [26] would require the departure of more than 200 Chrome developers within a single quarter. This 'truck factor' threshold would need to be renamed the 'airplane factor' on large projects.

By modifying Cataldo *et al.*'s coordination requirements matrix we are able to predict the developer with the most related expertise as a successor. This simple preliminary approach allowed us to correctly predict the successor who would perform maintenance 48% and 34% percent of the time for Avaya and Chrome. Recalculating the loss distribution to include these successors as co-owners, we found that having readily available successors on a project reduces the risk of expected loss from turnover by 7% and 15% for Chrome and Avaya.

Obtaining a detailed turnover risk profile is a first step in increasing the understanding of knowledge loss risk. The identification of the parts of the code that are most vulnerable to knowledge loss could help newcomers to prioritize their contributions and help mentors with successor training. It could lead to new approaches that increase the resilience of a project against knowledge losses. The proposed affordances created by the ability to objectively and easily identify developers and code that pose high risk can help projects and contributors take actions improving the chances of project survival even in cases of rare adverse events.

8. REFERENCES

[1] M. Armstrong-Stassen. Coping with downsizing: A comparison of executive-level and middle managers. *International Journal of Stress Management*, 12(2):117–141, 2005.

[2] G. Beenen, K. Ling, X. Wang, K. Chang, D. Frankowski, P. Resnick, and R. E. Kraut. Using social psychology to motivate contributions to online communities. In *Proceedings of the 2004 ACM conference on Computer supported cooperative work*, 2004.

[3] C. Bird, A. Gourley, P. Devanbu, A. Swaminathan, and G. Hsu. Open borders? immigration in open source projects. In *MSR: Proceedings of the Fourth International Workshop on Mining Software Repositories*, page 8. IEEE Computer Society, 2007.

[4] C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. Devanbu. Don't touch my code!: Examining the effects of ownership on software quality. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ESEC/FSE '11, pages 4–14, New York, NY, USA, 2011. ACM.

[5] B. W. Boehm. Software risk management: Principles and practices. *IEEE Softw.*, 8(1):32–41, Jan. 1991.

[6] L. Briand, W. Melo, C. Seaman, and V. Basili. Characterizing and assessing a large-scale software maintenance organization. In *Proceedings of the 17th International Conference on Software Engineering*, ICSE '95, pages 133–143, New York, NY, USA, 1995. ACM.

[7] G. Canfora, M. Di Penta, R. Oliveto, and S. Panichella. Who is going to mentor newcomers in open source projects? In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, page 44. ACM, 2012.

[8] P. A. Cataldo, Marcelo an dWagstrom, J. D. Herbsleb, and K. M. Carley. Identification of coordination requirements: implications for the design of collaboration and awareness tools. In *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, CSCW'06, pages 353–362, NewYork,NY,USA, 2006. ACM.

[9] W. G. Cochran. *Sampling Techniques.* John Wiley, 1963.

[10] V. Cosentino, J. Izquierdo, and J. Cabot. Assessing the bus factor of git repositories. In *Software Analysis, Evolution and Reengineering (SANER), 2015 IEEE 22nd International Conference on*, pages 499–503, March 2015.

[11] S. M. Donadelli. The impact of knowledge loss on software projects: turnover, customer found defects, and dormant files. Master's thesis, Concordia University, 2015.

[12] M. Foucault, M. Palyart, X. Blanc, G. C. Murphy, and J.-R. Falleri. Impact of developer turnover on quality in open-source software. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, pages 829–841, New York, NY, USA, 2015. ACM.

[13] T. Fritz, G. C. Murphy, E. Murphy-Hill, J. Ou, and E. Hill. Degree-of-knowledge: Modeling a developer's knowledge of code. *ACM Trans. Softw. Eng. Methodol.*, 23(2):14:1–14:42, Apr. 2014.

[14] A. Hindle, D. M. German, and R. Holt. What do large commits tell us?: a taxonomical study of large commits. In *Proceedings of the 2008 international working conference on Mining software repositories*, pages 99–108. ACM, 2008.

[15] D. Izquierdo-Cortazar, G. Robles, F. Ortega, and J. Gonzalez-Barahona. Using software archaeology to measure knowledge loss in software projects due to developer turnover. In *System Sciences, 2009. HICSS '09. 42nd Hawaii International Conference on*, pages 1–10, 2009.

[16] P. Jorion. *Value at risk: the new benchmark for managing financial risk*, volume 3. McGraw-Hill New York, 2007.

[17] D. Joseph, K.-Y. Ng, C. Koh, and S. Ang. Turnover of information technology professionals: A narrative review, meta-analytic structural equation modeling, and model development. *MIS Quarterly*, 31(3):547–577, 2007.

[18] A. McNeil, R. Frey, and P. Embrechts. *Quantitative risk management*, volume 10. 2005.

[19] A. Mockus. Succession: Measuring transfer of code and

developer productivity. In *Proceedings of the 31st International Conference on Software Engineering*, ICSE '09, pages 67–77, Washington, DC, USA, 2009. IEEE Computer Society.

[20] A. Mockus. Organizational volatility and its effects on software defects. In *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*, FSE '10, pages 117–126, New York, NY, USA, 2010. ACM.

[21] A. Mockus and J. Herbsleb. Challenges of global software development. In *Software Metrics Symposium, 2001. METRICS 2001. Proceedings. Seventh International*, pages 182–184. IEEE, 2001.

[22] A. Mockus and J. D. Herbsleb. Expertise browser: a quantitative approach to identifying expertise. In *ICSE: Proceedings of the 24th International Conference on Software Engineering*, pages 503–512. ACM Press, 2002.

[23] PayScale. Companies with the Most & Least Loyal Employees, 2015. http://www.payscale.com/data-packages/employee-loyalty/least-loyal-employees.

[24] F. Ricca, A. Marchetto, and M. Torchiano. On the difficulty of computing the truck factor. In *Proceedings of the 12th International Conference on Product-focused Software Process Improvement*, PROFES'11, pages 337–351, Berlin, Heidelberg, 2011. Springer-Verlag.

[25] P. B. Tambe and L. M. Hitt. How offshoring affects it workers. *Communications of the ACM*, 53(10):62–70, 2010.

[26] M. Torchiano, F. Ricca, and A. Marchetto. Is my project's truck factor low?: theoretical and empirical considerations about the truck factor threshold. In *Proceedings of the 2nd International Workshop on Emerging Trends in Software Metrics*, WETSoM '11, pages 12–18, New York, NY, USA, 2011. ACM.

[27] B. Vasilescu, A. Serebrenik, and M. G. Brand. The babel of software development: Linguistic diversity in open source. In *Proceedings of the 5th International Conference on Social Informatics - Volume 8238*, SocInfo 2013, pages 391–404, New York, NY, USA, 2013. Springer-Verlag New York, Inc.

[28] L. Williams. Risk Management, 2004. http://agile.csc.ncsu.edu/SEMaterials/RiskManagement.pdf.

[29] N. Zazworka, K. Stapel, E. Knauss, F. Shull, V. R. Basili, and K. Schneider. Are developers complying with the process: An xp study. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '10, pages 14:1–14:10, New York, NY, USA, 2010. ACM.

[30] M. Zhou and A. Mockus. Developer fluency: Achieving true mastery in software projects. In *Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE '10, pages 137–146, New York, NY, USA, 2010. ACM.

[31] M. Zhou and A. Mockus. Does the initial environment impact the future of developers? In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 271–280, New York, NY, USA, 2011. ACM.